SUN/95.45

Starlink Project
Starlink User Note 95.45

Malcolm J. Currie & David S. Berry

2021 May 21

KAPPA — Kernel Application Package
2.6-12
User's Guide

Abstract

KAPPA is an applications package comprising about 180 general-purpose commands for image processing, data visualisation, and manipulation of the standard Starlink data format—the NDF. It is intended to work in conjunction with starlink's various specialised packages.

In addition to the NDF, KAPPA can also process data in other formats by using the 'on-the-fly' conversion scheme. Many commands can process data arrays of arbitrary dimension, and others work on both spectra and images. KAPPA operates from both the UNIX C-shell and the ICL command language.

This document describes how to use KAPPA and its features. There is some description of techniques too, including a section on writing scripts. This document includes several tutorials and is illustrated with numerous examples. The bulk of this document comprises detailed descriptions of each command as well as classified and alphabetical summaries.

Contents

List of Figures

1 Introduction

1.1 Background

It is Starlink's aim to provide *maintainable*, *portable*, and *extensible* applications packages that work in harmony by sharing a common infrastructure toolkit, standards, conventions and above all, a standard data format. Individual packages are no longer required to perform all functions, thus carry less inertia, and are more adaptable to outside developments. Additional functionality can be added piecemeal to the relevant package. New user interfaces, such as graphical, could be layered within the toolkit for obtaining parameters and so make the enhancement available to all applications that make use of those tools. An example of this approach has allowed us to access 'foreign data formats' throughout Starlink packages, because the packages use a common infrastructure library.

An important part of the rationalisation is that applications are unified by sharing the same basic data structure—the NDF (Extensible *n*-dimensional Data Format). This contains an *n*-dimensional data array that can store most astronomical data such as spectra, images and spectral-line data cubes. The NDF may also contain information like a title, axis labels and units, error and quality arrays, and World Co-ordinate System information. There are also places in the NDF, called *extensions*, to store any ancillary data associated with the data array, even other NDFs.

1.2 Rôle of KAPPA

The backbone of the applications packages is KAPPA (**K**ernel **AP**plication **PA**ckage). It provides general-purpose applications that have wide applicability, concentrating on image processing, data visualisation, and manipulating NDF components. KAPPA provides facilities that integrate with specialised Starlink packages such as those for CCD reduction (CCDPACK), stellar and galaxy photometry (PHOTOM, EXTRACTOR, PISA, ESP), spectroscopy (ECHOMOP, FIGARO), polarimetry (POLPACK, TSP), format conversion (CONVERT), *etc. Thus the functionality of* KAPPA *should not be regarded in isolation.*

In a wider context, KAPPA offers facilities not in IRAF, for instance handling of data errors, quality masking, a graphics database, availability from the shell, as well as more *n*-dimensional applications, widespread use of data axes, and a different style. It integrates with instrument packages developed at UK observatories. With the automatic data conversion and the availability of KAPPA and other Starlink packages from within the IRAF command language, you should be able to pick the best of the relevant tools from both systems to get the job done.

1.3 Functionality of KAPPA

1.3.1 Applications

Currently, KAPPA has over 200 commands that are available both from the UNIX C-shell and from the ICL command language. They provide the following facilities for data processing:

- FITS readers that generate NDFs and text tables, and the import and export of ancillary data through the NDF FITS extension;

- generation of test data, and NDF creation from text files;
- setting and examining NDF components;
- world co-ordinate systems, and calculation of a sky co-ordinate system;
- arithmetic including a powerful application that handles expressions;
- pixel and region editing, including polygons and circles; re-flagging of bad pixels by value or by median filtering; and pasting arrays over others;
- masking of regions, and of pixels whose variances are too large;
- configuration change: flip, rotate, shift, reshape, subset, permute axes change dimensionality;
- normalisation of NDF pairs;
- compression and expansion of images;
- generalised resampling of NDFs using arbitrary transformations;
- mosaic creation;
- filtering: box, Gaussian, and median smoothing; very efficient Fourier transform, convolution;
- deconvolution: maximum-entropy, Lucy-Richardson, Wiener filter;
- surface and trend fitting;
- statistics including ordered statistics, histogram; pixel-by-pixel statistics over a sequence of images;
- inspection of image values;
- centroids of features, particularly stars; stellar PSF fitting;
- detail enhancement using histogram equalisation and Laplacian, convolution, edge enhancement via a shadow effect, thresholding;
- calculation of polarimetry images;
- creation of one-dimensional profiles through n-dimensional data sets; and
- conversion between various forms of complex data.

There are also many applications for data visualisation:

- use of the graphics database, AGI, to pass information about pictures between tasks; tools for the creation, labelling, selection of pictures, and obtaining co-ordinate information from them;
- image plots with a selection of scaling modes and many options such as axes;

- creation, selection, saving and manipulation of colour tables and palettes (for axes, annotation, coloured markers and borders);
- line graphics: contouring; histogram; line plot of one-dimensional arrays, multiple-line plot of images, and a grid of line plots for cubes; pie sections, and slices through an image; vector plot of an image.
- many aspects of the appearance of line graphics can be tailored to individual needs and stored within 'style files'.

1.3.2 General

KAPPA handles bad pixels, and processes quality, variance, World Co-ordinate System (WCS), and other information stored within NDFs (SUN/33 and Section 8). In order to achieve generality KAPPA does not process non-standard extensions; however, it does not lose non-standard ancillary data since it copies extensions to any NDFs that it creates. The standard extensions that KAPPA recognises are the FITS airlock, that holds metadata in the form of FITS headers; and PROVENANCE that records the lineage of an NDF, much like a family tree.

KAPPA can also process data in other formats, such as FITS and IRAF, using an automatic-data conversion facility (CONVERT, SSN/20).

Although oriented to image processing, many commands will work on NDFs of arbitrary dimension and others operate on both spectra and images, and cubes. Many applications handle all non-complex data types directly, for efficient memory and disc usage. Those that do not will usually undergo automatic data conversion to produce the desired result.

KAPPA's graphics are produced using the widely used PGPLOT package, and are thus device independent.

Most commands can be automatically re-invoked to process multiple NDFs by supplying a group of NDFs as input. Groups of NDFs can be specified using wild-cards, or by listing them explicitly either in response to a prompt or within a text file.

1.4 This document

This document is arranged as follows. First are two annotated KAPPA tutorials to give you a quick summary of basic usage. The main text follows, which amplifies the points sketched in the demonstrations, and describes other functionality and modes of use illustrated with further examples. Finally, there are extensive appendices, including a classified list of commands and detailed descriptions of each command, which are also available on a quick-reference card.

2 Tutorials

So the facilities summarised in the introduction sound appealing. Now you want to know how to access them, but the thick manual looks daunting. Actually, most of this manual comprises descriptions of each application. The best way to learn the basics is to try some example sessions.

Login to a colour workstation or X-terminal. Then enter the commands following the prompts shown below. The % is the shell prompt string, which you don't type. As we go along there will be commentary explaining what is happening and why. Let's begin.

2.1 From the C-shell

```
% kappa
```

This defines C-shell aliases for each KAPPA command, includes the help information, and shows the version number. It need only be issued once per login session. Thus you will see

```
KAPPA commands are now available -- (Version 1.0)

Type kaphelp for help on KAPPA commands
```

Let's run a KAPPA application. CADD adds a scalar constant to an NDF file—the Starlink standard data format—to make a new NDF file (usually called an *NDF* for short). In this case ten is added to the pixels in `$KAPPA_DIR/comwest.sdf` to create `test.sdf` in the current directory.

```
% cadd $KAPPA_DIR/comwest 10 test
```

There are three *parameters* qualifying the CADD command: the names of the input and output NDFs and the constant. Notice that these parameters are separated by spaces. Most applications have a few of these positional parameters, usually the most commonly used. Parameters given on the command line are not subsequently prompted for by the application. Also you see that the NDF file extensions are not given. The `.sdf` extension indicates that it was created by the Hierarchical Data System (SUN/92), or HDS for short. Note that an arbitrary `.sdf` file is not necessarily an NDF.

Next we run the statistics task. Here we have not given any parameters. In this case the application will ask for the values of any parameters it needs.

```
% stats
```

The only parameter required is called NDF, and STATS prompts us for it.

```
NDF - Data structure to analyse /@test/ >
```

In this example, STATS wants to know for which NDF we require statistical data. The text between the // delimiters is the suggested default for the parameter. By pressing the carriage return we accept this default as the parameter's value. Here the suggested default is the name of the NDF created by CADD. (Ignore the `@` for the moment—it just tells the application that it is a file.) KAPPA remembers the last NDF used or created, and uses it for the suggested default to save typing. Since `test` is the NDF whose statistics we want we just hit the return key. Again we exclude the `.sdf` extension. Here is the output from STATS.

```
Pixel statistics for the NDF structure /home/scratch/mjc/test

   Title                     : Comet West, low resolution
   NDF array analysed        : DATA

      Pixel sum              : 11851773
      Pixel mean             : 180.8437
      Standard deviation     : 63.47324
      Minimum pixel value    : 13.89063
         At pixel            : (59, 83)
         Co-ordinate         : (58.5, 82.5)
      Maximum pixel value    : 255.9375
         At pixel            : (248, 45)
         Co-ordinate         : (247.5, 44.5)
      Total number of pixels : 65536
      Number of pixels used  : 65536 (100.0%)
```

Of course, in your case the current directory will not be `/home/scratch/mjc`. The NDF title is the unchanged from the `$KAPPA_DIR/comwest` NDF. This is the normal behaviour for tasks that create a new NDF from an old one; they do, however, have a parameter for changing this default. To alter a defaulted parameter you supply its new value on the command line. Defaulted parameters exist to prevent a long series of prompts where reasonable values can be defined, and hence save time. (However, there is a way of being prompted for all parameters of a command should you wish.)

NDFs may contain three standard arrays—the data array, the data variance and quality. STATS can calculate statistics for any of these. By default, STATS uses the data array, as indicated here.

Next we wish to smooth our data. GAUSMOOTH performs a Gaussian smooth of neighbouring pixels.

```
% gausmooth
```

Again we are prompted with the same suggested default, since we have not created any new NDFs within STATS. Say we don't want to smooth that NDF, but the original one. We just enter the name of the NDF at the prompt. Notice that we don't need the `@` prefix, since Parameter IN expects a file. (One occasion where you would need it is when the filename is a number, *e.g.* if your NDF was called 234 you must enter `@234`, otherwise the parameter system will think you are giving the integer 234. Yes . . . I know . . . the parameter system is trying to be too clever.)

```
IN - Input NDF /@test/ > $KAPPA_DIR/comwest
```

The description of Parameter FWHM is too brief for us to select a value. So we obtain some help on this parameter, and then GAUSMOOTH reprompts for a value. The smoothed NDF is written to the NDF called `testsm` in the current directory.

```
FWHM - Gaussian PSF full-width at half-maximum /5/ > ?

GAUSMOOTH

  Parameters

    FWHM

      FWHM() = _REAL (Read)
         This specifies whether a circular or elliptical Gaussian
         point-spread function is used in smoothing a two-dimensional
         image.  If one value is given it is the full-width at
         half-maximum of a one-dimensional or circular Gaussian PSF.
         (Indeed only one value is permitted for a one-dimensional
         array.)  If two values are supplied, this parameter becomes the
         full-width at half-maximum of the major and minor axes of an
         elliptical Gaussian PSF.  Values between 0.1 and 10000.0 pixels
         should be given.  Note that unless a non-default value is
         specified for the BOX parameter, the time taken to perform the
         smoothing will increase in approximate proportion to the
         value(s) of FWHM.  The suggested default is the current value.
         will increase in approximate proportion to the value of FWHM.

FWHM - Gaussian PSF full-width at half-maximum /5/ >
OUT - Output NDF > testsm
```

Next we want to look at the result of our image processing. The first thing to do is to select an graphics device. The `xwindows` device becomes the current graphics device and remains so until the next GDSET command. (You may need to enter the `xdisplay` command (SUN/129) to redirect the output from the host computer to the screen in front you.)

```
% gdset xwindows
```

Now we actually display it on the screen. Some applications have many parameters, and it would be impractical to have to specify all those preceding the ones we wanted to alter. The solution is to specify the parameter by keyword. Here we have requested that the scaling of the data values to colour indices within the graphics device uses the current percentile range. Note that you may abbreviate the options of a menu, such as offered by Parameter MODE, to any unambiguous string.

```
% display mode=pe accept
Data will be scaled from 78.38278 to 235.3536.
```

If you have just created the window, the image will not look much like the comet, because the existing colour table is poor. If we replace the table with a grey-scale ramp from white to black,

```
% lutneg
```

what happens depends on your workstation hardware and settings. If your graphics system is set to 256 colours (technically, an *8-bit pseudo-colour visual*), then the effects of the above LUTNEG command will be immediately visible, and you will see a blurred image of the ubiquitous Comet West on the screen. If, on the other hand, your graphics system is set to 16-bit or 24-bit graphics, then the effects of the LUTNEG command will only become visible when you next display an image. In this case, re-invoking the above DISPLAY command will make the image appear correctly with the requested grey-scale colour table.

The ACCEPT keyword is a very useful feature. It tells an application to accept all the suggested defaults. In this case DISPLAY uses the current NDF and scales between the current percentile limits—10 and 90. The keyword can be abbreviated to double backslash from the shell. Aside: the parameter system actually requires a single backslash. From the shell, however, backslash is a *metacharacter*, and so must be 'escaped' to treat the character literally. One way is to place a `\` before each metacharacter. You can escape a series of characters by placing them inside single quotes `' '`. Other metacharacters to watch out for when using KAPPA include `[]()\"*?$`.

Next we want to make the image colourful. There are a number of predefined lookup tables, or you may create and modify your own. Here we've given the X-window a 'warm' brown-yellow colour table[1]:

```
% lutwarm
```

If you are not happy with this colour table, you may want to explore a wider range of colour tables using the LUTEDIT command which provides a complete graphical user interface for manipulating and viewing colour tables.

```
% lutedit image=$KAPPA_DIR/comwest
```

2.2 From ICL

ICL is a command language designed for use with Starlink applications, such as KAPPA. It is now of some antiquity but is still in use. The main advantages for the KAPPA user are that shell metacharacters like `[]()\"` need not be escaped; command names may be abbreviated; far fewer executables need be loaded, and therefore it is slightly faster than using the shell when you want to invoke more than a few commands on a busy system; there is a wide selection of intrinsic functions and floating-point arithmetic; and results may be passed between applications via ICL variables. However, in these two demonstrations the command languages are interchangeable apart from the accept backslash.

Let's start the second example.

```
% icl
```

[1] Again, you will need to re-display the image to see the effects of this command unless your graphics system is set to 256 colours.

This starts ICL. System, local and user-defined ICLlogin files are invoked. Here there is only a system login procedure which sets up help on Starlink packages, and commands for setting up definitions for those packages. One of those commands is `kappa`; it is analogous to the `kappa` command from the shell. We enter it after receiving the ICL prompt.

You should see something like the following.

```
ICL (UNIX) Version 3.1-9 14/02/2000

Loading installed package definitions...

   - Type HELP package_name for help on specific Starlink packages
   -   or HELP PACKAGES for a list of all Starlink packages
   - Type HELP [command] for help on ICL and its commands

 ICL> kappa

    KAPPA commands are now available (Version 1.0).

    Type ‘help kappa’ or ‘kaphelp’ for help on KAPPA commands.
```

Now we run an application, ADD, that adds the pixels in `$KAPPA_DIR/comwest.sdf` to those in `$KAPPA_DIR/ccdframec.sdf`. Although these images have different dimensions, the intersection is made.

```
 ICL> add $KAPPA_DIR/comwest $KAPPA_DIR/ccdframec
```

After the first KAPPA command is issued you'll see an arcane message like this.

```
Loading /star/bin/kappa/kappa_mon into kappa_mon11601 (attached)
```

It just tells you that the KAPPA monolith is being loaded. You'll see similar messages for each of the three monoliths when they are first wanted.

Since we did not give the name of the destination NDF that will hold the co-added NDFs, ADD prompts for it. Notice that literal parameters are case insensitive.

```
 OUT - Output NDF / / > demo1


 ICL> ndftrace \

    NDF structure /home/soft2/mjc/alpha_OSF1/kappa/package/demo1:
       Title:  Comet West, low resolution

    Shape:
        No. of dimensions:  2
        Dimension size(s):  256 x 256
        Pixel bounds     :  1:256, 1:256
        Total pixels     :  65536
```

```
   Data Component:
       Type        :  _REAL
       Storage form:  PRIMITIVE
       Bad pixels may be present
```

This shows that the demo1 NDF has the same dimensions as the smaller of the two NDFs.

We were going to display the image on the current graphics device, but then changed our minds. A !! in response to a prompt aborts a task.

```
ICL> display demo1
MODE - Method to define the scaling limits /'PERCENTILES'/ > !!
!! SUBPAR: Abort (!!) response to prompt for Parameter MODE
OBEYW unexpected status returned from task "kapview_mon11601", action
- "DISPLAY"
ADAMERR  %PAR, Parameter request aborted
```

KAPPA uses the graphics database, which records the positions and extents of graphs and images, collectively called pictures.

```
ICL> picgrid 2 1
```

This instruction divides the display surface into two equally sized pictures, side by side. They are labelled 1 and 2 in the database. Picture 1 is the current picture, in which future pictures are drawn, unless we select a new current picture.

Thus in Picture 1 we display an image of Comet West around which we draw annotated axes. The backslash causes the current scaling method to be used.

```
ICL> display comwest axes \
!! Error accessing file '/home/scratch/mjc/comwest.sdf' -
!     No such file or directory
!  HDS_OPEN: Error opening an HDS container file.
!  NDF_ASSOC: Unable to associate an NDF structure with the '%IN' parameter.
```

DISPLAY could not find the a comwest.sdf in the current directory. So there is an error message and we are prompted. This time we remember to add the environment variable.

```
IN - NDF to be displayed /@comwest_bas/ > $KAPPA_DIR/comwest
Data will be scaled from 67.46276 to 226.5568.
```

An image of Comet West should be visible to the left of the screen.

SHADOW creates an image that appears like a bas-relief. We've called the resulting NDF comwest_bas. The backslash causes the current NDF to be the input NDF for SHADOW.

```
ICL> shadow out=comwest_bas \
```

We select the right-hand picture created earlier.

```
ICL> picsel 2
```

As above we display the current NDF, the bas-relief image, with annotated axes on the right of the raw comet image (we do not need to request the axes explicitly this time, since the `axes` parameter retains the value used in the previous invocation until changed).

```
ICL> display border \
Data will be scaled from -4.721756 to 5.697861.
```

The relief looks best with a grey-scale colour table. Note that this does not affect the colour of the border. LUTGREY is a procedure which calls a more-general application. Since it is the first procedure we've invoked there is a short pause while all the KAPPA procedures are compiled and loaded.

```
ICL> lutgrey
Loading procedure file $KAPPA_DIR/kappa_proc.icl
```

Next we decide to make a hard copy of the bas-relief image. DISPLAY does this and can add a key of grey levels and their corresponding values. The chosen device is `ps_l`; this overrides the `xwindows` device for the duration of DISPLAY. If this name isn't recognised at your site, issue the GDNAMES command for a list of your local device names. Select the landscape PostScript device. We scale between wider limits to reduce the glare.

```
ICL> display key=yes device="/PS"
IN - NDF to be displayed /@comwest_bas/ >
MODE - Method to define the scaling limits /'SCALE'/ >
LOW - Low value for display /187.5625/ > 11
HIGH - High value for display /-183.453125/ > -8.33
```

DISPLAY does not send your plot to the printer, since this is hardware and node dependent. Therefore, you must issue a shell command from ICL to perform this action. That's not difficult—just insert a `!` before the UNIX command, and in most cases just issue the command as if you were in the shell, like we do below.

```
ICL> !lpr -P1 pgplot.ps
```

Shell aliases may also be used, so if `ri` equates to `rm -i`, we could remove any unwanted HDS files. If you don't have this symbol, as is likely, then you will receive the appropriate error message from ICL.

```
ICL> ri *.sdf
```

That's the end of the second demonstration. Of course, these introductions have only scratched the surface of what KAPPA can do for you. You should look at Appendix A to search for the desired function, and then find more details in Appendix C.

If you get stuck or something untoward happens, there is a `Hints` help topic.

3 Getting started

3.1 Running KAPPA

KAPPA runs from the C-shell and variants, and also from the interactive command language—ICL. Both run monolithic programmes for efficiency. Both have their advantages and disadvantages. Of the latter, the shell forces you to escape certain characters, and ICL does not have a `foreach` to loop through a wildcarded list of NDFs. You may simply prefer the familiar shell to ICL, though UNIX commands, including editing, are accessible from ICL via a `!` prefix. This is not the soapbox to expound the intrinsic merits of the two command languages, but where there are differences affecting KAPPA, they'll be indicated. The choice is yours.

To run KAPPA from the shell just enter the following command.

```
% kappa
```

This executes a procedure setting up aliases for KAPPA's command names and to make help information available. Then you'll be able to mix KAPPA commands with the familiar shell ones.

If the `kappa` command is not recognised, you probably haven't enabled the Starlink software. In your `.cshrc` or `.tcshrc` file, you insert the line

```
source /star/etc/cshrc
```

and in `.login` you include the equivalent line

```
source /star/etc/login
```

At non-Starlink sites the `/star` path may be different.

To run KAPPA from ICL you have to start up the command language if you are not already using it. This requires just one extra command, namely

```
% icl
```

You will see any messages produced by system and user procedures, followed by the `ICL>` prompt. Again there is a procedure for making the commands known to the command language, and not unexpectedly, it too is

```
ICL> kappa
```

Then you are ready to go. Not too painful, was it? In either case you'll see message from KAPPA telling you which version is ready for use.

So what do you get for your trouble? Appendix **??** lists in alphabetical order all the commands and their functions, and Appendix A is a classified list of the same commands. Many examples are given in subsequent sections.

3.2 Issuing Commands

To run an application you then can just give its name—you will be prompted for any required parameters. Alternatively, you may enter parameter values on the command line specified by position or by keyword. More on this in Section 4.

Commands are interpreted in a case-independent way from ICL, but from the shell they must be given in lowercase. In ICL, commands may also be abbreviated provided they are unambiguous strings with at least four characters. Commands shorter than five characters, therefore, cannot be shortened. So

```
ICL> CREF
ICL> crefr
ICL> CreFra
ICL> CREFRAM
```

would all run CREFRAME. Whereas

```
ICL> FITS
ICL> FITSI
```

would be ambiguous, since there are several commands beginning `FITS`, and two starting `FITSI`, namely FITSIN and FITSIMP.

Note if other packages are active there is the small possibility of a command-name clash. Issuing such a command will run that command in the package last activated. You can ensure running the KAPPA command by inserting a `kap_` prefix before the command name. For example,

```
% kap_rotate
```

will execute KAPPA's ROTATE application. There may also be a clash with UNIX commands and shell built-in functions, though there are now far fewer conflicts than in earlier versions of KAPPA, with only **look** being ambiguous. There is also a **glob** in the C-shell which might confuse you should you forget that GLOBALS cannot be abbreviated from the shell.

Since the KAPPA commands are the same in both the shell and ICL, the `%` and `ICL>` prompts in the examples and description below are interchangeable unless noted otherwise.

3.3 Obtaining Help

3.3.1 Hypertext Help

A modified version of this document exists in hypertext form. One way to access it is to use the `showme` command

```
% showme sun95
```

and a Web browser will appear, presenting the index to the hypertext form of this document. The hypertext permits easy location of referenced documents and applications. It also includes colour illustrations.

The `findme` command lets you search the Starlink documents by keywords. For instance,

```
% findme masking
```

searches the document looking for the word 'masking' in them. The level of searching depends on whether a match is found. The search starts with the document title, the page (section) titles, and finally the document text. The deeper the search, the longer it will take. There are switches provided to limit the level of the search. The search string may include **sed** or **grep** regular expressions. See SUN/188 or enter

```
% findme findme
% findme showme
```

to learn more about the `findme` and `showme` commands.

3.3.2 Entering the Help System

To access the KAPPA help use KAPHELP.

```
ICL> kaphelp
```

The system responds by introducing KAPPA's help library, followed by a long list of topics for which help is available, followed by the prompt `Topic?`. These topics are mostly the commands for running applications, but they also include global information on matters such as parameters, data structures and selecting a graphics device.

From ICL you can issue other commands for obtaining help about KAPPA.

```
ICL> help kappa
ICL> help packages
```

The former is nearly equivalent to entering `kaphelp`. However, it is less easy to use as it lacks many of the navigational aids of KAPHELP. The latter gives a summary of Starlink packages available from ICL. If you select the `KAPPA` subtopic, you'll get a precis of the package's facilities. (This is part of an index of Starlink packages.)

If you have commenced running an application you can still access the help library whenever you are prompted for a parameter. See Section 4.7 for details.

3.3.3 Navigating Help Hierarchies

The help information is arranged hierarchically. The help system enables you to navigate the library by prompting when it has either presented a screen's worth of text or has completed displaying the previously requested help. The information displayed by the help system on a particular topic includes a description of the topic and a list of subtopics that further describe the topic.

You can select a new topic by entering its name or an unambiguous abbreviation. If you press the carriage-return key (<CR>) you will either continue reading a topic where there is further text to show, or move up one level in the hierarchy. Entering a CTRL/D (pressing D whilst holding the CTRL key) terminates the help session. See the description of KAPHELP for a full list of the options available at prompts inside the help system, and the rules for wildcarding and abbreviating topics.

3.3.4 Help on KAPPA commands

Help on an individual KAPPA application is simply achieved by entering `kaphelp` followed by the command name, for example

```
% kaphelp centroid
```

will give the description and usage of the CENTROID command. There are subtopics which contain details of the parameters, including defaults, and valid ranges; examples; notes expanding on the description; implementation status; and occasionally timing. For example,

```
ICL> kaphelp hist param ndf
```

gives details of Parameter `NDF` in all applications prefixed by `HIST`.

(From ICL you can also invoke its help system, thus

```
ICL> help centroid
```

is similar to `kaphelp centroid`, though the ICL system has drawbacks, and you are recommended to run KAPHELP.)

The instruction

```
ICL> kaphelp classified
```

displays a list of subject areas as subtopics. Each subtopic lists and gives the function of each KAPPA application in that classification. There is also an alphabetic list which can be obtained directly via the command

```
ICL> kaphelp summary
```

3.4 Changing the Current Directory in ICL

You should change default directories in ICL using its DEFAULT command, and not **cd**. Thus

```
ICL> default /home/scratch/dro
```

makes `/home/scratch/dro` the default directory for the ICL session, and for existing and future subprocesses, including application packages.

3.5 Exiting an Application

In normal circumstances when you've finished using KAPPAnothing need be done from the shell, but to end an ICL session, enter the `EXIT` command to return to the shell.

What if you've done something wrong, like entering the wrong value for a parameter? If there are further prompts you can enter the abort code `!!` to exit the application. This is recommended even from the shell because certain files like your NDFs may become corrupted if you use a crude `CTRL/C`. If, however, processing of the data has begun in the application, it is probably best to let the task complete, unless it is a long job like image deconvolution. If you really must abort, `CTRL/C` should be hit. From ICL this ought to return you to a prompt, but the processing will continue. Then you can stop the running process by 'killing' it. First find the task name

```
ICL> tasks
                        TASKNAME  Process Id

                ndfpack_mon16528  15186
```

and then kill it.

```
ICL> kill ndfpack_mon16528
```

This removes a the NDFPACK monolith. NDFPACK will be loaded again once you enter one its commands. If pressing `CTRL/C` several times fails to return you to an ICL prompt then it's time for the heavy artillery—you may have to kill your window. Once back to the shell enter `icl` to return to ICL, and then kill the process as described above.

If you have interrupted a task, it may be necessary to delete the parameter file (Section 4.2) and the graphics database (Section 11.4).

4 Parameters

KAPPA is a command-driven package. Commands have *parameters* by which you can qualify their behaviour. Parameters are obtained in response to prompts or supplied on a command line.

For convenience, the main aspects of the parameter system as seen by a user of KAPPA are described below, but note that most of what follows is applicable to any Starlink application.

4.1 Summary

For your convenience, here is a summary of how to use parameters. If you want more information, go to the appropriate section.

Command-line values
: On the command line you can supply values by keyword or by position. See Section 4.10 for more details including abbreviated keywords.

`ACCEPT, PROMPT, RESET` **command-line special keywords**
: `ACCEPT` accepts all the suggested defaults that would otherwise be prompted. `PROMPT` prompts for all the parameters not given on the command line, and `RESET` resets all the suggested defaults to their initial values. You can find more details and examples in Section 4.11.

`NAME - Prompt string /Suggested default/ >`
: This is a schematic of a prompt. NAME is the parameter's name. You normally respond with the value for the parameter, but there are special responses available (see below). If you just hit the return key, the suggested default becomes the parameter value. Many parameters are defaulted without prompting. See Section 4.2 and Section 4.3 for more details.

Here is a list of some example parameter values to illustrate the possible ways you can respond to a prompt. Where there are command-line differences, they are noted.

`5409.12` This is a scalar. Numerical values can be integer, real, or double precision.

`12,34,56,78` This is a vector. They must be enclosed in `[ ]` if the array is supplied on the command line, or for character arrays.

`[[11,21,31],[12,22,32]]` This is a 3×2 array. Arrays of dimension > 2 should appear in nested brackets. See Section 4.5 for more about array values.

`T`

`no` This is a `TRUE` value followed by a `FALSE` values for logical parameters. Acceptable values are `TRUE`, `FALSE`, `YES`, `NO`, `T`, `F`, `Y`, `N` and their lowercase equivalents. On the command line, the parameter name as a keyword means `TRUE`. If the name is prefixed with `NO`, the value is set to `FALSE`.

`a string`

`"a string"` This is a string. Strings need not be quoted at prompts. Quotes are required on the command line if the string includes spaces or wildcards, or is a comma-separated array of strings. There is more in Section 4.4. Some parameters offer a selection from a menu to which you give an unambiguous abbreviation to select an option. Other parameters can be numerical or a string. (See Section 4.8 for more information.)

`filename`

`@123` This enters a filename (or tape drive). You give a text filename verbatim, and NDFs without the file extension. Foreign formats will usually have the file extension. Should the filename be a numerical value, it must be preceded by an `@`. There is more in Section 4.4.

`min`

`max` This selects the minimum- or maximum-allowed value, but not all parameters have a defined range of permitted values. See Section 4.12.

`!` Enters the null value. This has a variety of special meanings; which one will depend on the particular parameter. For example, null might indicate that an output file is not to be created, or a loop is to be ended. There are more examples in Section 4.6.

`!!` This aborts the application cleanly.

`?`

`??` A single question mark presents the online help for the parameter, and then reprompts. A double question mark leaves you in the help system to explore other help information. See Section 4.7 for examples. These special values are not supported from the command line.

`\` This accepts the suggested default for the prompted parameter and the respective suggested defaults for all subsequent parameters for which prompting would otherwise occur. On the command line `\` is an abbreviation of the `ACCEPT` keyword, and it applies to all parameters that would otherwise be prompted. Note that from the shell you write `\\`, as `\` is a shell metacharacter.

4.2 Defaults

Command-line values are used mostly for those parameters that are normally defaulted by the application. Defaulted parameters enable applications to have many options, say for controlling the appearance of some graphical output, without making routine operations tedious because of a large number of prompts. The values of normally defaulted parameters are given in Appendix C. You can also find them by obtaining online help on a specific parameter. They are enclosed in square brackets at the end of the parameter description.

```
ICL> kaphelp median param *
```

gives details of all parameters in application MEDIAN. Other packages have similar help commands. If you want to override one of these defaults, then you must specify the parameter's value on the command line.

When you are prompted you will usually be given a suggested default value in / / delimiters. You can choose to accept the default by pressing carriage return. For example, 64 is the suggested value below.

```
XDIM - x dimension of output array /64/ >
```

Alternatively, enter a different value

```
XDIM - x dimension of output array /64/ > 109
```

to override the default. Some defaults begin with an @.

```
IN - Input image /@starfield/ >
```

These are associated with files (text and HDS) and devices (graphics and tape). If you want to override the default given, you do not have to prefix your value with an @, *e.g.*

```
DEVICE - Name of display device /@xwindows/ > x2w
```

There are rare cases when the syntax is ambiguous, and it is then that you need to include the @. Section 4.4 describes when the @ is needed.

From both ICL and the shell the default value can be edited to save typing by first pressing the <TAB> key. The editor behaves like the shell command-line editor.

Defaults may change as data are processed. Often the current (last) value of the parameter will be substituted, or a dynamic value is suggested depending on the values of other parameters. Here is an example comprising a loop within an application.

```
NDF - NDF to examine /@horsehead >
CENTRE - Position at the centre of the listing /'64 64'/ > 100 120
   .              .              .              .
   .              .              .              .
   .              .              .              .
CENTRE - Position at the centre of the listing /'100 120'/ >
SIZE - The dimensions (in pixels) of the area to be listed /7/ >
   .              .              .              .
   .              .              .              .
   .              .              .              .
```

and so on. Notice that the current values of the centres are the suggested values at the second prompt.

Current values of parameters are stored in a *parameter file*, and so they persist between sessions. For tasks run from the shell, this is an HDS file $ADAM_USER/<*application*>.sdf, where <*application*> is the name of the application. (If the environment variable ADAM_USER is not defined the parameter file is situated in $HOME/adam).

Unfortunately, tasks invoked from ICL use a monolith parameter file, which contains the individual application parameter files for the members. So for example, KAPPA has `kappa_mon.sdf`, `kapview_mon.sdf`, and `ndfpack_mon.sdf` stored in the same directory as the individual files. This duality means that there are two independent sets of current values for each task depending on where you ran it from.

The parameter file should not be deleted unless the parameters values are to be completely reset, or the file has been corrupted in some way. If you suspect the latter case, say after interrupting a task(Section 3.5), run HDSTRACE (SUN/102) on the file to check its integrity.

4.3 Globals

KAPPA stores a number of global parameters that are used as defaults to reduce typing in response to prompts. Global means that they are shared between applications. The most common is the last dataset (usually NDF) written or accessed. In the example above, this was `horsehead.sdf`. If you just press `<CR>` in response to the prompt, the global value is unchanged. Only when you modify the parameter and the application completes without error is the global value updated.

All global parameters are stored in the HDS file `$ADAM_USER/GLOBAL.sdf`, or `$HOME/adam/GLOBAL.sdf` if the `ADAM_USER` environment variable is not defined. The full list is given below.

`GLOBAL.DATA_ARRAY` — Last NDF or foreign data file accessed or written.

`GLOBAL.GRAPHICS_DEVICE` — Current graphics workstation.

`GLOBAL.INTERACTIONMODE` — Current interaction mode.

`GLOBAL.LUT` — Last lookup table file accessed or written.

`GLOBAL.TRANSFORM` — Current transformation structure.

KAPPA uses the last DATA_ARRAY written or accessed as the suggested default value for the next prompt for an NDF structure or foreign data format. The same applies to the current lookup table and transformation structure. However, the remaining, including the graphics global parameter are defaulted—you will not be prompted. Details of how to control these parameters are given in the relevant sections.

The values of all global parameters may be inspected with the GLOBALS task. You can make them undefined using NOGLOBALS.

```
ICL> globals
The current data file                : @/home/dro/jkt/ccdpic
The current graphics device is       : @ps_l
The current lookup table file is     : @$KAPPA_DIR/spectrum_lut
The current transformation is        : @/home/dro/deform/warpit
The current interaction mode is      : <undefined>
The current message-report level is  : NORMAL
```

In the above example no interaction mode is defined. The next time you call an application that uses the interaction mode you would be prompted for a value. (Under normal circumstances you will not have to enter the `@` prefix yourself.)

Note that the message-reporting level is not a global parameter, but is defined by the MSG_FILTER environment variable. It controls the verbosity of informational messages. Since it applies to all tasks that report such messages, it is convenient to allow GLOBALS to show its value. Further details are in Section 5 for details. Since it is not a parameter as such, the message-reporting level is not unset by NOGLOBALS.

4.4 Strings

Notice that the single or double quotes around strings given in response to prompts for a character parameter can be omitted. However, on the command line these delimiters are needed if the string contains spaces, otherwise the second and subsequent words could be treated as separate positional parameters.

From the shell the quotes must be escaped. For example,

```
% settitle myndf \"A new title\"
```

would assign the title "A new title" to the NDF called myndf.

To indicate that the parameter should come from a data-structure object, prefix the name with an @ to tell the parameter system that it is a file name, and not a literal value.

```
TITLE - New NDF title /' '/ @$ADAM_USER/galaxy.mytitle
```

In this example TITLE has the value of object MYTITLE in galaxy.sdf. If the @ were omitted TITLE would be "$ADAM_USER/galaxy.mytitle". You will need the @ prefix if your file name is a number. Note that the file extension should not be included when giving the name of an HDS data file, including NDFs . Otherwise HDS will look for an object called SDF nested within the file.

Responses to prompts are case insensitive for comparison purposes. Strings for character components in NDFs, plot captions and labels are treated literally.

4.5 Arrays

If a parameter requires an array of values, the series should be in brackets separated by commas or spaces. For example,

```
PERCENTILES - List of percentiles > [25,95.5,75]
```

would input three values: 25, 95.5, and 75 into the real parameter PERCENTILES. If the application is expecting an exact number of values you will be reprompted, either for all the values if you give too many, or the remaining values if you supply too few. There is one exception where you can omit the brackets—a fairly common one—and that is in response to a prompt for a one-dimensional numeric array as above.

From the shell you must escape the brackets.

```
% display key=yes mode=pe percentiles=\[95,5\]
% display key=yes mode=pe percentiles='[95,5]'
% display key=yes mode=pe percentiles="[95,5]"
```

All the above do this. Single quotes have the advantage that you can protect all the metacharacters that lie between the quotes, so you don't need to escape each metacharacter.

Arrays of parameter values should appear in nested brackets. For example,

```
CVALUE - Component values > [[2,3],[5,4],[7,1]]
```

supplies the values for a 2×3-element parameter array, where element (1,3) has value 7.

4.6 Abort and Null

Responding to a prompt with a null value `!` will not necessarily cause the application to abort, but it can force a suitable default to be used, where this is the most-sensible action. A further use is when an optional file may be created, such as a lookup table; a `!` entered in response to the prompt for the filename means that no file is to be output. Many tasks use null as a default for optional files. In some applications, a null ends an interactive loop.

Responding to a prompt with `!!` will abort the application. This process includes the various tidying operations such as the unmapping and closing of files. Any other method of stopping an application prematurely can leave files mapped or corrupted.

4.7 Help

To get help about a parameter enter `?`. Usually, this will give access to the help-library information for that parameter, for example,

```
BOX - Smoothing box size /3,3/ > ?

BLOCK

  Parameters

    BOX() = _INTEGER (Read)
       The sizes (in pixels) of the rectangular box to be applied to
       smooth the data.  These should be given in axis order.  A value
       set to 1 indicates no smoothing along that axis.  Thus, for
       example, BOX=[3,3,1] for a three-dimensional NDF would apply a
       3x3-pixel filter to all its planes independently.

       If fewer values are supplied than the number of dimensions of
       the NDF, then the final value will be duplicated for the
       missing dimensions.

       The values given will be rounded up to positive odd integers, if
       necessary, to retain symmetry.

BOX - Smoothing box size /3,3/ >
```

and then immediately reprompt you for the parameter. There are occasions when information about the parameter is insufficient; you may require to examine the examples or the description of related parameters. This can be achieved by entering `??` to the prompt. You can then delve anywhere in the help information. When you exit the help system you're reprompted for the parameter.

4.8 Menus

Some parameters offer menus from which you select an option. You do not have to enter the full option string, but merely a string that selects a choice unambiguously. In many cases this can be as little as a single character. Here is an example,

```
MODE - Method for selecting contour heights /'Free'/ > ?
    The method used to select the contour levels.  The options are
    described below.

      "Area"      - The contours enclose areas of the array for
                    which the equivalent radius increases by equal
                    increments.  You specify the number of levels.
      "Automatic" - The contour levels are equally spaced between
                    the maximum and minimum pixel values in the
                    array.  You supply the number of contour
                    levels.
      "Free"      - You define a series of contour values
                    explicitly.
      "Linear"    - You define the number of contours, the start
                    contour level and linear step between contours.
      "Magnitude" - You define the number of contours, the start
                    contour level and step between contours.  The
                    step size is in magnitudes so the nth contour
                    is dex(-0.4*(n-1)*step) times the start contour
                    level.
```

where an `F` would be sufficient to select the `"Free"` option, but at least two characters would be needed if you wanted `"Area"` or `"Automatic"`.

Some parameters permit a mixture—a choice from a menu, or a numerical value within a range. The options are described in full in the help system and Appendix C.

4.9 Environment Variables

Environment variables operate both on the command line and prompts, and both from the shell and ICL. Thus if `IMAGEDIR` is an environment variable pointing to a directory containing the NDF called ngc1365, you could access it at a prompt as shown below.

```
IN - Input image /@starfield/ > $IMAGEDIR/ngc1365
```

4.10 Specifying Parameter Values on Command Lines

Parameters may be assigned values on the command line. This is useful for running tasks in batch mode and in procedures, and for specifying the values of parameters that would otherwise be defaulted. A command-line parameter will prevent prompting for that parameter unless there is an error with the given value, say giving an alphabetic character string where a floating-point value is demanded.

There are two ways in which parameter values may be given on the command line: by keyword and by position. The two forms may be mixed with care. The parser looks for positional parameters then keywords, so you can have some positional values followed by keyword values. See some of the examples presented in Appendix C.

4.10.1 Keyword

Keywords may appear in any order. Here is an example of command-line defaults using keywords.

```
ICL> picdef current fraction=0.4
```

FRACTION is a real parameter. CURRENT is a logical parameter; by giving just its name it is assigned the value `TRUE`. `CURRENT=T` would have the same effect. To obtain a `FALSE` value for a logical parameter you add a `NO` prefix to keyword, for example,

```
icl> picdef nocurrent
```

would be equivalent to the following.

```
icl> picdef current=false
```

4.10.2 Abbreviations

There is an experimental system that allows you to abbreviate parameter keywords to the minimum unambiguous length. To use it, you must first create an environment variable called `ADAM_ABBRV` with an arbitrary value.

So for example

```
% setenv ADAM_ABBRV true
% display mo=pe pe=\[5,95\] ba=blue
```

would display an NDF between the 5 and 95 percentiles, and marking bad pixels in blue.

If you give an ambiguous keyword, the parameter system will present the list of possible keywords and ask you to select the one you intended.

4.10.3 Position

Alternatively, you can specify command-line values by position. Here is an example.

```
% thresh raw clipped 0 255
```

This applies thresholds to the NDF called raw to form a new NDF called clipped. The values between 0 and 255 are unchanged. Note that trailing parameters may be omitted—NEWLO and NEWHI in the above example—but intermediate ones may not. The position of a parameter can be found in the `Usage` heading in Appendix C or the help for the application.

4.10.4 Keyword versus Positional Parameters

For tasks with a few parameters, using position is quick and convenient. However, in complex applications with many parameters it would be tedious not only to enter all the intermediate values between the ones you want to define, but also to remember them all. Another consideration is that some parameters do not have defined positions because they are normally defaulted. Keywords may also be abbreviated(Section 4.10.2). Thus the keyword technique is recommended for most parameters, especially in scripts and procedures. Unabbreviated keywords insulate scripts against new keywords and positional changes that are sometimes needed.

See Section 21 if you want to learn how further to abbreviate command strings to reduce typing for manual operation.

4.10.5 Special Behaviour

Sometimes specifying a parameter on the command line induces different behaviour, usually to inhibit a loop for procedures, or to eliminate unnecessary processing. For instance,

```
ICL> centroid blob init="51,42" mode=i
```

will determine the centroid near the point (51,42) in the NDF called blob, and then it exits, whereas without the INIT value you would be reprompted for a further initial position; and

```
% display galaxy mode=sc high=3000 low=1000
```

prevents the calculation of the extreme values of the NDF called galaxy that are normally given as suggested defaults for parameters HIGH and LOW, because HIGH and LOW are already known.

4.11 Special Keywords: ACCEPT, PROMPT, RESET

Another way in which prompts and default values can be controlled is by use of the keywords ACCEPT, PROMPT and RESET.

The RESET keyword causes the *suggested* default value of all parameters (apart from those already specified before it on the command line) to be set to the original values specified by the application or its interface file. In other words global and current values are ignored.

The PROMPT keyword forces a prompt to appear for every application parameter. This can be useful if you cannot remember the name of a defaulted parameter or there would be too much to type on the command line. However, it may prove tedious for certain applications that have tens of parameters, most of which you normally never see. You can abort if it becomes too boring.

The ACCEPT keyword forces the parameter system to accept the suggested default values either for every application parameter if the keyword appears on the command line, or all subsequent parameters if it is supplied to a prompt. In other words, those parameters that would normally be prompted with a value between / / delimiters take the value between those delimiters, *e.g.* XDIM we saw in Section 4.2 would take the value `64`. Parameters that are normally defaulted behave as usual. The ACCEPT keyword needs to be used with care in scripts because not every parameter has a default, and therefore must be given on the command line for the application to work properly. For example, CREFRAME must have a value specified for parameter OUT, the name of the output NDF. If we run the application like this:

```
ICL> creframe accept
```

it would fail in the sense that it would still have to prompt for a value—it does not know where to write the output NDF. However, if we run CREFRAME like this:

```
ICL> creframe out=stars accept
```

it would generate an output image using default values for all the parameters except OUT, and write the output to file `stars.sdf`. Another point to be wary of is that some applications have loops, *e.g.* LOOK, LUTABLE, and if you use the ACCEPT keyword it will only operate the first time the application gets a parameter value.

Sometimes the keyword ACCEPT can be used without any parameter value specifications on the command line. For example, we could follow the above command by the command:

```
ICL> look accept
```

and the central 7×7 array of the image created by CREFRAME would be displayed on your terminal without any parameter values being prompted. The symbol \ has the same effect as ACCEPT when used on the command line or at prompts, thus:

```
ICL> look \
```

would have the same effect as the previous example—and is quicker to type. In command lines from the shell, the backslash is a metacharacter and has to be escaped. The easiest way to do that is to double the backslash.

```
% look \\
```

How do you find out which parameters have suggested defaults, as opposed to those that are normally defaulted? Well, a good rule of thumb is that parameters for output files (images, lookup tables and text) will not have a default, but the remainder will. There are some exceptions, such as where null is the default for optional files. Consulting the description of the parameters should give the suggested defaults, where appropriate. If a parameter is given a suggested default it will have a line beginning `ppath` or a `default` line. If you want to use ACCEPT for an automatic procedure or batch job you could do some tests to find which parameters get prompted and then put them on the command line in your procedure.

The RESET and ACCEPT keywords will work in tandem. So for instance,

```
ICL> look reset accept
```

will reset the suggested defaults of LOOK to their original, preset values, and accept these as the parameter values.

These special keywords may be abbreviated to no fewer than two characters, if you have enabled keyword-abbreviation (Section 4.10.2).

4.12 MIN and MAX parameter values

Many parameters have well-defined ranges of allowed values. In some cases it is useful to assign the maximum or minimum value to the parameter. Rather than give some numerical value, you can instead supply `MIN` to select the minimum-allowed value, and `MAX` to select the maximum. This applies both on the command line and at prompts. In the example,

```
% block wlim=max
```

Parameter WLIM takes its maximum (1.0) meaning that if any of the input pixels in the smoothing box is bad, the corresponding output pixel is set bad.

Consult the reference section or the online help to see if a given parameter has such a range. If you attempt to use `MIN` and `MAX` where there is no range defined, you'll see an error message like

```
!! SUBPAR_MNMX: Parameter FONT - no upper limit set
```

and you'll be invited to give another value.

4.13 Specifying Groups of Objects

Some parameters are describing in the reference documentation as being associated with a *group* of objects. For instance, some parameters may require a group of strings, others may require a group of numerical values, or data files. No matter what the nature of the object, groups are specified using a syntax called a *group expression*. The GRP library is used to interpret these group expressions, and the GRP documentation (SUN/150), should be consulted for full details. A summary is given here.

A group expression can identify the members of the group in any of the following ways:

- As a comma-separated list (*e.g.* `"12.1, 23.2, 1.3"` or `"HH1_B1S1,HH2_B1S2"`).
- By reading them from a text file (see "Indirection").
- By modifying an existing group of objects using editing specified within the group expression (see "Modification").

A typical group expression will often include characters that are of significance to the shell. If the group expression is supplied on the command line it may be necessary to place quotes around the string to prevent the shell from removing these characters. Two lots of quotes are usually required, as in the following example where a group expression (in this case, a comma-separated list) is used to specify a plotting style:

```
% display style='"grid=1,colour(grid)=red,title=My new image"'
```

These quotes are not required if the group expression is given in response to a prompt.

If the supplied group expression is terminated with a hyphen, the user is re-prompted for another group expression (using the same parameter). The objects specified by this second group expression are added to those specified by the first. This re-prompting continues until a group expression is supplied that does not end with a hyphen.

Certain classes of objects have additional features, for instance if the objects are the names of data files, then wild-card characters are allowed in the supplied values.

4.13.1 Indirection

It is sometimes convenient to store the strings specifying the objects to be used within a text file. The name of the text file can then be given in response to a prompt for a group expression, rather than giving a long list of explicit values. This is done by preceding the name of the text file with an up-arrow (`"^"`) character. For instance, the group expression `"^style.dat"` would result in the file `style.dat` being opened and the strings read from the file. Each line within the file is considered to be a group expression, and is processed in the same way as a group expression supplied directly. In particular, a text file may contain references to other text files. If the file `style.dat` contained the following two lines:

```
grid=1,colour(grid)=red,border=1
colour(border)=red,^labels.dat
```

then the strings `grid=1`, `colour(grid)=red`, `border=1` and `colour(border)=red` would be returned to the application, and in addition the file `labels.dat` would be searched for further strings. This nesting of text files can go down to seven levels. Text files may also contain comments. Anything occurring after a `"#"` character is ignored. To ignore an entire line the # character must be in column 1 (any blanks in front of the # character are considered to be significant).

4.13.2 Editing

A group expression can contain a request to edit the supplied strings before passing them to the application. The editing facilities provided are fairly simple. You can:

- Add a specified prefix to the start of each string.
- Add a specified suffix to the end of each string.
- Replace all occurences of a given sub-string in each string.
- Any combination of the above.

To perform this editing, you:

(1) Enclose the group expression specifying the strings to be edited within curly braces (`"{"` and `"}"`). Note, if no prefix or suffix is supplied, and the group expression is not a comma-separated list, then the curly braces can be omitted.

(2) Precede the opening curly brace with the prefix (if any) to be added to the start of each string.

(3) Follow the closing curly brace with the suffix (if any) to be added to the end of each string.

(4) Append a string specifying the substitution to be performed (if any) to the end of the whole thing. This string should be of the form `|<old>|<new>|` where `<old>` is the text to be replaced and `<new>` is the text with which to replace it. Note, the substitutions occur *before* any specified prefix or suffix is added to the strings.

For instance;

```
A{^file}B|my|your|
```

This will read strings from the text file `file`. Each occurence of the string `"my"` will then be replaced by `"your"`. The resulting strings will then have `"A"` added at the start, and `"B"` added at the end.

4.13.3 Modification

A group of objects can be given by specifying some editing to apply to another already existing group of objects. For instance, if the string `new_*b|_ds|_im|` was given in response to a request for a group expression, then the following steps occur:

- Each element in some existing group of objects (identified in the description of the parameter concerned) is substituted in turn for the `"*"` character.
- Any occurrences of the string `"_ds"` is replaced by the string `"_im"`.
- The string `"b"` is added to the end of the string.

- The string "new_" is added to the start of the string.

Thus if the existing group contained the strings file1_ds and file2_ds, the resulting group would be new_file1_imb and new_file2_imb. Note, this facility is only available if the parameter description identifies an existing group which will be used as the basis for the modified strings.

4.13.4 Ignoring Syntax Characters

You can see from the above that several characters have special meanings within group expressions. Examples are ^ | -. This may sometimes cause a problem if you want to include these characters within the strings being passed to the application. For instance, if you want to specify a group of data files using a shell pipe-line, you may want to do something like:

```
% display
  IN - NDF to be displayed > ‘find . -newer a.fit | grep good ‘
```

Within a group expression, the | character indicates a request for a string substitution (as described above). In this case, the GRP library considers the request to be incomplete because there is only one | character, and issues an error report. Of course, the | character was actually intended to indicate that the output from the find command should become the input to the grep command. This can be accomplished by *escaping* the | character so that its special meaning within the context of a group expression is ignored.

To escape a group expression syntax character, it should be preceded with a backslash ("\"). So the above command should be changed to:

```
% display
  IN - NDF to be displayed > ‘find . -newer a.fit \| grep good ‘
```

Any other special character can be escaped in the same way. For instance, you can escape commas within text strings using this method.

4.13.5 Groups of Data Files

If a group expression is used to specify a list of input data files (NDFs or positions lists), then file names may be specified that contain wild-card characters ("*" and "?"—character classes can also be matched using strings such as "[0-9]", "[abcd]"). These will be expanded into a list of explicit file names before returning the group to the application.

If a group of output data files are specified by modification of a previously supplied group of input data files, the asterisk in the output group expression refers just to the file base-name (*i.e.* without directory path or file type). So, for instance, the group expression B_* would cause each output file name to be equal to the corresponding input file name, but with "B_" added to the start of the file base name. Thus an input file /home/dsb/data.fit would result in an output file /home/dsb/B_data.fit. If no directory is given in the output group expression, the directory associated with the input file is then added to the start of the file name. Likewise, any HDS path or file type is inherited from the input file if none are given in the output group expression.

If the final character in a group expression is a colon (:), then a list of the data files represented by the group expression (minus the colon) is displayed, but no data files are actually added to the group of files to be processed. The user is then re-prompted for another group expression, using the same parameter.

If an HDS container file[2] is supplied that contains two or more NDF structures, then each NDF within the container file is processed as a separate image. NDFs that are contained within an extension of another NDF are not included.

If a group of native output NDFs are created by modification of a group of native input NDFs (*i.e.* if the supplied string includes an asterisk), then the structure of each output container file will be copied from the corresponding input container file. For instance, if the container file `o66_int.sdf` contains 16 NDFs in components `I1` to `I16`, then specifying `"o66_int"` when asked for a group of input images will result in all 16 NDFs being used. If the corresponding output images are specified using the string `"*_A"` then a single output file named `o66_int_A.sdf` will be created. The structure of this file will be copied from the input file, and will therefore contain the 16 output NDFs in components `I1` to `I16`.

4.13.6 Examples

- If an application asks for a group of input data files, the following are all possible responses:

  ```
  b1,b2,b3,b4
  ```

  This means "Use the NDFs stored in files `b1.sdf`, `b2.sdf`, `b3.sdf` and `b4.sdf`". If 'on-the-fly format conversion' (see Section 18.1 and SUN/55) is being used, then this example would pick up data files with the highest priority data format (*i.e.* the format nearest to the start of the list of formats supplied in environment variable `NDF_FORMATS_IN`). So for instance, if the current directory contained both `b1.sdf` and `b1.fit`, then only one file would be used, depending on the relative positions of the `.sdf` and `.fit` formats within `NDF_FORMATS_IN`. If you want to restrict things explicitly to a particular data format, then you should include the corresponding file type in the group expression. An example such as:

  ```
  b1.fit,b2.fit,b3.fit,b4.fit
  ```

  would read just the specified FITS files.

  ```
  cena_b1-
  ```

  This means "Use `cena_b1.sdf` and then (because of the hyphen at the end) ask the user for more data files".

  ```
  *
  ```

  This means "Use all accessible data files in the current directory".

  ```
  hh1_b1s*_ds
  ```

  This means "Use `hh1_b1s1_ds.sdf`, `hh1_b1s2_ds.sdf`, *etc.*".

[2]HDS container files can usually be identified by the fact they have a file type of `.sdf`. They can be used to store one or more standard Starlink NDF structures.

```
hh1_b[12]s*_ds
```

This means "Use `hh1_b1s1_ds.sdf`, `hh1_b1s2_ds.sdf`, *etc.*, and also `hh1_b2s1_ds.sdf`, `hh1_b2s2_ds.sdf`, *etc.*". The string `"[12]"` matches either a single character `"1"` or a single character `"2"`. The string `"[0-9]"` would match any single digit character. The string `"[a-z]"` would match any single lowercase alphabetical character.

```
data.fit[12]
```

This means "Use files `data.fit1` and `data.fit2` if they exist. If neither of these files exists, use the twelth image extension in the multi-extension FITS file `data.fit`".

```
^files.lis
```

This means "Read the names of data files from the text file `files.lis`."

```
../data/*
```

This means "Use all accessible data files contained in the directory `../data`".

```
data_{new,old,back}
```

This means "Use files `data_new.sdf`, `data_old.sdf` and `data_back.sdf`".

```
{^files}_A
```

This means "Read names of data files from text file `files` and append `_A` to the end of each one".

```
‘grep -l "OBJECT  = ’m57’" *.fit‘
```

The string is enclosed in back quotes (‘) which causes the string to be executed as a shell command, and the resulting output to be used as the group expression. Thus this example means "Use all FITS files that contain an OBJECT keyword equal to `’m57’`".

- If an application asks for a group of output data files, the following are possible responses:

```
file1,file2,file3
```

This means "Create `file1.sdf`, `file2.sdf` and `file3.sdf`".

```
^out.dat
```

This means "Read the names of the output data files from text file `out.dat`".

```
*_ds
```

This means "Append the string `"_ds"` to the end of all the input data file names."

```
../bk/*|_ds|_bk|
```

This means "Substitute the string `"_bk"` for all occurrences of the string `"_ds"` in the input data file names, and put the files in directory `../bk`".

4.14 Output Parameters

Not only can programmes have parameters to which you supply values, but they can also write out the results of their calculations to *output* or *results parameters*. This makes the results accessible to subsequent tasks and to shell and ICL variables. Example results are statistics like the standard deviation or the FWHM of the point-spread function, the co-ordinates of points selected by a cursor, or the attributes of an NDF. They are not data files created by the application. In Appendix C they are listed separately from other parameters.

From the shell you can access these output parameters using the KAPPA tool PARGET. Suppose that you want to subtract the mean of an NDF called myndf to make an a new NDF called outndf.

```
% stats myndf > /dev/null
% set mean = ‘parget mean stats‘
% csub myndf $mean outndf
```

STATS calculates the statistics of myndf, the displayed output being discarded. PARGET reports the mean value which is assigned to shell variable `mean`. Thereafter the mean value is accessible as `$mean` in that process. Thus the final command subtracts the mean from myndf.

You can obtain vector values too.

```
% ndftrace myndf > /dev/null
% set axlab = ‘parget alabel ndftrace‘
% display otherndf style="’label(1)=$axlab[1],label(2)=$axlab[2]"’ axes
```

This displays the image otherndf surrounded by axes, but the plot's axis labels originate from another dataset called myndf [3]. There are more examples in the *C-Shell Cookbook*.

At the time of writing, ICL only permits scalar variables. To do the first example above from ICL, you would enter something like this.

```
ICL> stats myndf mean=(vmean)
ICL> csub myndf (vmean) outndf
```

`vmean` is an ICL variable. The parentheses have the same effect as the `$` in the C-shell example, meaning "the value of" the variable. Note that you can't redirect the output to `/dev/null`.

If you use the PROMPT keyword (see Section 4.11) for an application with output parameters, the programme will bizarrely prompt you for these. It is not asking for a value, but a *location* where to store the value. It is strongly recommended that you just accept the default (normally zero) so that the values are written to their parameter file, and hence permits PARGET to work.

[3] The style parameter specifies the appearance of the annotated axes, and is given as a comma-separated group of *attribute settings*, each of which is a `name=value` string specifying the attribute name and value. In this case, we assign values to the two attributes label(1) and label(2). The whole group must be enclosed in single quotes to prevent the parameter system splitting the string at the commas. The string must then also be enclosed in double quotes to prevent the UNIX shell from interpreting the parantheses and equals signs. A simpler way of specifying a plotting style is to put the attribute settings in a text file (see Section 7).

5 Verbosity of Messages

Informational messages (as opposed to error messages) are tagged with a reporting level that permits some control in the detail and quantity of messages you see. Three levels are supported in KAPPA.

NORMAL the normal reporting level, that aims to give a balanced, 'Goldilocks' level of reporting. However, for historical reasons most messages are set to this level, although that is gradually changing. It excludes messages tagged as verbose.

VERBOSE reports additional information, such as instructions or warnings for new users to a task, progress reports in a long-running command, and detailed analysis.

QUIET reports only the most important messages that you would not want to miss, but eliminates the messages at normal and verbose levels.

The chosen level is set through environment variable `MSG_FILTER`. Abbreviations may be used such as `"N"`, `"v"`, `"Qu"` for normal, verbose, or quit reporting respectively. The normal mode is the default if you do not define `MSG_FILTER`.

6 Graphics Devices and Files

6.1 Selecting a Graphics Device

You can find the list of available devices and their names with task GDNAMES. Names can be abbreviated provided they remain unambiguous. Two alternative naming schemes are supported, and the list produced by GDNAMES will include both.

PGPLOT A general graphics device specification is of the form `<file>/<type>`, where `<type>` indicates the type of the graphics device (*e.g.* PostScript printer, X-window, *etc.*) and `<file>` is an optional string which indicates either a file in which the graphical output should be stored or a specification for a particular device of the specified type. For instance, `m31.ps/VPS` produces a file called `m31.ps` containing output suitable for sending to a PostScript printer in portrait mode, and `xwindows2/GWM` sends graphical output to the GWM X-window with name `xwindows2`. If the `<file>` string is omitted, a default device-dependent value is used (for instance, `pgplot.ps` for postscipt files and `xwindows` for X-windows).

GNS For compatibility with previous versions of KAPPA graphics devices may also be specified using a scheme that approximates closely to that of the Starlink Graphics workstation Name Service (GNS) library (see SUN/57). In this scheme a complete workstation specification is of the form `<type>;<file>`. This is very similar to the PGPLOT scheme described above, but the device type and file are swapped round, and the separator is a semicolon instead of a solidus. As for PGPLOT, the device file can be omitted, in which case a default is used, but note that *in this case the semicolon should also be omitted*. The only supported devices are those for which PGPLOT drivers are available. If a device was also available in the GKS-based graphics systems previously used by KAPPA, then you can refer to it either using its PGPLOT device type or its GNS device type.

In either scheme, either the device type or the file name or the entire device specification may be replaced by a logical name, in which case the value of the logical name will be used instead.

6.1.1 Global Parameters

There is a global parameter for the graphics device. The purpose of this global parameter is ostensibly to prevent unnecessary prompting. However, there is an ulterior motive as well. The selection of devices outside of the graphics applications enables us to perform other necessary actions just once.

There is a command for selecting the current graphics device: GDSET. For example,

```
ICL> gdset xwindows
```

A selection remains in force until you change it using GDSET again, use NOGLOBALS, or delete the globals file. The current choice can be inspected via the GLOBALS command. If the global parameter is undefined you will be prompted for the device if an application requires it.

You can override the global parameter for the duration of a single application by specifying it by keyword (normally `DEVICE=`), or in some applications, by position. Here is an example.

```
ICL> contour device=ps_p
```

6.1.2 X-windows

The most commonly used devices are X-windows. These can require a little preparation before you select a device. Starlink graphics use GWM to manage windows. It enables a window to persist between separate applications; or to be shared by different applications, potentially even running on different machines. See SUN/130 for details of GWM and how to change your X-defaults file (`$HOME/.Xdefaults`), but the salient points are given below.

If the window appears on a terminal or workstation other than the one running the KAPPA executables you will need to redirect output to your screen, if you have not already done so for some other software. You either use the `xdisplay` command

```
% xdisplay myterm.mysite.mydomain.mycountry
```

or set the `DISPLAY` environment variable to point to the address of your screen.

```
% setenv DISPLAY myterm.mysite.mydomain.mycountry:0
```

You substitute your machine's address or IP number. (Ask your computer manager.)

If you do not create the window before running KAPPA, the first graphics application to open an X-windows device will create the window, using certain defaults. The defaults control amongst others the foreground and background colours, the number of colours allocated, the size and location of the window. These defaults may be altered with an X-defaults file, or a window created with the GWM **xmake** command.

```
% xmake xwindows -geom 600x450 -fg yellow -bg black
```

This example makes a window of dimension 600-by-450 pixels, the background colour is black and colour for the line graphics is yellow.

The following set up to place in your X-defaults file is a reasonable compromise, as it maximises the number of colour indices for the graphics window (xwindows), and has a second graphics window (x2windows). In the defaults file there are the following lines

```
gwm*xwindows*colours:          80
gwm*xwindows2*colours:         20
```

and you can also set the sizes of the windows too. Notice that the second device name is x2windows, but the window name is xwindows2. Don't ask why. This confusing name rule applies also to all but the first window of the maximum of four windows allowed.

The device names can be abbreviated, to give unambiguous names. Thus you can enter `xw` for the xwindows device; `x2w` for the x2windows device; and so on. This is the reason for having device names as they are.

The following tells KAPPA that this is the current device. This remains as a global parameter, so you probably will not need to issue this command that often.

```
% gdset x2windows
```

6.2 Composite Hardcopy Plots

KAPPA applications are modular so that you can build up more-complicated plots, and possibly add annotations with other packages especially PONGO. This is fine if the device is some sort of screen. However, care needs to be taken when using other types of device. For instance, some PGPLOT PostScript devices put the output from each command into a separate disk file. So how do you get the composite plot out on paper? There are two solutions:

(1) the easy way—use an "accumulating" PostScript device instead. If you look at the list of device names produced by GDNAMES you will see that some of the encapsulated PostScript (EPS) devices are described as "accumulating" such as the following.

```
% gdnames
...
aps_p     (/AVPS)          Accumulating EPS, monochrome, portrait
aps_l     (/APS)           Accumulating EPS, monochrome, landscape
apscol_p  (/AVCPS)         Accumulating EPS, color, portrait
apscol_l  (/ACPS)          Accumulating EPS, color, landscape
...
```

If you use one of these devices, each subsequent graphics application will merge its PostScript output automatically into any existing `pgplot.ps` file. If you display `pgplot.ps` using a modern PDF/PostScript viewer such OKULAR or EVINCE, the display will update automatically as each subsequent graphics application modifies the file. Once the final graphics application has been run, you can rename the `pgplot.ps` file to something more covenenient[4].

To give an example, suppose we wanted to overlay a contour plot on an image.

```
% gdset apscol_l
% display $KAPPA_DIR/comwest lut=$KAPPA_DIR/spectrum_lut mode=ra axes
% contour noclear mode=au noaxes \\
% mv pgplot.ps myplot.ps
```

(2) The hard way—use PSMERGE to merge separate PostScript files. If you use one of the non-accumulating PostScript devices, the output from each graphics application goes into a separate file which must then be merged. Here is what you must do.

- Select one of the non-acculumlating encapsulated PostScript device, be it colour or monochrome.
- Produce your graphics, being careful to rename the output file (usually `pgplot.ps`) created by each command to avoid it being over-written by the output from the next command. Alternatively, you can specified the device to use separately when running each command, including an explicit unique file name in each device specification (see above).
- Use PSMERGE (the Starlink version, usually in `/star/bin`) to combine the plots.

[4]You could alternatively include the required final file name in the device name before running the graphics applications.

To use the same example as above:

```
% gdset epsfcol_l
% display $KAPPA_DIR/comwest lut=$KAPPA_DIR/spectrum_lut mode=ra axes
% mv pgplot.ps display.ps
% contour noclear mode=au noaxes \\
% mv pgplot.ps contour.ps
% psmerge display.ps contour.ps  > myplot.ps
% rm display.ps contour.ps
```

or alternatively:

```
% display $KAPPA_DIR/comwest device="epsfcol_l;display.ps" \
          lut=$KAPPA_DIR/spectrum_lut mode=ra axes
% contour noclear device="epsfcol_l;contour.ps" mode=au noaxes \\
% psmerge display.ps contour.ps  > myplot.ps
% rm display.ps contour.ps
```

to form the merged graphic. You then print `myplot.ps` to the colour PostScript printer. PSMERGE also has options for scaling and rotating plots.

7 Plotting Styles and Attributes

7.1 Plotting Styles and Attributes

Many different aspects of the appearance of line graphics produced by KAPPA applications can be controlled by specifying a *plotting style* when running the application. This includes things like the colour, line width, line style, character size, and fount, each of which can be specified individually for different components of a plot (*e.g.* the tick marks, numerical labels, border, textual labels, *etc.*).

Each aspect of a plot is controlled by a *plotting attribute*. A plotting attribute has a *name* and a *value*.

Strings that assign new values to particular plotting attributes are called attribute setting strings. For instance:

```
Border=1
Title=This is my plot title
```

are two attribute setting strings. The first assigns the value `1` to the attribute called Border, and the second assigns the string `"This is my plot title"` to the attribute with name Title (attribute names are case insensitive but cannot be abbreviated).

Here is a list of the available attributes, with a brief description of each. Full descriptions are included in Appendix E.

Border	Draw a border around valid regions of a plot?
Colour(element)	Colour for a plot element
DrawAxes	Draw axes?
DrawDSB	Annotate both sidebands in a dual sideband spectrum?
DrawTitle	Draw a title?
Edge(axis)	Which edges to label
FileInTitle	Include the NDF name as a second line in the title?
Font(element)	Character fount for a plot element
Gap(axis)	Interval between major axis values
Grid	Draw grid lines?
LabelAt(axis)	Where to place numerical labels
LabelUnits(axis)	Include unit descriptions in axis labels?
LabelUp(axis)	Draw numerical axis labels upright?

Labelling	Label and tick placement option
MajTickLen	Length of major tick marks
MinTickLen	Length of minor tick marks
MinTick(axis)	Density of minor tick marks
NumLab(axis)	Draw numerical axis labels?
NumLabGap(axis)	Spacing of numerical axis labels
Size(element)	Character size for a plot element
Style(element)	Line style for a plot element
TextBackColour	Background colour to use when drawing text
TextLab(axis)	Draw descriptive axis labels?
TextLabGap(axis)	Spacing of descriptive axis labels
TextMargin	Width of margin to clear when drawing text
TickAll	Draw tick marks on all edges?
TitleGap	Vertical spacing for the title
Tol	Plotting tolerance
Width(element)	Line width for a plot element

Some attribute names can be qualified so that they refer to a particular component of the plot. This is done by putting the qualifier in parentheses after the attribute name. For instance:

```
Colour(border)=2
Edge(2)=left
```

assigns the value 2 to the Colour attribute for the plot border, and assigns the value `left` to the Edge attribute for the second axis. The full description of each attribute describes what happens if you omit the qualifier. Attribute names that end in `"(axis)"` in the above list can be qualified by an integer axis index to refer to a particular plot axis. Attribute names that end in `"(element)"` in the above list can be qualified by any of the following strings to refer to a particular component of the plot (the qualifiers are case-insensitive and unambiguous abbreviations may be used):

Axes	Axis lines drawn through tick marks within the plotting area, drawn if attribute DrawAxes is given a non-zero value. Values set using Axes are only only used if axis-specific values have not been set using Axis1 or Axis2. Thus, Axes provides default values for Axis1 and Axis2.
Axis1	An axis line drawn through tick marks on Axis 1 within the plotting area, drawn if attribute DrawAxes is given a non-zero value. Values specified using Axis1 override any supplied using Axes.

Axis2 An axis line drawn through tick marks on Axis 2 within the plotting area, drawn if attribute DrawAxes is given a non-zero value. Values specified using Axis2 override any supplied using Axes.

Border The plot border, drawn if attribute Border is given a non-zero value.

Curves Curves drawn over the top of a plot (*e.g.* contours, data curves).

Grid The grid lines, drawn if attribute Grid is given a non-zero value. Values set using Grid are only only used if axis-specific values have not been set using Grid1 or Grid2. Thus, Grid provides default values for Grid1 and Grid2.

Grid1 Grid lines that cross Axis 1, drawn if attribute Grid is given a non-zero value. Values specified using Grid1 override any supplied using Grid.

Grid2 Grid lines that cross Axis 2, drawn if attribute Grid is given a non-zero value. Values specified using Grid2 override any supplied using Grid.

Markers Graphical markers (symbols) drawn over a plot.

NumLab Numerical axis labels drawn around annotated axes. Values set using NumLab are only only used if axis-specific values have not been set using NumLab1 or NumLab2. Thus, NumLab provides default values for NumLab1 and NumLab2.

NumLab1 Numerical labels drawn along Axis 1. Values specified using NumLab1 override any supplied using NumLab.

NumLab2 Numerical labels drawn along Axis 2. Values specified usingNumLab2 override any supplied using NumLab.

Strings Text strings drawn over a plot (except for axis labels and plot title).

TextLab Descriptive axis labels drawn around annotated axes. Values set using TextLab are only only used if axis-specific values have not been set using TextLab1 or TextLab2. Thus, TextLab provides default values for TextLab1 and TextLab2.

TextLab1 Descriptive label for Axis 1. Values specified using TextLab1 override any supplied using TextLab.

TextLab2 Descriptive label for Axis 2. Values specified using TextLab2 override any supplied using TextLab.

Ticks Tick marks (both major and minor) drawn along annotated axes. Values set using Ticks are only only used if axis-specific values have not been set using Ticks1 or Ticks2. Thus, Ticks provides default values for Ticks1 and Ticks2.

Ticks1 Tick marks (both major and minor) drawn along Axis 1. Values specified using Ticks1 override any supplied using Ticks.

Ticks2 Tick marks (both major and minor) drawn along Axis 2. Values specified using Ticks2 override any supplied using Ticks.

Title The title drawn at the top of a plot.

A collection of attribute settings is called a *plotting style*. Attributes that are not specified in a plotting style take on default values as described later. An example of a plotting style is contained in the file `$KAPPA_DIR/kappa_style.def`. This file defines the default style used by KAPPA.

In general each graphical application will determine the plotting style to be used when drawing line graphics by accessing a parameter called STYLE. Some applications have more than one style parameter. For instance, applications such as CONTOUR and LINPLOT that produce keys to the content of the plot have a parameter called KEYSTYLE, in addition to the normal STYLE parameter. KEYSTYLE is used to specify the plotting style for the key, whereas STYLE is used to specify the plotting style for the main plot.

7.2 Specifying a Plotting Style

7.2.1 Group Expressions

The application parameters that are used to access plotting styles (STYLE, KEYSTYLE, *etc.*) expect a group of attribute setting strings to be supplied for the parameter. These should be supplied in the form of a *group expression*.[5]

A group expression is a character string containing a list of one or more sub-strings separated by some specified delimiter. The usual delimiter used when accessing plotting style parameters is a comma (although other parameters that require group expressions for other purposes may use a different delimiter). Each sub-string is known as a group *element*. Each group element must be either:

- an attribute setting string (*e.g.* `"Border=1"`), or
- the name of a text file, preceded by an up-arrow character (`"^"`).

If a group element starts with an up-arrow character, then the rest of the element is interpreted as a file name, and an attempt is made to read further group elements from the specified file. Each line in the file is interpreted as a group expression in its own right, using exactly the same rules as described above. In particular, references to text files can be nested (*i.e.* a text file can include a group element that refers to another text file). Blank lines and lines in which the first non-blank character is a hash (`"#"`) are ignored and can be used to add textual comments to a text file.

Attribute settings are used in the order in which they occur in the group expression. If a group expression includes more than one setting string for a given attribute, then the value that occurs nearest to the end of the group expression will overwrite any earlier values.

All this means that there are several ways in which plotting styles can be supplied. The simplest method is probably to store all your attribute setting strings in a text file, one per line, and then just give the name of this text file (preceded by an up-arrow) as the value for the STYLE parameter. For instance, the file `style.dat` may contain:

[5]The complete group expression syntax is described in SUN/150. This is the documentation for the GRP subroutine library, which provides a programming interface for obtaining groups of strings.

```
# A test plotting style

width=2
edge(2)=right
title=This is my title
```

The first line and the following blank line are ignored. The remaining three lines specify three attribute values to use. When running an application such as DISPLAY, this style file would be specified on the command line as follows:

```
% display style=^style.dat
```

Alternatively, you can specify the setting strings explicitly on the command line. This can get a bit messy because you need to protect special characters (commas, parentheses, spaces, equals signs, and so on) both from the UNIX shell and from the parameter system. One safe way to do this is to enclose the whole group expression in single quotes, and then enclose the whole thing again in double quotes.[6] So the above style could be given on the command line as follows:

```
% display style="'width=2,edge(2)=right,title=This is my title'"
```

A bit messy as I said! However, it can be useful to combine this method with the previous method. If you have a long, complicated style file, and you want to change one or two attribute settings, one method would be to take a copy of the style file and edit it. This would probably be the best thing to do if you intend to re-use the edited style file several times. But if you just want to try a quick experiment, just to see what the results look like, you can avoid the trouble of editing the style file by giving both the original style file *and* the new attribute settings on the command line. For instance:

```
% display style="'^style.dat,width=3'"
```

This reads the contents of our test style file `style.dat`, which includes the attribute setting `width=2`. It then also applies the attribute setting `width=3`, over-writing the effect of the `Width` value included in the style file. If you wanted to try temporarily changing the value of several attributes at once, you could put the new attribute settings into a second file, say `style2.dat`, and then run the application as follows:

```
% display style="'^style.dat,^style2.dat'"
```

Again, the contents of `style.dat` would be read, followed by the contents of `style2.dat`, over-writing any settings for the same attributes included in `style.dat`.

[6]The up-arrow character is not one of these special characters, and so a simple reference to a single text file does not need to be enclosed in quotes.

7.2.2 Temporary Attributes

Sometimes you need to effect a change of style that only lasts for a single invocation of a task. For example, plotting data from different NDFs or NDF sections in different colours. This is available at the cost of a little further syntax to learn. The temporary attributes should be preceded by a plus sign.

In the following example, three histograms are plotted on a single graphic.

```
% histogram ndf1 \\
% histogram ndf2 style="+colour(curve)=yellow" noclear noaxes \\
% histogram ndf3 style="+width(curve)=4" noclear noaxes \\
```

The first uses the current attributes including line colour to plot the histogram for NDF ndf1. (Recall \\ is a synonym for the accept keyword, so other defaults are used for the likes of the data range and number of bins omitted for clarity.) The second NDF's histogram is plotted in yellow. For the third NDF the locus is again in the colour of the first plot (since the yellow was only temporary) but has thickness four times normal. If you ran HISTOGRAM again, the line thickness would return to its original value used in the first graph.

You are not limited to just one temporary attribute. You can supply a list that can include indirection to text files.

```
% linplot spectrum style="'+style(curve)=2,grid,^temp.sty,colour(textlab)=green'"
```

Here LINPLOT uses three temporary attributes plus whatever is defined in the text file file `temp.sty`. Note for a list the string requires an extra set of enclosing quotes to protect these from being misinterpreted by the UNIX shell. The experimental method that used a second parameter called TEMPSTYLE will be withdrawn.

It is also possible to combine persistent and temporary attributes. Persistent style attributes must be supplied first, then after a plus sign comes the list of temporary attributes.

```
% linplot spectrum style="'colour(line)=red,width=3+style(curve)=2'"
```

Literal plus signs should be avoided if using both temporary and persistent attributes in a group expression.

7.2.3 Synonyms for Attribute Names

The available plotting attributes and their names are defined by the AST subroutine library (see SUN/210) upon which KAPPA graphics are based (together with PGPLOT). However, KAPPA provides synonyms for certain plotting attributes where this would provide a clearer indication of the purpose of the attribute within the context of a particular application. These synonyms are listed in the description of the STYLE parameter for the particular applications concerned. For instance, the CONTOUR applications draws contours as 'curves', that is, it uses the attributes Colour(Curves), Width(Curves) and Style(Curves) to determine the appearance of the contours. However, the synonym Contours (minimum abbreviation Cont) may be used in place of Curves, so the appearance of the contours can also be specified using the 'pseudo-' attributes Colour(Cont), Width(Cont) and Style(Cont).

You should remember that a synonym is simply an alternative way of specifying a particular attribute. So if you are running CONTOUR and you give the two setting strings:

```
Colour(cont)=red
Colour(curve)=blue
```

the contours will appear blue, not red, because the second attribute setting overrides the first one.

Any particular synonym will only be recognised by certain applications. Thus, Contours is only recognised as a synonym for Curves when running CONTOUR. Applications ignore attributes (or synonyms) that they do not recognise without reporting an error. Thus if the file containing your default plotting style (used by all applications—see later) contains the two lines:

```
Colour(curve)=blue
Colour(cont)=red
```

then CONTOUR will draw contours in red, but other applications will draw their curves in blue since they ignore the `Colour(cont)=red` line. Note, if these two lines were the other way round:

```
Colour(cont)=red
Colour(curve)=blue
```

then all curves, *including* contours drawn by CONTOUR, would be blue. This is because CONTOUR will process both lines in the order supplied, ending up with blue contours.

7.2.4 Colour Attributes

The Colour attribute can be used to specify the colours of various components of a plot. The value assigned to the attribute can be one of the following options.

- An integer colour index. Colour indices can be thought of as 'pen numbers'. For instance, the string `"Colour(border)=3"` says "Draw the border using pen number 3". The resulting colour depends on the colour of Pen 3, which can be set using PALENTRY.
- A standard X colour name, *e.g.* `"Colour(border)=red"`.
- A triple of floating-point red, green and blue intensity values in the range zero to one, separated by spaces, *e.g.* `Colour(border)=1.0 0.0 0.0`.
- An HTML colour code. This is a hash followed by three pairs of hexadecimal digits giving red, green, and blue intensity in the range 0 to 255, *e.g.* `"Colour(border)=#ffa700"`. Within style files, the `"#"` character is used to introduce a comment, and so the colour code would be ignored. To avoid this, the `"@"` character can be used in place of `"#"`.

If no pen is currently available in the palette with the requested colour, then the 'nearest' colour will be used instead—sometimes this may not be very near at all! If you specifically want the requested colour, then you should use PALENTRY first to set one of the available pens to the required colour.

Note, if you produce a plot on an X-window and then change the representation of pens using PALENTRY, then any components of the existing plot that were drawn with the modified pens will change colour immediately *if and only if* your X window is set to 256 colours (*i.e.* if you have an 8 bit pseudo-colour visual). If you are in the more usual situation of having a 16- or 24-bit display, then changes to pen colours will only affect subsequently drawn graphics.

7.3 Establishing Defaults for Plotting Attributes

When an application needs a plotting style, it uses a style parameter to get a group of attribute settings. But what values are used for attributes that are not included in this group?

Obviously, if an attribute has been assigned an explicit value using the parameter then that value is used, but you should note that most styles are 'sticky'.[7] That is, once a group of attribute settings has been specified for a style parameter, that group continues to be used by subsequent invocations of the application until a new group is supplied for the parameter. If the group is supplied within a text file, then the 'current value' stored for the parameter is the list of attribute settings read from the file, *not* the name of the file. Thus, changing the contents of the file at a later time will have no effect on the value used for the parameter unless you re-specify the parameter on the command line.

If an attribute is not specified in the supplied group, then a default value is used for the attribute, determined as follows:

(1) If the plot is being overlayed on another existing plot, then the value that was used for the attribute when the existing plot was created is used (but only if it was set to an explicit value).

(2) Otherwise, the attribute value specified in a *defaults file* is used. The defaults file is found as follows:

- If the environment variable `KAPPA_<APP>_<PARAM>` is defined (where `<APP>` is the name of the application *e.g.* DISPLAY, and `<PARAM>` is the name of the parameter, *e.g.* STYLE, both in upper-case), its value is taken to be the full path to the defaults file.
- If `KAPPA_<APP>_<PARAM>` is not defined, the file `$HOME/kappa_<app>_<param>.def` is used (where `<app>` and `<param>` are in lower case this time).
- If the file `$HOME/kappa_<app>_<param>.def` cannot be accessed, the file `$KAPPA_DIR/kappa_<app>_<param>.def` is used.
- If the file `$KAPPA_DIR/kappa_<app>_<param>.def` cannot be accessed, then the value of the environment variable `KAPPA_<PARAM>` is used as the full path to the defaults file.
- If `KAPPA_<PARAM>` is not defined, the file `$HOME/kappa_<param>.def` is used.
- If the file `$HOME/kappa_<param>.def` cannot be accessed, the file `$KAPPA_DIR/kappa_<param>.def` is used.

(3) If the above process failed to produce a value for the attribute (either because no file was found, or the file did not contain a setting for the attribute), then the default value supplied by the AST library is used. These defaults are included in the full description of the relevant attributes in Appendix E.

From the above, you can see that defaults can be specified either for individual applications, or for all applications (any application-specific defaults file will be used in preference to the general defaults file).

[7] The exceptions are the TEMPSTYLE parameters. These parameters are used to make temporary style changes and always forget any previous values assigned to them.

It should be remembered that settings for unknown attributes are ignored, and do not cause the application to abort.[8] So if you set a value for an attribute and it seems to have no effect, it may be worth checking that you have used the correct spelling for the attribute name.

7.4 Graphical Escape Sequences

Strings used for axis labels, plot titles, *etc.*, can include special *escape sequences* which control the appearance of subsequent text when the string is drawn as part of a plot.[9] Escape sequences are introduced using a percent character. For instance, if the string `"10%^50+%^s50+Z"` was used as an axis label in a plot, it would produce a string similar to "10^{Z}"—that is, the character `"Z"` would be displayed as a small superscript character. Any unrecognised, illegal or incomplete escape sequences are printed literally. The following escape sequences are currently recognised (`"..."` represents a string of one or more decimal digits):

- `%%` : Print a literal `"%"` character.
- `%^...+` : Draw subsequent characters as superscripts. The digits `"..."` give the distance from the base-line of 'normal' text to the base-line of the superscript text, scaled so that a value of `"100"` corresponds to the height of 'normal' text.
- `%^+` : Draw subsequent characters with the normal base-line.
- `%v...+` : Draw subsequent characters as subscripts. The digits `"..."` give the distance from the base-line of 'normal' text to the base-line of the subscript text, scaled so that a value of `"100"` corresponds to the height of 'normal' text.
- `%v+` : Draw subsequent characters with the normal base-line (equivalent to `%^+`).
- `%>...+` : Leave a gap before drawing subsequent characters. The digits `"..."` give the size of the gap, scaled so that a value of `"100"` corresponds to the height of 'normal' text.
- `%<...+` : Move backwards before drawing subsequent characters. The digits `"..."` give the size of the movement, scaled so that a value of `"100"` corresponds to the height of 'normal' text.
- `%s...+` : Change the Size attribute for subsequent characters. The digits `"..."` give the new Size as a fraction of the 'normal' Size, scaled so that a value of `"100"` corresponds to 1.0;
- `%s+` : Reset the Size attribute to its 'normal' value.
- `%w...+` : Change the Width attribute for subsequent characters. The digits `"..."` give the new width as a fraction of the 'normal' Width, scaled so that a value of `"100"` corresponds to 1.0;
- `%w+` : Reset the Width attribute to its 'normal' value.
- `%f...+` : Change the Font attribute for subsequent characters. The digits `"..."` give the new Font value.

[8] This is because they may be synonyms recognised by other other applications.

[9] When displayed in a non-graphical environment (for instance, on an alpha-numeric terminal) the characters forming an escape sequence are stripped out of the string before the string is displayed.

- %f+ : Reset the Font attribute to its 'normal' value.
- %c...+ : Change the Colour attribute for subsequent characters. The digits "..." give the new Colour value.
- %c+ : Reset the Colour attribute to its 'normal' value.
- %t...+ : Change the Style attribute for subsequent characters. The digits "..." give the new Style value.
- %t+ : Reset the Style attribute to its 'normal' value.
- %- : Push the current graphics attribute values on to the top of the stack (see %+).
- %+ : Pop attributes values of the top the stack (see %-). If the stack is empty, 'normal' attribute values are restored.

PGPLOT escape sequences may also be included in strings that are to be drawn.

8 Data structures

In an ideal world you would not need to know how your data are stored. It would be transparent. Starlink applications attempt to achieve this through standard, but extensible, data structures, and the ability to apparently operate on other formats through the so-called 'on-the-fly conversion' (see Section 18.1 and SUN/55).

The official standard data format used by Starlink applications is the NDF (Extensible *n*-dimensional Data Format, SUN/33). The data in an NDF is stored using HDS which has numerous advantages, not least that *HDS files are portable between operating systems*; both have file extension `.sdf`.

The NDF has been carefully designed to facilitate processing by both general applications like KAPPA and specialist packages. It contains an *n*-dimensional data array that can store most astronomical data such as spectra, images and spectral-line data cubes. The NDF may also contain other items such as a title, axis labels and units, error and quality arrays, and World Co-ordinate System (WCS) information. There are also places in the NDF, called *extensions*, to store any ancillary data associated with the data array, even other NDFs.

The NDF format and its components are described more fully in Appendices H, which includes commands for manipulating the components.

The NDF format permits arrays to have seven dimensions, but some applications only handle one-dimensional and/or two-dimensional data arrays. The data and variance arrays are not constrained to a single data type. Valid types are the HDS numeric primitive types, see Appendix J.

Many applications are generic, that is they can work on all or some of these data types directly. This makes these applications faster, since there is no need to make a copy of the data converted to the type supported by the application. If an application is not generic it only processes _REAL data. Look in the `Implementation Status` in the help or the reference manual. If none is given you can assume that processing will occur in _REAL.

In KAPPA the elements of the data array are often called *pixels*, even if the NDF is not two-dimensional.

8.1 Restrictions on the Usage of Data Structures

By default, KAPPA plays safe and will not allow you to use the same data structure as both input and output for a command. This is to minimise the risks of accidentally over-writing valuable data. So, for instance, if you try the following command:

```
% cadd in=m31 scalar=10 out=m31
```

you will find that the value of `m31` for Parameter OUT is rejected with a message indicating that the data structure is already in use, and you will be prompted for an alternative value.

However, KAPPA does allow you to 'live on the edge' if you prefer—if you define the environment variable `KAPPA_REPLACE` before running a command, then KAPPA will happily overwrite

the input data structure if requested to do so. You can assign any value you like to this environment variable, since its mere existence is the trigger for this optional behaviour. Note, this facility is only available in those commands that access the input data structures *before* the output data structures (the vast majority).

8.2 Looking at the Data Structures

You can look at a summary of an NDF structure using the task NDFTRACE, and obtain the values of NDF extension components with the `setext option=get` command. HDSTRACE (SUN/102) can be used to look at array values and extensions.

8.3 Editing the Data Structures

There are facilities for editing HDS components, though these should be used with care, lest you invalidate the file. For instance, if you were to erase the DATA_ARRAY component of an NDF , the file would no longer be regarded as an NDF by applications software.

In KAPPA, ERASE will let you remove any component from within an HDS container file, but you have to know the full path to the component. SETEXT has options to erase extensions and their contents, without needing to know how these are stored within the HDS file. It also permits you to create and rename extension components, and assign new values to existing components. There are a number of commands for manipulating FITS-header information stored in the NDF's FITS extension. These are described in Section 18.4.

FIGARO offers some additional tasks (CREOBJ, DELOBJ, and RENOBJ) for editing HDS components.

8.4 Native Format

Although HDS files are portable you are recommended to copy them to the host machine, and run application NATIVE on them for efficiency gains. NATIVE converts the data to the native format of the machine on which you issue the command. If you don't do this, every time you access the data in your NDF , this conversion process occurs. NATIVE also replaces any IEEE floating-point NaN or Inf values with the appropriate Starlink bad value. The following converts all the HDS files in the current directory.

```
% native "*"
```

9 NDF Sections

You will frequently want to examine or process only a portion of your dataset, be it to focus on a given object in an image, or a single spectrum between nominated wavelengths, or a plane of a cube. You could use NDFCOPY or MANIC in some circumstances to make a new NDF containing the required data, but this would be inconvenient as you would need more disc space, and to invent and remember a new filename. You will be pleased to learn that there is a succinct and powerful alternative that obviates the need to create a new file—the NDF section. The application just processes a 'rectangular' subset, or section, of the NDF that you nominate. Certainly, it requires you to learn a little syntax, but after you use it a few times it will seem cheap at the price for the advantages it offers.

An NDF section is defined by specifying the bounds of the portion of the NDF to be processed immediately following the name of the NDF. You can do this in any place where an NDF name alone would suffice, for example, on the command line or in response to a prompt or as a default in an interface file. The syntax is a series of subscripts within parentheses and may be given in several ways. Here is a simple example.

```
ICL> stats cluster(101:200,51:150)
```

This would derive statistics of a 100×100-pixel region starting at pixel indices (101, 51) in the NDF called cluster. Alternatively, ranges of axis co-ordinates may be given instead of pixel indices. Besides giving lower and upper bounds as above, you may specify a centre and extent. Sections are not limited to subsets—supersets are allowed. See the paragraphs below for more details of these features.

If you *do* want to make a new NDF from a portion of an existing one, you should use the command NDFCOPY. An NDF's shape may be changed *in situ* by SETBOUND.

Note if you supply an NDF section on a C-shell command line, you must escape the parentheses. For example, the following are both equivalent to the earlier example.

```
% stats cluster"(101:200,51:150)"
% stats cluster\(101:200,51:150\)
```

9.1 Specifying Lower and Upper Bounds

The subscript expression appended to an NDF name to specify a section may be given in several ways. One possible method (corresponding with the example above) is to give the lower and upper bounds in each dimension, as follows

```
NAME( a:b, c:d, ... )
```

where '`a:b`', '`c:d`', (*etc.*) specify the lower and upper bounds. The bounds specified need not necessarily lie within the actual bounds of the NDF, because *bad* pixels (see Section 15) will be supplied in the usual way, if required, to pad out the NDF's array components whenever

they are accessed. However, none of the lower bounds should exceed the corresponding upper bound.

Omitting any of the bounds from the subscript expression will cause the appropriate (lower or upper) bound of the NDF to be used instead. If you also omit the separating ':', then the lower and upper bounds of the section will both be set to the same value, so that a single pixel will be selected for that dimension. Omitting the bounds entirely for a dimension (but still retaining the comma) will cause the entire extent of that dimension to be used. Thus,

```
image(,64)
```

could be used to specify row 64 of a two-dimensional image, while

```
cube( 1, 257:, 100 )
```

would specify column 1, pixels 257 onwards, selected from plane number 100 of a three-dimensional 'data cube', forming a one-dimensional section.

9.2 Specifying Centre and Extent

An alternative form for the subscript expression involves specifying the centre and extent of the region required along each dimension, as follows

```
name( p~q, r~s ... )
```

where '`p`∼`q`', '`r`∼`s`', (*etc.*) specify the centre and extent respectively. The extent must be positive. Thus,

```
name(100~11,200~5)
```

would refer to an 11×5-pixel region of an image centred on pixel (100, 200).

If the value before the delimiting '∼' is omitted, it will default to the index of the central pixel in that dimension (rounded downwards if there are an even number of pixels). If the value following the '∼' is omitted, it will default to the number of pixels in that dimension. Thus,

```
image( ~100, ~100)
```

could be used to refer to a 100×100-pixel region located centrally within an image, while

```
image( 10~, 20~ )
```

would specify a section that is the same size as the original image, but displaced so that it is centred on pixel (10, 20).

9.3 Using World or Axis Co-ordinates to Specify Sections

Not only can you specify sections in terms of pixel indices but also in terms of more-tangible co-ordinates such as right ascension and declination, wavelength, frequency, and time. Section locations in such world co-ordinate systems (WCS) (see Section 12) are indicated by non-integer values; whereas integer values in an NDF section are intepreted as pixel indices. Here an integer value is defined as one neither containing a decimal point nor an exponent. The WCS bounds of each section are converted to the nearest pixel indices in order to specify the included data elements.

Since there may be many co-ordinate systems that could represent your desired section, some rules are necessary to decide how to interpret the section limits. First, to retain backwards compatibility (with pre-V1.8 KAPPA), if your NDF has AXIS components then non-integer values will refer to *axis* co-ordinates (for a description of *AXIS* co-ordinates, see Section 12.2). Otherwise the values are specified within the co-ordinate system represented by the current Frame in the NDF's FrameSet (see Section 12.3). Command NDFTRACE will show whether or not an NDF has AXIS components, and if so, it reports their extents, labels, and units. NDFTRACE also summarises the properties of the current WCS Frame (or optionally all the WCS Frames present), if the NDF has a WCS component. You can change the current Frame with WCSFRAME.

9.3.1 World co-ordinates:

The standard formats used to specify WCS co-ordinates apply.See SUN/210 AST_UNFORMAT sections "Frame Input Format" and"SkyFrame Input Format" for details and examples.

Here are some examples of WCS co-ordinates defining NDF sections.

```
spectrum(9000:2E4)
```

This could specify a region of a spectrum between 9000 and 20000 Ångstrom.

```
image(12h59m49s~4.5m,27.5d:28d29.8m)
```

This could refer to a region approximately one degree on each side centred the Coma Cluster. The section is 4.5 minutes of time along the right-ascension axis centred at $12^{\rm h}59^{\rm m}49^{\rm s}$, and extends from $+27^\circ30'$ to $+28^\circ29'48''$ in declination.

```
ifu(1h34m10.1s:1h34m12.4s,-2d35m:-2d35.5m,6563)
```

This might specify a region of sky 2.3 seconds of time by 0.5 arcminutes in H_α light from an integral-field unit.

It is possible to use a colon to separate the fields in sexagesimal celestial and time co-ordinates, as in the following example that defines the equivalent section as the preceding Coma Cluster example.

```
image(12:57:34;13:02:04,27:30:00;28:29:48)
```

There is a cost; you must use a semicolon to demarcate lower and upper bounds. This is to enable differentiation of the two uses of the colon.

Using the colon as co-ordinate field separator can leave some degree of ambiguity. For instance, a subscript expression of "12:34" could mean "use pixel indices 12 to 34", or could mean "use the single declination value 12:34". It is also harder to read, is an extra rule to remember, and could be error prone. For these reasons, the strong recommendation is to use the *dms* and *hms* notation for celestial and time co-ordinates.

Even given no sky co-ordinate range as in this example,

```
cube(12:34:56.7,-41:52:09,-100.0;250.0)
```

do not mix colons for the two uses in the same expression. Hence there is a semicolon to define the range in the third co-ordinate. The semicolon expression delimiter works with all classes of co-ordinate system. This example might extract a spectrum from a cube at the right ascension $12^{\rm h}34^{\rm m}56.7^{\rm s}$ declination $-41°52'9''$ between -100 and $250\,{\rm km\,s^{-1}}$ velocity.

You should also be aware that the non-integer co-ordinates within an NDF section apply to *WCS* axes, in contrast to integer bounds that define pixel indices along *pixel* axes. These are not necessarily the same. For example, the WCS axes may be rotated or permuted. If the WCS axes are rotated, the NDF section actually used will be a box just large enough to hold the requested range of WCS-axis values. Be careful.

9.3.2 Axis co-ordinates:

For axis co-ordinates double-precision arithmetic is used to process the section values, but either double- or single-precision notation may be used when supplying them. Both linear and non-linear *axis* co-ordinates are supported, the values supplied being automatically converted into the corresponding pixel indices before use. For instance,

```
spectrum(6500.0:7250.0)
```

could be used to select the appropriate region of a spectrum calibrated in Ångstroms, while

```
spectrum(6000.0~500.0)
```

would select a region of the spectrum approximately from 5750 to 6250.0 Ångstroms (the exact extent depending the values of the axis co-ordinates), and

```
spectrum(5500.0~21)
```

would select a 21-pixel-wide region of the spectrum centred on 5500 Ångstroms.

9.4 Specifying Fractional Extents

A further form for the subscript expression involves specifying a fractional position along each dimension as a percentage, as follows

```
name( p%~q%, r%:s%, ... )
```

where 'p%∼q%' specifies the centre and extent as percentages, and 'r%:s%' specifies a percentage range. Thus,

```
image(25%:75%,50%:~50%)
```

would refer to the central quarter of the image. Both sections are equivalent. The percentages are converted to the nearest pixel centre to decide the centre and extents of the sections.

The rules concerning omitted values before or after the delimiting ∼ apply. Thus,

```
image(40%~, ~50%)
```

could be used to refer to a full-width, half-height section centred at the (40%, 50%) fractional position.

A percentage value is not limited to the range 0–100%. In such circumstances the areas beyond the bounds of the NDF are set to bad. For example,

```
image(~110%, -5%:105%)
```

would give an enclosing border of bad pixels, extending 5% of the original dimensions. Both sections are equivalent.

9.5 Changing Dimensionality

The number of dimensions given when specifying an NDF section need not necessarily correspond with the actual number of NDF dimensions, although usually it will do so.

If you specify fewer dimensions than there are NDF dimensions, then any unspecified bounds will be set to (1:1) for the purposes of identifying the pixels to which the section should refer. Conversely, if extra dimensions are given, then the shape of the NDF will be padded with extra bounds set to (1:1) in order to match the number of dimensions. In all cases, the resulting section will have the number of dimensions you have actually specified, the padding serving only to identify the pixels to which the section should refer.

In KAPPA there are a number of applications that can only handle a fixed number of dimensions (*e.g.* DISPLAY, LINPLOT, MEDIAN). NDF sections permit such applications to have wider applicability, since the applications can operate on full NDFs of arbitrary dimensionality. So for instance, DISPLAY could show planes of a datacube.

9.6 Mixing Bounds Expressions

In the last example (in Section 9.3) both *axis* co-ordinates and pixel indices were mixed in the same subscript expression. In fact, any of the features described earlier may be combined when specifying an NDF section, the only restrictions are as follows.

(1) When the shape of the resulting section is expressed in pixel indices, the lower bound must not exceed the upper bound in any dimension.

(2) If the bounds for an axis are specified by centre and width values (rather than as lower and upper bounds), then a WCS value should not be used with a pixel index. That is, the centre and width values must both refer to the same co-ordinate system.

Thus, all the following might be used as valid specifications for NDF sections

```
ndf(3.7)
ndf(,5:)
ndf(-77:01h23m45s,,4)
ndf(66~9,4:17)
ndf(~5,6~)
ndf(~,:)
ndf(5500.0~150,)
ndf(2.137~10%)
ndf(3.0~1.5,-78.06D-3:13.0545,,,,)
```

Many other combinations are obviously possible. In cases where some bounds are given in pixel indices and some in WCS co-ordinates, two boxes will be formed; one representing the pixel-index bounds and one representing the WCS bounds. The actual NDF section used will be the overlap of the two boxes. The pixel box will inherit any pixel index limits supplied in the bounds expression, and will use default values for any missing limits. These default pixel-index bounds are just the bounds of the full NDF. Likewise, the WCS box will inherit any WCS limits supplied in the bounds expression, and will use default values for any missing limits. The default WCS limits are the bounds of a box that just includes the whole pixel box.

10 NDF History

During a spring clean of directories to free some space (what d' y'mean you don't?), most of us will have encountered data files whose purpose and worth are long forgotten. We're reluctant to remove them in case they contain irreplaceable data. Some people are very good and make copious notes... Even then the result of a casual experiment might not be recorded. For those who are lazy, such files can be a frequent dilemma. Even a quick look at a plot of the data is often of little assistance. As you've probably surmised, the NDF offers a solution.

Within an NDF you may record *history* information. This is usually a chronicle of the processing stages used to form the NDF, including the parameter values of the applications invoked; but it may also include commentary you provide, for example, the rationale for doing certain operations.

History is associated with individual NDFs; it is not some global attribute of a data-processing session. An NDF has a *history update mode*, which remains with the NDF and any descendant NDF, until the update mode is altered or the history erased. By default, the update mode is `"Disabled"`, meaning that no history recording occurs. To permit history recording you must first switch it on, selecting from three update modes—`"Quiet"`, `"Normal"`, and `"Verbose"`—which give increasingly more detailed information.

10.1 Control and Content of History Recording

Task HISSET lets you set the history update mode. The default is `"Normal"`, thus here the command

```
% hisset hr1068
```

switches normal history recording on for NDF hr1068. Thereafter whenever you alter this NDF, or create another NDF from it, the task automatically records the name of the application that was run, the date and time, a reference name that identifies the NDF, your username, and some text comprising the command-line parameters and the full path of the application. In KAPPA the package name and version is appended to the application name. Other packages may provide task-dependent additional text.

If disc space is not a concern, you might prefer the verbose level.

```
% hisset hr1068 verbose
```

the supplementary information being the machine type, and its operating system name and version.

For small datasets, such as spectra, the history can amount to a significant part of the NDF's size, so for these you might prefer the quiet level. This does not record the command line.

HISSET lets you switch off history recording, if you want to do something 'off the record', or erase the history altogether.

```
% hisset hr1068 disabled
% hisset hr1068 erase
```

Some applications create new NDFs from scratch, not inheriting the history records and update mode from an input NDF. Some examples are CREFRAME, TRANDAT, and PSF. Should you wish these to have history recording enabled as you create such NDFs, there is an environment variable NDF_AUTO_HISTORY that should be set to a non-zero integer value, or immediately run HISSET with the new NDF. Note that some applications may choose to disregard the value of NDF_AUTO_HISTORY for good reason, such as for ancillary NDFs created with an NDF extension. Whenever this option is exercised, the reference documentation for the task should indicate this behaviour in its Implementation Notes. There are currently no such applications in KAPPA.

10.2 Adding Commentary to History Recording

Once history recording is enabled, you can add commentary to an NDF using HISCOM.

```
% hiscom hr1068 i "There may have been cloud during the integration."
```

You aren't limited to single lines if you respond to the prompt for the comment. You can give a series of lines, terminated by supplying !.

```
% hiscom hr1068
COMMENT - Comment line > The dome may have been obstructing the telescope
COMMENT - Comment line > during the integration.  We are not sure that the
COMMENT - Comment line > filter is correct either.
COMMENT - Comment line > !
```

If you prefer, you may edit some text into a file and append its contents to the history records. Thus

```
% hiscom hr1068 f file=comments.lis
```

appends the text contained in comments.lis to the history records of NDF hr1068.

10.3 Listing History Records

At some point you will want to refer back to the history records. The HISLIST task does this.

```
% hislist hr1068

   History listing for NDF structure /home/scratch/dro/hr1068:

   History structure created 1995 Sep 24 11:16:15.000

1: 1995 Sep 24 11:16:15.000 - HISSET          (NDFPACK V1.0)

   Parameters: MODE='Normal' NDF=@hr1068
   Software: /star/bin/kappa/hisset
```

Before you ask... at present there are no parameters for selecting a time interval and there is no output of the machine and username, but they're not forgotten.

Here is another example showing a series of history records.

```
% hislist hr1068 \\

   History listing for NDF structure /home/scratch/dro/hr1068sm2:

   History structure created 1995 Nov 24 11:16:15.000

1: 1995 Sep 24 11:16:15.000 - HISSET          (NDFPACK V1.0)

   Parameters: MODE='Normal' NDF=@hr1068
   Software: /star/bin/kappa/hisset

2: 1995 Sep 24 11:19:53.000 - GAUSMOOTH       (KAPPA V0.9)

   Parameters: BOX=13 FWHM=5 IN=@hr1068 OUT=@hr1068sm TITLE=! WLIM=!
   Software: /star/bin/kappa/gausmooth

3: 1995 Sep 24 11:20:15.000 - HISSET          (NDFPACK V1.0)

   History update mode changed from NORMAL to VERBOSE.
   Parameters: MODE='Normal' NDF=@hr1068sm
   Software: /star/bin/kappa/hisset

4: 1995 Sep 24 11:20:49.000 - GAUSMOOTH       (KAPPA V0.9)

   Parameters: BOX=9 FWHM=3 IN=@hr1068sm OUT=@hr1068sm2 TITLE=! WLIM=!
   Software: /star/bin/kappa/gausmooth
   Machine: alpha, System: OSF1 214 (release V3.2)

5: 1995 Sep 24 11:22:32.000 - HISCOM          (NDFPACK V1.0)

   Parameters: MODE='Interface' NDF=@hr1068sm2 WRAP=TRUE
   Software: /star/bin/kappa/hiscom
   The dome may have been obstructing the telescope during the integration.
   We are not sure that the filter is correct either.
```

The first history item shows HISSET enabling history. This was followed by a smooth of the data with GAUSMOOTH. Then the recording level was set to verbose. The fourth record recalls another smooth, and this time you can see the machine details. Finally, some commentary is added with HISCOM.

11 The Graphics Database

Have you ever been in a situation where you would like an application to know about graphics drawn by some other programme? For instance, you display an image of the sky, then later you want to obtain the co-ordinates of the stars within the image via the cursor. There are two main approaches to achieving this functionality. The first is to duplicate the display code in the CURSOR application. This is wasteful and inflexible. The second is to store information about 'pictures' in a database that can be accessed by subsequent graphics programmes. This is the technique used by KAPPA.

Each graphics application that creates a display on a graphics device, also stores information describing the display in the *graphics database*. This is a file, which usually resides in the user's home directory, and is often referred to as the *AGI database*.[10] Displays are described in terms of *pictures*. A picture is basically a rectangular area on the graphics device within which an application produces graphical output. Each time an application creates a picture, the dimensions and position of the picture (together with other ancillary information) are stored in the graphics database. Subsequent applications can then read this information back from the database, and use it (for instance) to align new graphics with previously displayed graphics.

The best way to demonstrate the use of the graphics database is to give some illustrated examples.

11.1 The Graphics Database in Action

The following examples assume that KAPPA is loaded and the graphics device—an X-window containing a plotting area of 850 by 500 pixels—is available. To create such a window use the `xmake` (SUN/130) command:

```
% xmake xwindows -height 500 -width 850 -colours 64
```

This command limits the number of colours used by the X-window to 64. It is usually a good idea to be sparing with X-window colours. Creating X-windows with too many colours can restrict the availability of colours for other X applications.

First of all we make the X-window the current graphics device (as described in Section 6.1.1). This selection will remain in force until changed. The following command would not be necessary if the global parameter already had this value:

```
% gdset xwindows
```

Next we shall clear the X-window, and empty the graphics database of xwindows pictures (pictures relating to other graphics devices are retained):

```
% gdclear
```

[10] "AGI" is the name of the subroutine library that provides access to the graphics database. AGI stands for "Applications Graphics Interface", and is documented in SUN/48.

The display is now clear, but one picture remains in the graphics database; this is the BASE picture and corresponds to the entire plotting area. To understand why this BASE picture is required we need to introduce the idea of the *current picture*.

At any time, one of the pictures stored within the graphics database is nominated as the current picture. All graphical applications are written so that the graphical output that they create is scaled to fit within the current picture. Thus, the plotting area used by a graphics application can be controlled by selecting a suitable current picture before running the application. Since we have just cleared the database, the only picture remaining is the BASE picture, which consequently becomes the current picture. This allows subsequent applications to draw in any part of the plotting area.

We now load a grey-scale colour table into the X-windows and display an IRAS 12 μm image of M31. The pixel values are scaled so that 10% of the pixels appear as pure black, and 1% appear as pure white. We ensure that no annotated axes are produced (this will result in the image being a little larger since no margins need to be left for the axes):

```
% lutgrey
% display $KAPPA_DIR/iras noaxes
MODE - Method to define the scaling limits /'FLASH'/ > perc
PERCENTILES - Percentiles for scaling /[10,90]/ > 10,99
Data will be scaled from 0.05178363 to 1.051904.
```

The X-window should now look like Figure 1. The application has displayed the image in the middle of the current picture (the BASE picture), and has made it as large as possible, subject to the constraints that it must lie entirely within the current picture. Note, as with many IRAS images, equatorial north is downwards in this image

Let's now use the PICLIST command to look at the contents of the graphics database (press return in response for the prompt for Parameter PICNUM):

```
% piclist

  No. Name            Comment                   Label           Ref
  ------------------------------------------------------------------
C   1 BASE            Base picture
    2 DATA            KAPPA_DISPLAY                             Ref
PICNUM - Number of new current picture /!/ >
```

This shows us that there are now two pictures in the database, listed in the order in which they were created. The current picture is still the base picture, as indicated by the letter `C` at the left of the line describing picture number 1. The second picture was created by the KAPPA application DISPLAY, and has the name DATA, indicating that it contains a representation of a set of data values. DATA is one of four standard picture *names*. BASE is another of these standard names. We shall come across the other two shortly. The PICLIST application allows you to select a new current picture by supplying a picture number in response to the prompt for parameter PICNUM. Accepting the null default by pressing return causes the current picture on entry to be retained.

The above use of DISPLAY illustrates an important rule regarding the behaviour of most graphical applications; *the current picture is not changed by applications that produce graphical*

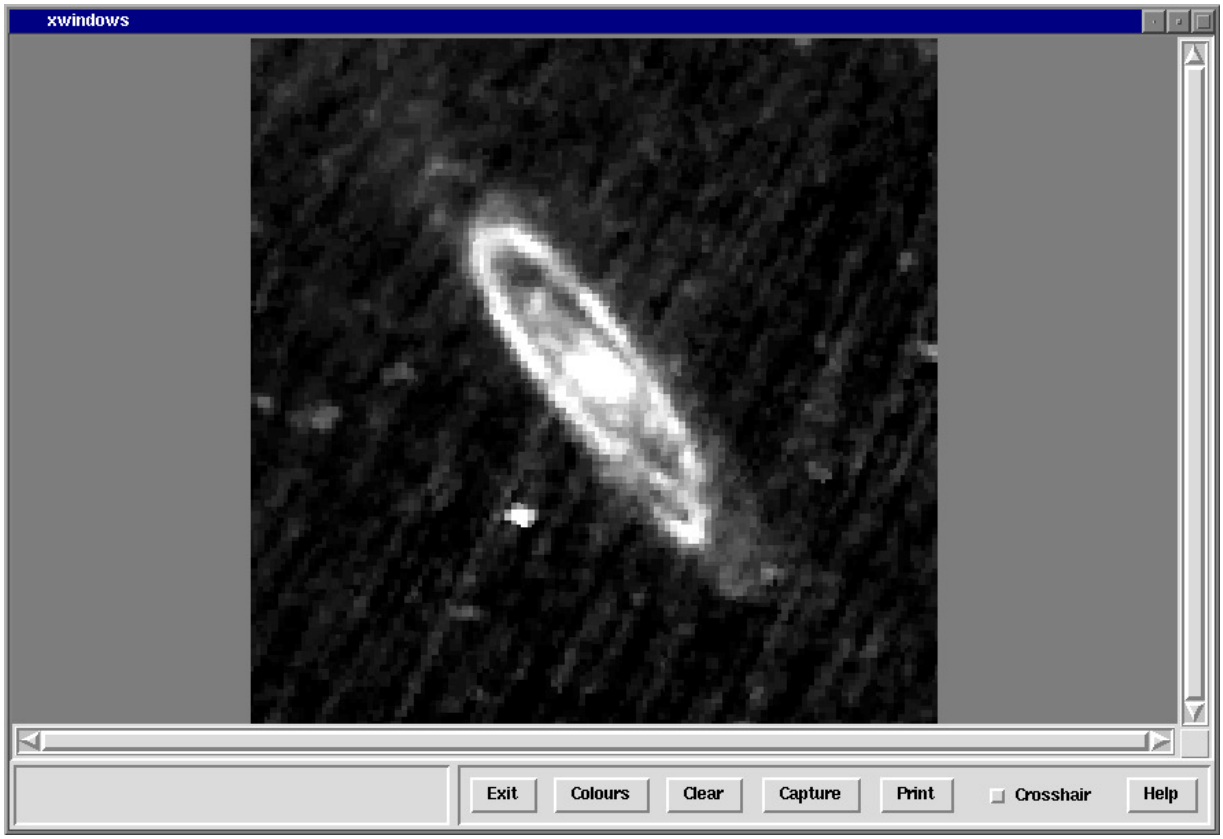


Figure 1: An IRAS 12 μm image of M31 displayed in the middle of the BASE picture.

output.[11] If it were not for this rule, pictures would become progressively smaller, vanishing into the distance, since new pictures cannot be drawn outside the current picture.

We will now display an optical image of M31. This time we will arrange for it to be placed towards the left hand side of the X-window. To do this, we first clear the whole X-window and graphics database using GDCLEAR:

```
% gdclear
```

We now create a new picture using the PICDEF command (note, the "\" characters are needed to prevent the UNIX shell interpreting the square bracket characters):

```
% picdef mode=cl fraction=\[0.6,1.0\] outline=no
```

The MODE parameter specifies the position of the new picture within the BASE picture; in this case "cl" indicates that the new picture is to be centred ("c") vertically within the BASE picture and placed at the left ("l") hand edge. The FRACTION parameter specifies the dimensions

[11] This rule does not apply to applications that manage the database itself rather than producing graphical output. Thus, for instance, it is legal for PICLIST to change the current picture if you request such a change. Also, an uncontrolled exit from an application, *e.g.* CTRL/C may leave the database in an abnormal state.

of the picture; the first value (`0.6`) gives the horizontal size of the picture as a fraction of the horizontal size of the BASE picture, and the second value (`1.0`) gives the vertical size of the picture as a fraction of the vertical size of the BASE picture. Thus, the new picture is just over half the width of the BASE picture, and is the full height of the BASE picture. The OUTLINE parameter specifies whether a box should be drawn on the screen showing the outline of the new picture. In this case we switch this option off.

If we run PICLIST again, we get:

```
% piclist

  No. Name            Comment                   Label           Ref
  ------------------------------------------------------------------
    1 BASE            Base picture
C   2 FRAME           KAPPA_PICDEF
PICNUM - Number of new current picture /!/ >
```

The picture created earlier by DISPLAY was deleted when we ran GDCLEAR. The BASE picture is still there (of course), and we also have the picture created by PICDEF. This is a FRAME picture, another of the four standard picture names. A FRAME picture acts as a 'frame' for other pictures. A FRAME picture can itself contain other nested FRAME pictures, together with DATA and KEY pictures. The picture created by PICDEF is the current picture (indicated by the letter `C` again), and so subsequent graphics applications will arrange for any pictures they create to fall entirely within this FRAME picture.

Now display the image, this time including annotated axes around the edges.[12] DISPLAY will ensure that all its output (both the image and the axis annotation) fall within the current picture (*i.e.* the picture created by PICDEF above). We scale the pixel values to produce a negative image in which 1% of the pixels appear black and 30% appear white:

```
% display axes
IN - NDF to be displayed /@$KAPPA_DIR/iras/ > $KAPPA_DIR/m31
MODE - Method to define scaling limits /'perc'/ >
PERCENTILES - Percentiles for scaling /[10,99]/ > 99,30
Data will be scaled from 10854.78 to 3889.016.
```

The X-window should now look like Figure 2. We can use PICLIST again to list the pictures stored in the database:

```
% piclist
  No. Name            Comment                   Label           Ref
  ------------------------------------------------------------------
    1 BASE            Base picture
C   2 FRAME           KAPPA_PICDEF
    3 FRAME           KAPPA_DISPLAY
    4 DATA            KAPPA_DISPLAY                             Ref
PICNUM - Number of new current picture /!/ >
```

[12] The current co-ordinate Frame in the image being used is RA/DEC and so the axis will be annotated in RA and DEC.

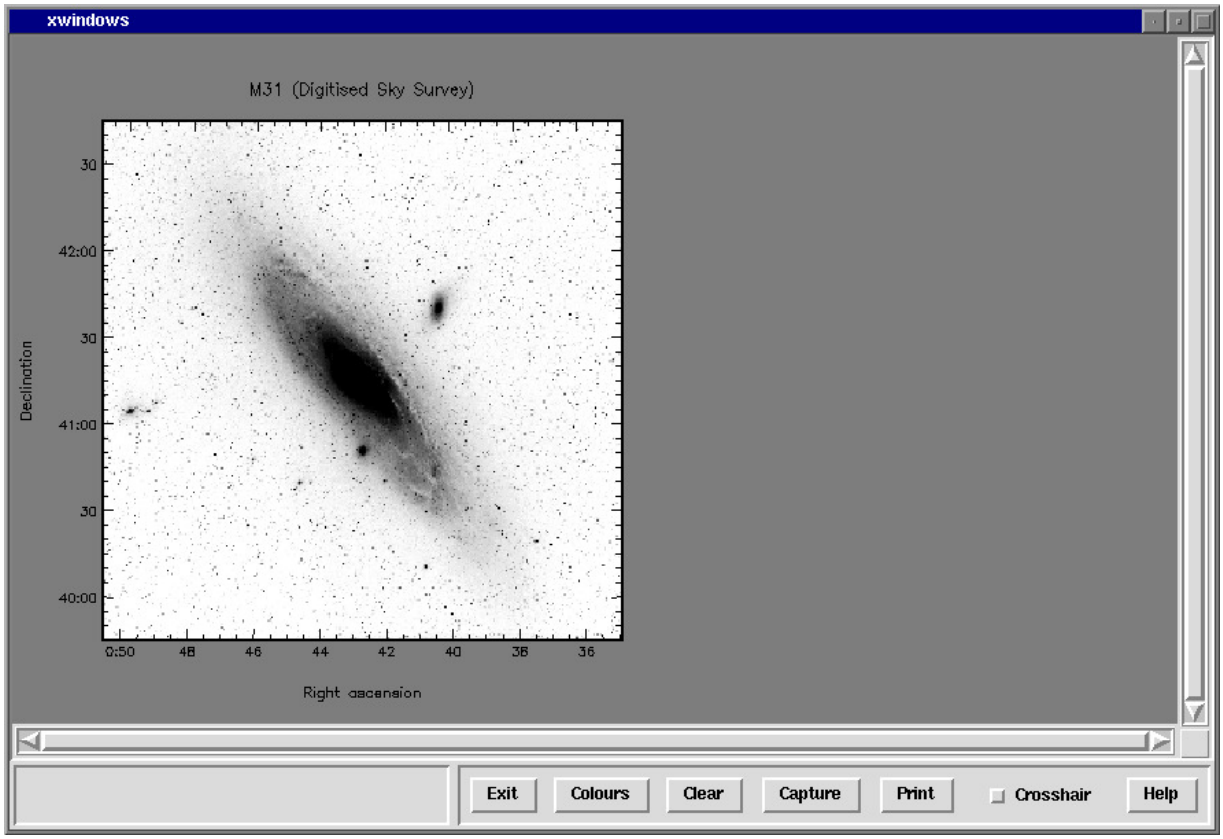


Figure 2: Optical M31 image with axes displayed toward the left of the BASE picture.

There are four pictures this time; the BASE picture, the picture created by PICDEF (number 2), and two pictures created by DISPLAY (numbers 3 and 4). Picture number 2 was made the current picture by PICDEF and as explained above, DISPLAY did not change this. DISPLAY creates two pictures, one (the DATA picture) containing just the image area itself, and another (the FRAME picture) to act as a frame for the DATA picture and the annotated axes.

Let's say you wanted to display an enlarged sub-section of the image in the top-right corner of the X-window. First, you need to decide on the bounds of the sub-section to be displayed. Here, we use a graphics cursor to indicate the bottom left and top-right corners of a box enclosing the required area. Click the left mouse button at the bottom left and top right corners of a box enclosing the required sub-section of the image:

```
% cursor showpixel plot=box maxpos=2

  Use the cursor to select up to 2 positions to be reported.
    To select a position press the space bar or left mouse button
    To forget the previous position press "d" or the middle mouse button
    To quit press "." or the right mouse button

 Picture comment: KAPPA_DISPLAY, name: DATA, reporting: SKY co-ordinates

 RA = 0:41:33.5 (hh:mm:ss.s)   Dec = 40:33:01 (ddd:mm:ss)
```

```
(172.4       78.2)

RA = 0:39:25.0    Dec = 40:54:17
(212.8       113.9)
```

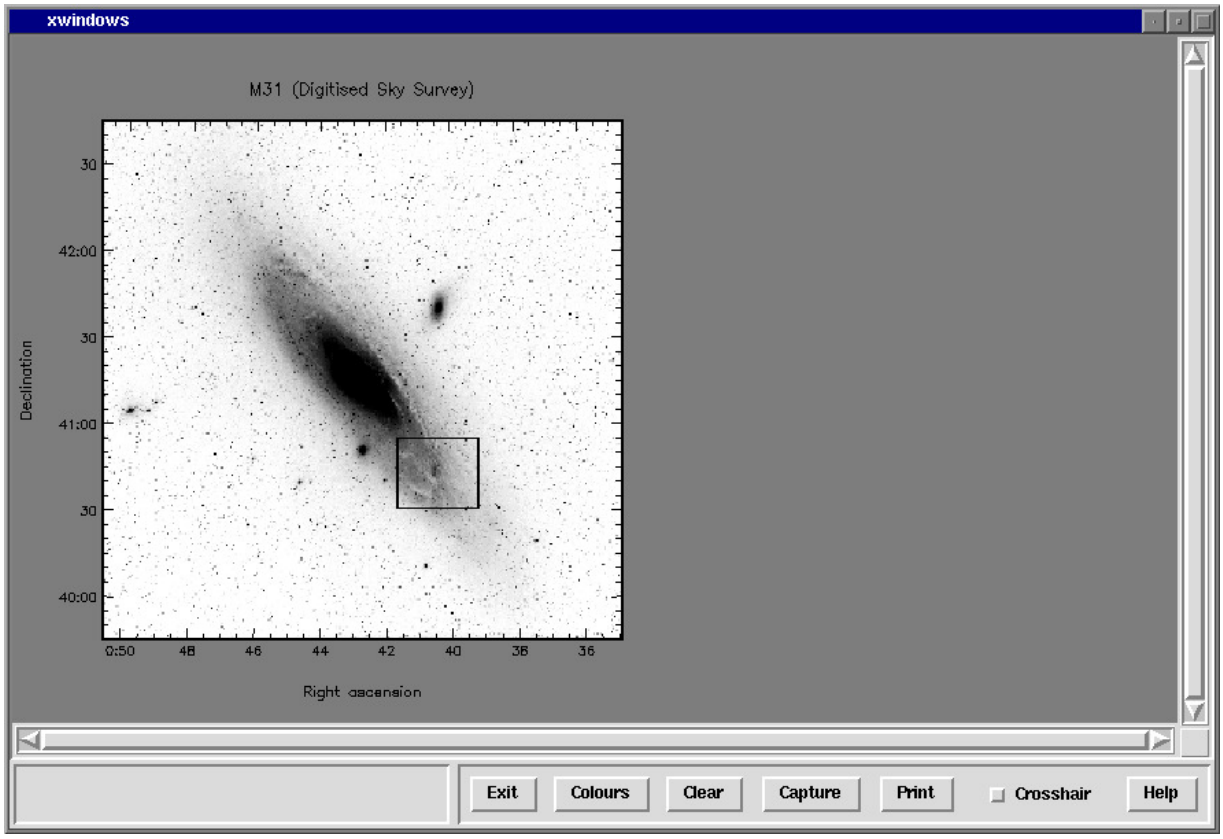


Figure 3: A box is drawn using the CURSOR application.

The box I selected is indicated in Figure 3.

When an application such as DISPLAY produces a DATA picture containing a representation of an NDF, it saves a copy of the NDF's WCS component (which contains co-ordinate Frame information) with the DATA picture in the graphics database. When CURSOR subsequently reports a position, it uses this saved WCS information to convert the graphics co-ordinates at the cursor into any of the available co-ordinate Frames. By default, CURSOR reports positions in the co-ordinate Frame that was current when the NDF was displayed (RA/DEC in this case), but other co-ordinate Frames may be requested using the FRAME parameter (*e.g.* `FRAME=PIXEL` displays pixel co-ordinates). In addition to the co-ordinate Frames inherited from the NDF, there are also four extra Frames available.

GRAPHICS – specifies positions in terms of millimetres from the bottom left corner of the graphics device (*e.g.* X-window or paper).

BASEPIC – similar to GRAPHICS but specifies positions in a normalised co-ordinate system in which the shorter dimension of the screen or paper has a length of 1.0 (the scales on both axes are equal).

NDC – similar to BASEPIC but specifies positions in a normalised co-ordinate system in which the bottom-left corner has co-ordinates $(0, 0)$ and the top-right corner has co-ordinates (1,1). Thus, for a non-square device the scales on the two axes will be different.

CURPIC – similar to BASEPIC except that it covers only the specified picture. Thus, the bottom-left corner of each picture is at $(0, 0)$ and the shorter dimension of each picture has length 1.0.

Going through the parameters supplied to the above CURSOR command, the first one (SHOWPIXEL) causes the pixel co-ordinates at each selected position to be displayed, in addition to the co-ordinates selected using Parameter FRAME (which defaults in this case to RA/DEC since FRAME was not specified). For instance, the first position is at a right ascension of $0^{\rm h}41^{\rm m}33.5^{\rm s}$ and a declination of $+40°33'1''$, and has pixel co-ordinates $(172.4, 78.2)$.[13] The second parameter (`PLOT=BOX`) causes a box to be drawn on the screen between each pair of positions. There are several other allowed values for the PLOT parameter, which mark the positions in different ways (markers, poly-lines, chains, text, *etc.*). The last parameter (`MAXPOS=2`) is purely a convenience, and causes the application to terminate when two positions have been supplied. Without this, you would need to press the right mouse button once the two positions had been given to indicate that you do not want to supply any more positions.

We now need to create a new FRAME picture to contain the magnified image section. We have seen how PICDEF can be used to create a FRAME picture of a given size at a given position within the BASE picture, but there are also several other ways in which PICDEF can be used. Here, we use PICDEF in 'cursor' mode; the bottom left and top-right corners of the new picture are specified by pointing and clicking with the mouse.[14] We use this mode to create a new FRAME picture occupying the area to the top right of the existing image:

```
% picdef mode=cursor nooutline

  Use the cursor to select 2 distinct points.
    To select a point press the space bar or left mouse button
    To quit press "." or the right mouse button

Co-ordinates are ( 0.9155139, 0.5470942 ) and ( 1.683106, 0.9799599 )
```

The co-ordinates displayed by PICDEF are BASEPIC co-ordinates.

We now display the required image section in this new FRAME picture. The screen output from CURSOR above shows that the selected image section runs from pixel 173 to 213 on the first (X) axis, and from pixel 79 to 114 on the second (Y) axis. To display just this section we include the pixel bounds in the specification of the input NDF when we run DISPLAY:

[13] Pixel *co-ordinates* are fractional, where-as pixel *indices* are integer. The pixel with indices $(1, 1)$ covers a range of pixel co-ordinates between 0.0 and 1.0 on each axis, with its centre at pixel co-ordinates $(0.5, 0.5)$.

[14] Note, by default, the FRAME picture created by PICDEF is *not* constrained to be within the current picture, it can be anywhere within the BASE picture.

```
% display border noaxes mode=scale high=3889.016 low=10854.78
IN - NDF to be displayed /@m31/ > m31(173:213,79:114)
Data will be scaled from 10854.78 to 3889.016.
```

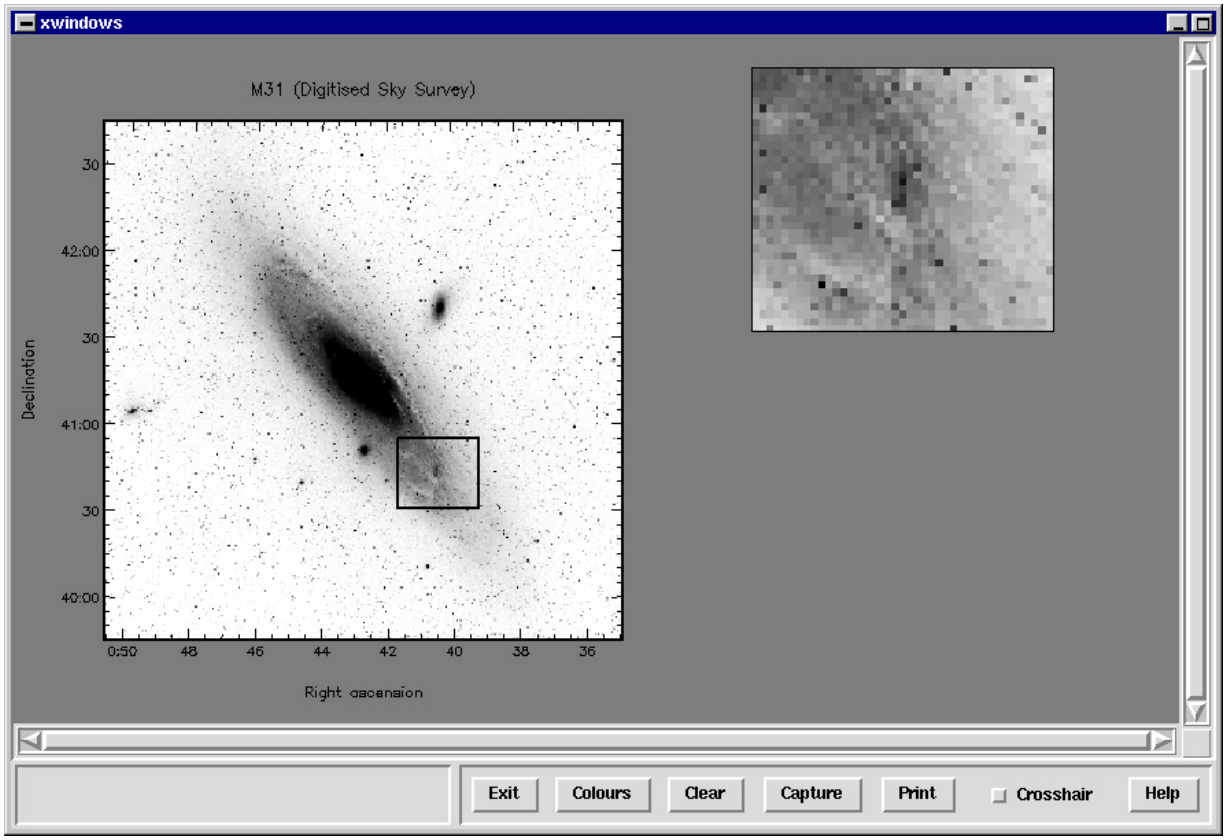


Figure 4: The selected section of the NDF is re-displayed.

The X-window should now look like Figure 4.

The string `m31(173:213,79:114)` is called an *NDF section specifier*. These are described more fully in Section 9. The image is surrounded by a thin border instead of fully annotated axes in order to make the image larger. This is achieved using the BORDER and NOAXES keywords (equivalent to setting BORDER=YES and AXES=NO). The pixel scaling was specified explicitly using parameters HIGH and LOW in order to ensure that the image was displayed with the same grey-scale as the main image (the high and low data values are the data values corresponding to black and white—or white and black if reversed – and were reported when the main image was displayed earlier).

Remember the IRAS image of M31 we started with? We'll now overlay contours of the IRAS image on top of the magnified section of the visual image we have just displayed. Don't forget, these two images have not been aligned—for instance, north is up in the visual image, but down in the IRAS image. However, both images have information within the WCS component describing the relationship between pixel co-ordinates and RA/DEC. This WCS information was stored with the DATA picture created by DISPLAY above, and so can be used by the CONTOUR

application in order to draw the IRAS contours in alignment with the displayed visual image.[15] In addition, we also use CURSOR to mark a position with a text string giving a title for the contour plot:

```
% contour noclear noaxes nokey iras mode=perc percentiles=\[55,75,95\]
Alignment has occurred within the SKY Domain.

Contour heights used:
0.5499523,   0.7577766,   1.012756.

% cursor plot=text maxpos=1 minpos=1 strings='"IRAS 12 \gmm contours"'

Use the cursor to select 1 position to be reported.
  To select a position press the space bar or left mouse button
  To quit press "." or the right mouse button

  Picture comment: Base picture, name: BASE, reporting: BASEPIC
  co-ordinates
  X = 1.308796      Y = 0.9839679
```

The X-window should now look like Figure 5.

The most important parameter in the above invocation of CONTOUR is the NOCLEAR keyword (equivalent to CLEAR=NO). This tells CONTOUR not to clear the graphics device before drawing the contours. Instead, CONTOUR looks for an existing DATA picture contained within the current picture. If one is found, then CONTOUR attempts to align the contours using the WCS information stored with the existing DATA picture. In our case, the current picture is still the top-right FRAME picture created by PICDEF, and so the contours are aligned using the WCS information stored with the DATA picture contained within this FRAME picture (*i.e.* the magnified image section). CONTOUR tells us that this alignment occurred within the 'SKY Domain'—if either of the two NDFs had not contained a calibration in a suitable celestial co-ordinate system, then alignment on the sky could not have been performed. In this case, CONTOUR would have aligned the contours in the 'PIXEL Domain' (*i.e.* in pixel co-ordinates).

Since we produced no annotated axes, a title string was added using the text drawing abilities of CURSOR. The pointer was positioned above the contour plot, and the left button clicked. The specified text string was drawn centred at this cursor position. Note, the string `\gm` is a 'PGPLOT escape sequence' that represents the Greek letter mu ("μ"). See the PGPLOT manual for a detailed description of the available escape sequences.

Let's say we wanted to compare the data values in the visual and IRAS images along a given line through the magnified sub-section. To do this, we first use the application PROFILE to create a pair of one-dimensional NDFs containing the data values along the line in each of the two images. We then display these one-dimensional NDFs using application LINPLOT. First we choose the line along which the profiles are to be taken, using CURSOR (again!):

```
% cursor plot=chain marker=3 style='colour=white' outcat=zz maxpos=2
```

[15] The accuracy of this alignment will depend on the accuracy of the astrometry information supplied with the image.

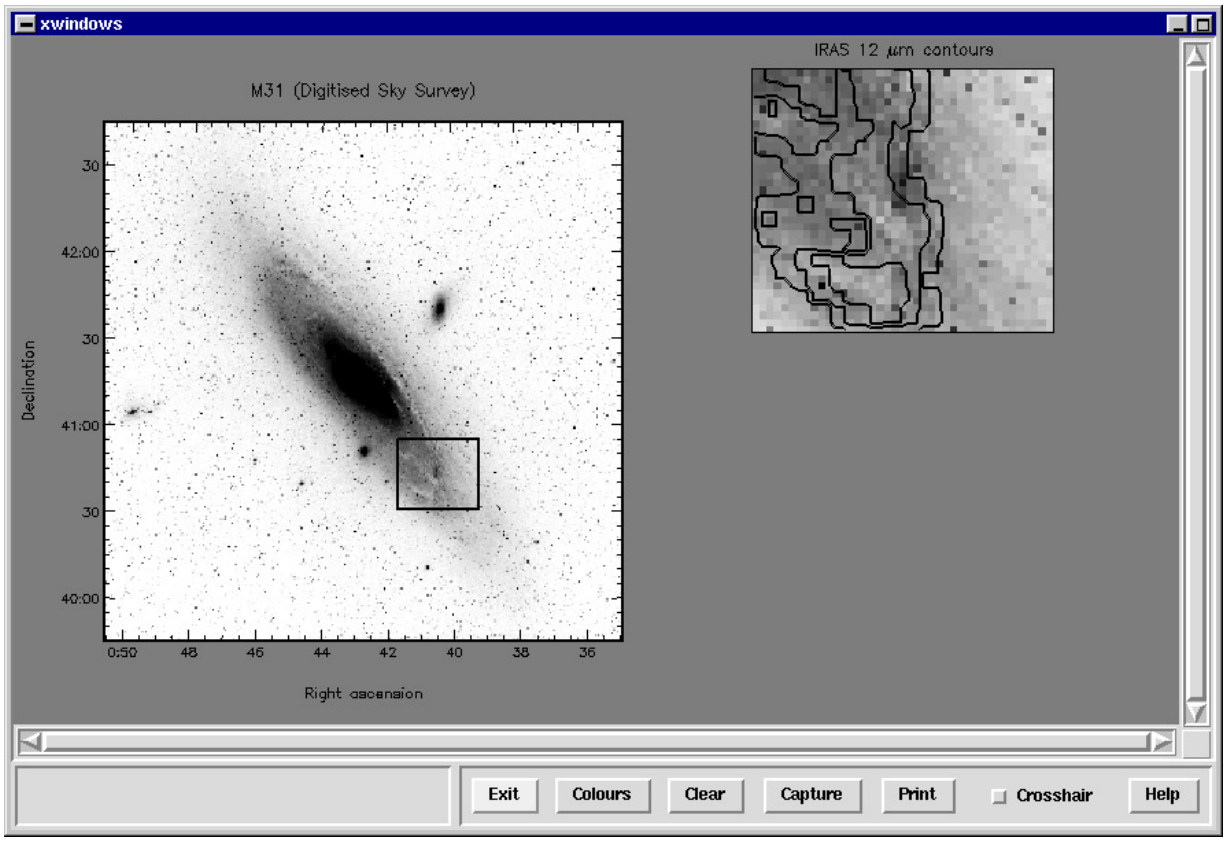


Figure 5: IRAS contours overlayed on the visual image.

```
Use the cursor to select up to 2 positions to be reported.
  To select a position press the space bar or left mouse button
  To forget the previous position press "d" or the middle mouse button
  To quit press "." or the right mouse button


Picture comment: KAPPA_CONTOUR, name: DATA, reporting: SKY co-ordinates
  RA = 0:41:22.7 (hh:mm:ss.s)   Dec = 40:35:12 (ddd:mm:ss)
  (175.8467    81.82527)

  RA = 0:39:56.5    Dec = 40:50:27
  (202.917     107.3986)
```

The pointer is positioned at the two ends of the required profile, and the left button clicked at each position. The supplied positions are marked by two markers of PGPLOT type 3 (small crosses), and a white line is drawn between them (forming a 'chain'). The selected positions are written to an output catalogue stored in file `zz.FIT`. We now sample the data in the two images along this profile to create a pair of one-dimensional NDFs (`m31_prof` and `iras_prof`):

```
% profile m31 incat=zz out=m31_prof
  Alignment has occurred within the SKY Domain.
```

```
  Profile contains 38 samples.

% profile iras incat=zz out=iras_prof
  Alignment has occurred within the SKY Domain.
  Profile contains 38 samples.
```

We want to draw the two line profiles in a new plot in the clear area at the bottom right of the X-window. We therefore need to create a new FRAME picture. This time we use PICDEF in 'XY mode', in which the bounds of the new FRAME picture are given explicitly using parameters UBOUND and LBOUND (we could have used cursor mode again; we use XY mode just to explore the different possibilities):

```
% picdef mode=xy lbound=\[0.93,0.03\] ubound=\[1.67,0.54\] nooutline
  Bounds are ( 0, 0 ) and ( 1.701635, 1 )
```

The bounds are supplied in the BASEPIC Frame (the bounds reported by the application are the bounds of the entire BASE picture). The new FRAME picture is, as usual, made the current picture by PICDEF and so any subsequent graphics applications will draw in the new FRAME picture.

We now draw the first of the two line profiles:

```
% linplot m31_prof style='"tickall=0,colour(curve)=black, \
          drawtitle=0,label(2)=DSS data value (black)"'
```

The X-window should now look like Figure 6. The plotting attributes specified by the STYLE parameter do the following; `tickall=0` stops tick marks from being drawn on the un-labelled edges (*i.e.* the top and right edges); `colour(curve)=black` specifies that the data curve is to be drawn in black; `drawtitle=0` prevents a title being drawn at the top of the plot; `label(2)=DSS data value (black)` specifies the textual label for the left hand edge.

We now draw the second line plot over the top of the first line plot:

```
% linplot iras_prof noclear noalign style='"edge(2)=r,tickall=0, \
          colour(curve)=white,drawtitle=0, \
          label(2)=IRAS data value (white)"'
```

Again, the NOCLEAR keyword prevents LINPLOT from clearing the graphics device before drawing the new line plot. Instead, the new line plot is drawn within the same axes as any existing line plot within the current picture. By default, the new line plot would adopt the bounds of the two existing axes. This would be inappropriate in this case since the two images have very different data scales. To prevent this, we specify the NOALIGN keyword, which causes the default bounds for the new axes to be determined from the supplied data instead of the existing line plot. The STYLE parameter is similar to the previous line plot except that the data values are annotated on the right hand edge (`edge(2)=r`), the data curve is drawn in white, and data label is different.

The X-window should now look like Figure 7.

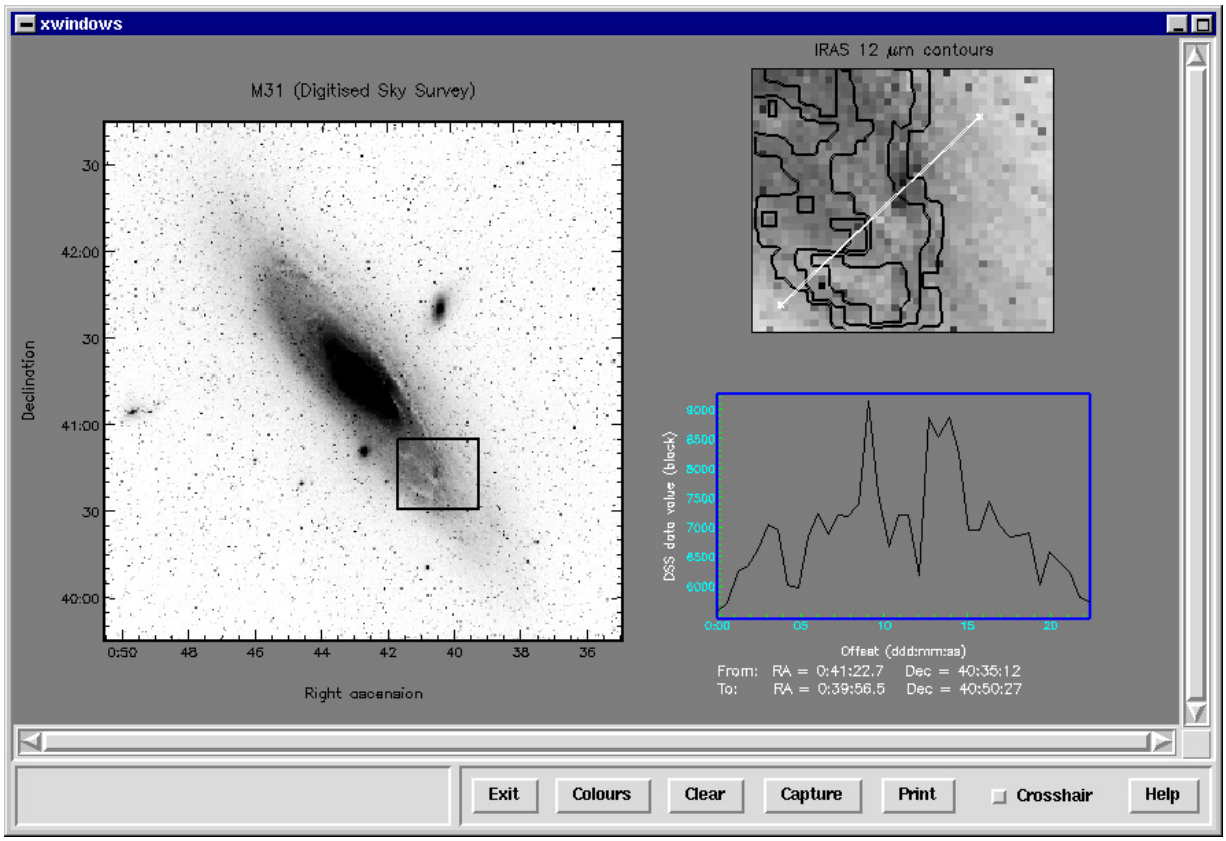


Figure 6: Data trace through the visual image.

As a final touch, we will add two 'warp' lines to the display joining the corners of the box marking the enlarged image area to the corresponding corners on the enlarged image. CURSOR can be used to draw these lines, but we need to take a little care. Normally, graphics produced by CURSOR will be clipped at the edge of the picture in which the supplied positions fall. In our case, the required lines start in one picture (the main image display DATA picture), but end in another (the magnified image DATA picture). To avoid the lines being clipped when they leave the main image DATA picture, we first ensure that the current picture is the BASE picture (*i.e.* the whole screen), and we tell CURSOR to report positions within the current picture. Normally, CURSOR reports each position within the most recent picture under the cursor, but setting parameter MODE=CURRENT when running CURSOR means that the reported positions always refer to the co-ordinate system of the current picture. So, first make the BASE picture the current picture. This is done using PICLIST, remembering that the BASE picture is always picture number 1:

```
% piclist picnum=1
```

We now run CURSOR to draw the first warp line:

```
% cursor plot=poly mode=current maxpos=2
```

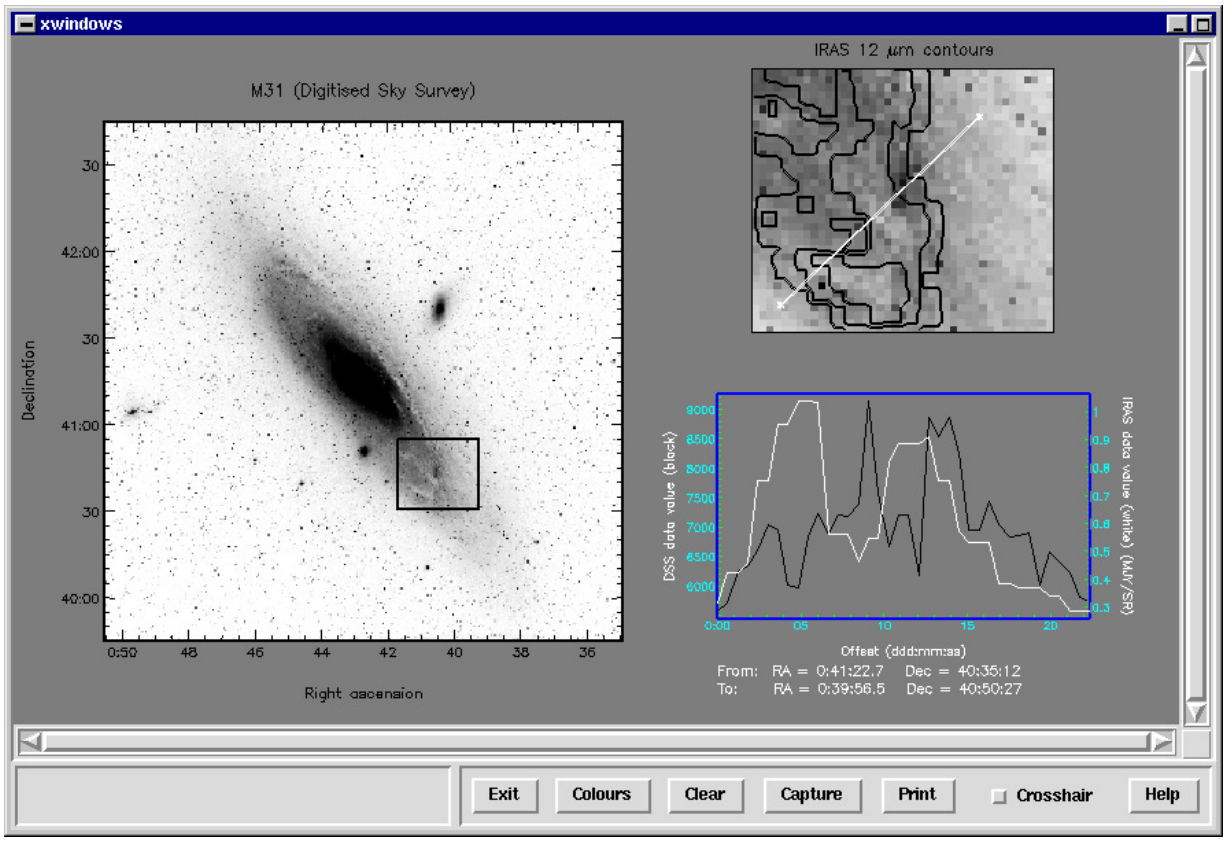


Figure 7: Data traces through both images.

Position the pointer over the top left corner of the black box marking the selected image section in the main image display, and click the left button. Then position the pointer over the top left corner of the border surrounding the enlarged image display, and click again. A line is drawn between these two points.

We now run CURSOR again to draw the second warp line:

```
% cursor plot=poly mode=current maxpos=2
```

Point and click over the bottom right corners of the two boxes this time. The final X-window should look now look like Figure 8.

11.2 Other Graphics Database Facilities

Various other facilities related to the graphics database exist as well as those described in the previous section. This section gives a brief description of a few more:

- A 'label' can be associated with a picture, using application PICLABEL. This provides a convenient 'handle' by which pictures can be referred to. For instance:

```
% piclabel fred
```

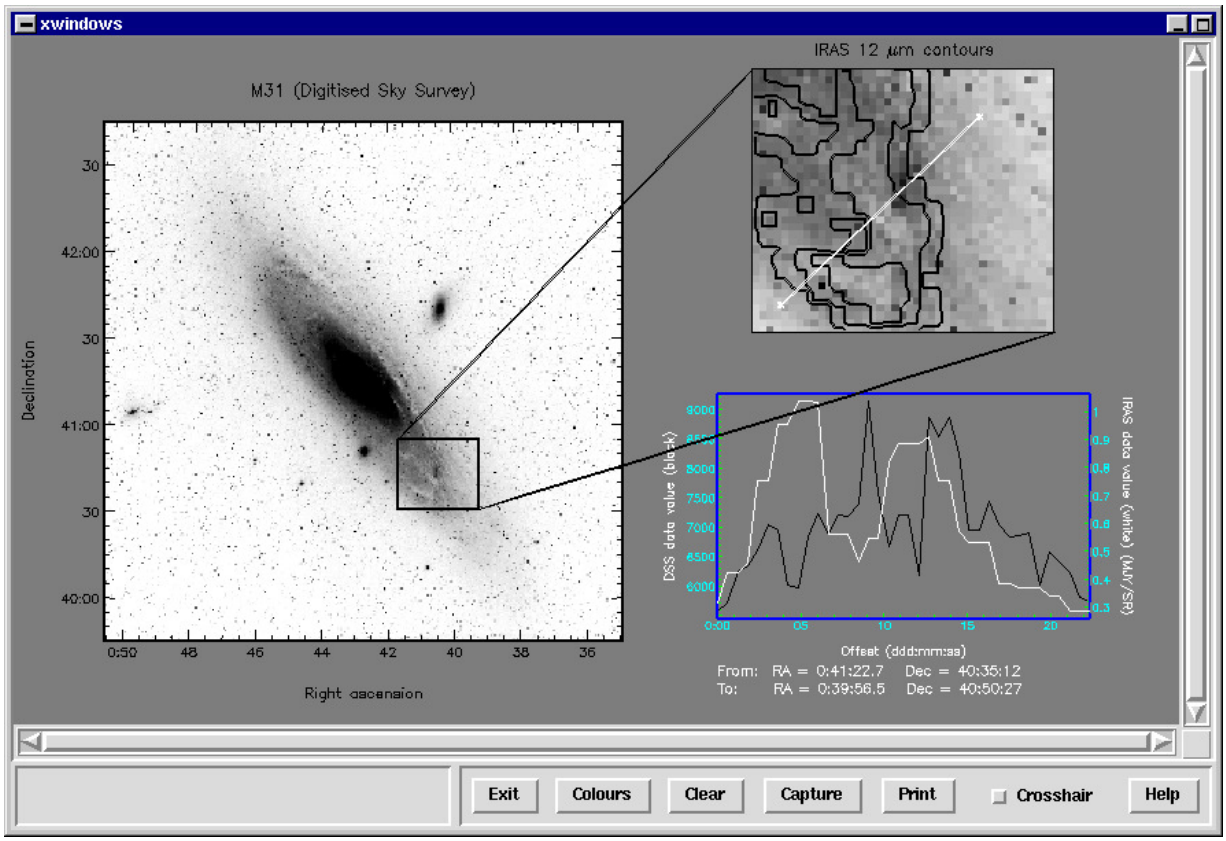


Figure 8: The complete display with warps.

will give the label FRED to the current picture. This picture could be made current again at a later time by re-selecting it using PICSEL:

```
% picsel fred
```

Any label associated with a picture is displayed in the list produced by PICLIST.

- PICDEF has one further mode—Array. This enables you to create an $n \times m$ grid of new FRAME pictures. It also has a mechanism for labelling all the pictures, so you can easily switch between the elements of the picture array. You might use the following command in a shell script to display a series of up to twelve spectra:

  ```
  % picdef mode=a prefix=spec xpic=3 ypic=4
  ```

  The bottom-left picture would be labelled SPEC1 and the rest are numbered in sequence from left to right to SPEC12—the top-right picture. You'd call PICSEL to select each picture in turn via a `while` loop in a C-shell script (see Section 19.1). Since this is a common operation a shorthand command, PICGRID, is available. For instance:

  ```
  % picgrid 3 4
  ```

  is equivalent to the previous example, except that the pictures are labelled 1 to 12.

- You can see that montages of pictures can rapidly be built. Occasionally, you will want some earlier picture to become the current picture. As we've seen, a labelled picture can be recalled via PICSEL, but not all pictures will be labelled, especially ones with name DATA, because of the rule that applications must not change the current picture. Another way to select a new current picture is via the command PICCUR. It displays a cursor. Move the cursor to lie on top of the picture you require and select a point following the instructions (usually by pressing the left-button of the mouse), then exit (normally by hitting the right-hand mouse button). Generally, this will be fine, but you can have cases where one plot is still visible through a transparent plot drawn subsequently. If the later picture extends entirely over the image you require, PICCUR will not let you access it. The moral is "be careful when arranging your pictures". A picture may only be partially obscured, so by moving the cursor around and hitting the left-hand button you can often find a portion that is topmost. PICCUR reports the name, comment and the label (if it there is one) of the picture in which the cursor is located to assist you. It is usually quite obvious where pictures begin and end, so in practice it is easier than described here.
- If you do get lost or forget what and where the current picture is, the GDSTATE command will come to your rescue. You can even plot an outline with the OUTLINE keyword if you can't visualise the device co-ordinates. In the following example, the current picture does not have a label. If it did this too would be listed by GDSTATE:

```
% gdstate

Status of the xwindows window graphics device...

   The current picture is a FRAME picture.
   Comment: KAPPA_PICDEF
   Current co-ordinate Frame: BASEPIC

Picture bounds in the BASEPIC Frame:
   Axis 1 (X) : 0.516 to 0.891
   Axis 2 (Y) : 0.010 to 0.605
```

11.3 The Co-ordinate Frames Associated with a Picture

Each picture in the graphics database has associated with it several co-ordinate Frames. Some of these describe positions within the displayed data array, and others describe the corresponding positions on the graphics device. Each Frame is referred to by a *Domain name* (see Section 12.2 for more information about co-ordinate Frames and Domains).

The following Domains may be used to specify positions within any picture:

GRAPHICS — This gives positions in millimetres from the bottom left corner of the graphics device.

BASEPIC — This gives positions within a normalised co-ordinate system spanning the BASE picture (*i.e.* the entire graphics device). The units on both axes are set so that either the width or the height of the graphics device, whichever is smaller, is set to 1.0 (put another way, a unit square would fit in the picture and span the shorter axis). The bottom-left corner of the graphics device is (0, 0).

NDC — Normalised Device Co-ordinates, which are similar to BASEPIC but specifies positions in a normalised co-ordinate system in which the bottom-left corner has co-ordinates (0, 0) and the top-right corner has co-ordinates (1, 1). Thus, for a non-square device the scales on the two axes will be different.

CURPIC — similar to BASEPIC except that it covers only the specified picture. Thus, the bottom-left corner of each picture is at (0, 0) and the shorter dimension of each picture has length 1.0.

DATA pictures have additional co-ordinate Frames inherited from the WCS information (see Section 12) associated with the displayed data. The details of the available Frames will depend on the application that created the picture and the nature of the data. For instance, if an image is displayed in which the current Frame is a SKY Frame (calibrated in RA/DEC for instance), then the Domains GRID, PIXEL, AXIS and SKY will also be available.

Pictures created by older applications that do not yet support WCS information will not have all of these Frames. They will still have GRAPHICS, BASEPIC, NDC and CURPIC Frames, but the only additional Frames will be:

AGI_WORLD - This corresponds to AGI 'world' co-ordinates.

AGI_DATA - This corresponds to AGI 'data' co-ordinates.

The interpretation of both these Frames will depend on the application that created the picture, and should be described in the associated documentation. Note, the AGI_DATA Frame may not always be present. This again depends on the application.

11.4 The Graphics Database File

The location of the graphics database file can be controlled by setting the environment variable `AGI_USER` to the desired location. If `AGI_USER` is undefined, the database file is placed in your home directory. Thus by default the file is `$HOME/agi_`<*node*>`.sdf`, where you substitute your computer's node name for <*node*>, *e.g.* `/home/dro/agi_rlsaxp.sdf`. A new database is created when you run a graphics application if none exists. AGI will purge the database for a device if the graph window has changed size, or if you switch between portrait and landscape formats for a printer device.

The contents of the graphics database are ephemeral. Therefore you should regularly purge the database entries with GDCLEAR or delete the database file.

If a graphics application is aborted abnormally (for instance by pressing `CTRL/C`), the graphics database may be corrupted, potentially resulting in various forms of peculiar behaviour when subsequent graphics applications are run. If you get 'strange' behaviour when running a graphics application, deleting the graphics database file and starting again will often cure the problem.

11.5 Working With PostScript Files

Having produced a pretty picture in an X-window, how do we get it on to a piece of paper? There are basically two ways. The simplest way is to do a screen-shot (*i.e.* copy the pixel values from the X-window into a PNG file, or some other suitable graphics format), and then print the file on a suitable printer. There are many ways to do this, depending on your operating system (`GIMP`, `ImageMagick`, `ksnapshot`, `print screen`, *etc*). The figures in this chapter were created using this method. It's easy—but the results only have the resolution of your X screen (typically 72 dots per inch) which is usually not good enough for general publication.

The second method is more involved, but retains the full resolution of your printer (typically 600 dots per inch or more), producing far superior results. It involves running all the KAPPAapplications again, but using a suitable encapsulated PostScript graphics device instead of the X-windows device used earlier in this chapter. Each application will then write its graphical output to one or more disk files. If you elect to use one of the "accumulating" PostScript devices (as indicated in the device description displayed by GDNAMES), then the PostScript output from each application will be merged automatically into a single file. If you use one of the other PostScript devices, each application will produce a separate output file, all of which must be combined using PSMERGE to create a single PostScript file containing the entire plot, which can be printed or included in another document.

We will look at these method in more detail in the rest of this section.

11.5.1 The Choice of Graphics Device

If you want to combine the results of several applications into a single plot, you need to specify an *encapsulated* PostScript device. These differ from normal PostScript files in that they contain information which allows them to be included within other PostScript files. The GDNAMES command displays a list of all the available graphics devices, together with a brief description of each. This should enable you to select an appropriate device name.[16] There may be various encapsulated PostScript devices available. For instance, you can choose between colour or monochrome devices, and between portrait or landscape devices. The choice of colour or monochrome obviously depends on the printer you will use. The choice of landscape or portrait depends on the shape of the total plot you are producing—for tall narrow plots choose portrait, for short wide plots choose landscape.[17]

Creating composite graphical output will be easier if you choose one the devices that are described as "accumulating", as this will avoid the need for you to merge separate PostScript files yourself.

For instance, if you want all applications to write graphical output to a single colour landscape encapsulated PostScript file set the graphics device as follows:

```
% gdset apscol_l
```

[16] Most encapsulated PostScript devices start with the string `"epsf"` or `"aps"`.

[17] The choice or portrait or landscape may also affect the orientation of the plot when the resulting PostScript file is included in a document.

11.5.2 The PostScript Files

Having selected a suitable encapsulated PostScript graphics device, each KAPPA application will either modify an existing graphics file, or produce an new output file in your current directory containing PostScript commands, depending on whether you choose an accumulating device or not. These files can be examined without needing to be printed by using OKULAR, EVINCE, `ghostview`, *etc*. By default, the output file will be called `pgplot.ps`.

> *Note, if you choose not to use an accumulating device, subsequent graphics commands will overwrite any file created by earlier graphics commands. For this reason you should rename the output file after running each graphics command. An alternative approach is to assign a unique value to the DEVICE parameter for each graphics command (rather than just allowing it to default to the value of the global parameter set using GDSET). For instance,* `DEVICE="epsf_l;contour.ps"` *would cause the command to write its output to file* `contour.ps`*.*

11.5.3 Combining the Files into a Single File

This section is only relevant if you are using a non-accumulating device.

Once all applications have been run, you will have a potentially long list of PostScript files in your current directory (unless you use an accumulating device, in which case you will have only one). These need to be stacked together to produce a single PostScript file which can either be printed or included in a document. To do this, we use PSMERGE as follows[18]:

```
% psmerge display.ps contour.ps stars.ps > total.ps
```

This stacks the three specified PostScript files into a single file called `total.ps`. This will be a normal (*i.e.* not encapsulated) PostScript file, so you can print it, but it will be difficult to include it in a document. If you include the `-e` option when running PSMERGE, then an encapsulated PostScript file is created instead:

```
% psmerge -e display.ps contour.ps stars.ps > total.ps
```

The output file can be included in other documents but often causes problem when being printed.

Note, the order in which you supply the input files is important because later files are pasted 'on top of' earlier files, and so can potentially obscure them. In general, you need to list the input files in the order in which they were created.

PSMERGE has other facilities which allow you to scale, shift and rotate individual input files before including them in the output. See SUN/164 for details.

[18] You can include all the PostScript files, including any that do not contain any actual drawing commands.

11.5.4 Running the Applications

The main difference between producing X-window output and PostScript output is that there is no cursor available when using PostScript. This means, for instance, that you cannot use PICDEF in cursor mode, and you cannot use CURSOR at all! We made use of both these facilities when we produced Figure 8. Pictures must be defined using PICDEF in `XY` mode, or one of the other positional mode (`CC`, `BL`, *etc.*). Annotation such as produced by CURSOR can be created using LISTMAKE and LISTSHOW. You specify the reference positions for the annotation in any suitable co-ordinate Frame using LISTMAKE. This creates a *positions list* file containing these positions, together with associated WCS information. You then supply this file to LISTSHOW which marks the positions on the graphics device in any of the ways provided by CURSOR.

11.5.5 Using X-windows to Produce a Prototype

It can often be advantageous to design your final plot using an X-windows graphics device. This provides more versatility in the form of a graphics cursor and dynamic lookup table, and also allows you to see the plot more easily. Once the plot looks right in the X-window, you can then run the drawing commands again specifying a suitable PostScript device instead.

If you choose to do this, it can be a big help to ensure that the X-window you are using has the same shape as your selected PostScript device. This prevents you accidentally drawing things within regions of the X-window that are not available in PostScript. To find the require aspect ratio, do:

```
% gdclear device=apscol_l
% gdstate device=apscol_l
```

This clears the PostScript device, and then displays the bounds of the BASE picture (all the other pictures were erased by GDCLEAR). The results will look something like this:

```
Status of the apscol_l graphics device...

   The current picture is a BASE picture.
   Comment: Base picture
   Current co-ordinate Frame: BASEPIC

   Picture bounds in the BASEPIC Frame:
      Axis 1 (X) : 0.000 to 1.455
      Axis 2 (Y) : 0.000 to 1.000
```

This tells you that the aspect ratio (the ratio of the width to the height) of the PostScript device is 1.455. You should now make an X-window with this aspect ratio:

```
% xdestroy xwindows
% xmake xwindows -height 500 -width 730
```

The `xdestroy` command deletes any existing X-window, and the `xmake` command creates a new one with a height of 500 pixels and a width of 730 pixels, giving the required aspect ratio. Of course, you could produce a larger or smaller X-window so long as the ratio of width to height is close to 1.455.

Another tip to ease prototyping on an X-window—when you run CURSOR, always store the selected positions in an output positions list, using Parameter OUTCAT. When you come to produce the PostScript files, you can then supply these positions lists as inputs to LISTSHOW, in order to mimic the annotation produced by CURSOR when you were prototyping.

11.5.6 An Example

As an illustrated example, this section describes how to produce a PostScript file containing a plot similar to that in Figure 8. For clarity (at the time), the earlier sections of this chapter did not adhere to either of the tips given above when prototyping in an X-window; that is, we did not choose the shape of the X-window to match our PostScript device, and we did not save the output from each run of CURSOR in a positions list file. For this reason, the steps taken here are a little more complicated than they need to be, and do not mimic exactly the steps taken while prototyping.

First, select and clear the graphics device. We choose a accumulating, monochrome, portrait, encapsulated PostScript device:

```
% gdset aps_p
% gdclear
% okular pgplot.ps &
```

Since we are using an accumulating PostScript device, we have chosen to use OKULAR above to display the contents of `pgplot.ps`. We have just run GDCLEAR, and so this will produce a blank display. Note that we have put OKULAR into the background by appending `&` to the end of the command line. This means that OKULAR will continue to run throughtout the following example. As each subsequent graphics application modifies the contents of `pgplot.ps`, the resulting composite plot will be displayed immediately within the OKULAR window, allowing you to review progress. Other modern document viewers such as EVINCE work in the same way.

In order to produce higher quality axis annotations, we now tell PGPLOT to use embedded PostScript fonts rather than it's own internal lower-quality fonts. Specifically, we use a Times New Roman font:

```
% setenv PGPLOT_PS_FONT Times
```

We could instead have set `PGPLOT_PS_FONT` to "Helvetica", "Courier", "NewCentury" or "Zapf", to use other fonts.

We will sometimes need to specify positions within the BASE picture. To do this we can either use the GRAPHICS, the BASEPIC or the NDC Frame, all of which span the entire graphics device. GRAPHICS is an absolute co-ordinate system giving millimetres from the bottom left corner of the graphics device, and BASEPIC is a normalised co-ordinate system in which both axes have the same scale and the shorter dimension of the graphics device has length 1.0[19]. The

[19]NDC is like BASEPIC but the top-right corner of the device has NDC co-ordinates (1, 1).

BASEPIC Frame is usually easier to work with since it does not depend on the size of the paper. To determine the bounds of the available plotting space in the BASEPIC Frame, use GDSTATE, having first ensured that the BASE picture is the current picture (this will be the case at the moment since we have just cleared the database using GDCLEAR):

```
% gdstate

Status of the epsf_p graphics device...

   The current picture is a BASE picture.
   Comment: Base picture
   Current co-ordinate Frame: BASEPIC

   Picture bounds in the BASEPIC Frame:
      Axis 1 (X) : 0.000 to 1.000
      Axis 2 (Y) : 0.000 to 1.455
```

The choice of a portrait PostScript device may have surprised you, given that Figure 8 seems more naturally to fit into a landscape page. Portrait mode was chosen so that the final plot appears up-right when included in a document. If landscape mode had been chosen, the final plot would have been rotated by 90° when included in a latex document such as this one, requiring the page to be viewed on its side. However, if we use the entire portrait page, we will have too much vertical space between the components of the plot. To avoid this we only use the bottom third (roughly) of the page—that is, we pretend that the top of our 'page' is at 0.6 instead of 1.454659.

We first create a FRAME picture at the left hand side of the page covering the full height of our reduced 'page' and 0.6 of the width. We specify the upper and lower bounds of the picture within the BASEPIC Frame, using the limits found above, giving the FRAME picture the label 'main' so that we can easily re-select the picture later. We then display the optical image within it, including annotated axes:

```
% picdef mode=xy lbound=\[0.0,0.0\] ubound=\[0.6,0.6\] outline=no
% piclabel main
% display m31 axes mode=perc percentiles=\[30,99\] style="'colour=black,size=0.6'" \
          margin=\[0.2,0.02\]
```

Note, we are using a monochrome PostScript device, and so the colour of all annotation produced by DISPLAY was set to black using the STYLE parameter. This is done for all applications that produce graphical output. We also request text 0.6 of the default size. We specified the MARGIN parameter when running DISPLAY in order to reduce the margin on the right hand side of the image. By default, DISPLAY leaves a margin equal to 0.2 of the width of the DATA picture. We reduced this to 0.02 to make more room for the other components of the plot (the first value supplied for MARGIN is the bottom margin and is left at 0.2).

We now draw a box over this image to mark the 'selected region' indicated in Figure 3. In this particular case, we use LISTMAKE to create a positions list holding the RA and DEC at the two opposite corners of the box. Alternatively, we could have saved the output from CURSOR while prototyping, using the OUTCAT parameter. In either case, we use LISTSHOW to draw the box:

```
% listmake ndf=m31 outcat=boxpos
  POSITION - A position for the output list /!/ > 0:41:33.5 40:33:01
  POSITION - A position for the output list /!/ > 0:39:25.0 40:54:17
  POSITION - A position for the output list /!/ >
% listshow boxpos plot=box style='colour=black'
```

Note, the PostScript output generated by LISTSHOW is merged automatically into the `pgplot.ps` file created by the previous graphics applications. If we had chosen not to use an accumulating device, we would need to rename the new `pgplot.ps` file created by each graphics application, and then merge them all together at the end using PSMERGE.

Specifying `ndf=m31` resulted in LISTMAKE interpreting the supplied positions as positions within the current co-ordinate Frame of the NDF `m31`. It also results in all the WCS information being copied from the NDF into the output positions list.

We now create a Frame in the top-right corner of the reduced 'page' in which a magnified image of the selected region will be displayed. The picture occupies the remainder of the width left over by the main image (0.4 of the total width) and half the height.:

```
% picdef mode=xy lbound=\[0.6,0.3\] ubound=\[1.0,0.6\] outline=no
```

We now display the selected image section within the FRAME picture just created, and overlay contours from `iras.sdf`. We draw the border with twice the default width:

```
% display 'm31(173:213,79:114)' border noaxes mode=scale low=3889 high=10854.78 \
          borstyle='width=2'
% picdata
% piclabel mag
% contour noclear noaxes nokey iras mode=perc percentiles=\[55,75,95\]
```

What are those two extra commands in the middle? Well, we will need to be able to re-select the DATA picture holding the image when we come to draw the warp-lines. To do this we need to label the picture now. For this reason, we use PICDATA to select the DATA picture created by DISPLAY as the current picture, and then use PICLABEL to give the label `"mag"` to the picture.

We now display a title above this DATA picture. The string is centred horizontally within the enclosing FRAME picture (*i.e.* half way between 0.6 and 1.0), and drawn so that its bottom edge is placed at the top of the Frame:

```
% listmake frame=basepic outcat=ttlpos
  POSITION - A position for the output list /!/ > 0.8 0.6
  POSITION - A position for the output list /!/ >
% listshow ttlpos plot=text strings='"IRAS 12 \gmm contours"' just=bc
```

LISTMAKE is used to store the position in a positions list, and then LISTSHOW is used to draw the title. Again, we could have saved the output from CURSOR in a positions list while prototyping, instead of using LISTMAKE to create the positions list. The JUST parameter is set to `bc` so that the title string is displayed with its bottom centre at the supplied position.

We now draw the line marking the path of the profiles that will be displayed later. We draw them in white so that they show up against the predominantly dark background of the image:

```
% listmake ndf=m31 outcat=prfpos
  POSITION - A position for the output list /!/ > 0:41:22.7 40:35:12
  POSITION - A position for the output list /!/ > 0:39:56.5 40:50:27
  POSITION - A position for the output list /!/ >
% listshow prfpos plot=chain marker=3 style="'colour=white,width=3'"
```

The line is drawn with three times the default width. We now create the profiles as before:

```
% profile m31 incat=prfpos out=m31_prof
% profile iras incat=prfpos out=iras_prof
```

We now create a FRAME picture occupying the remaining area at the bottom right corner of the page, and display the two profiles in it. We use line style to differentiate the two curves, instead of colour as we did when prototyping:

```
% picdef mode=xy lbound=\[0.6,0.0\] ubound=\[1.0,0.3\] outline=no
% linplot m31_prof style=^style1 keystyle='size=0.7'
% linplot iras_prof noclear noalign style=^style2
```

The appearances of the two plots are controlled by the two plotting styles contained in the text files `style1` and `style2`. `style1` contains the following attribute settings:

```
tickall=0
colour=black
gap(2)=1000
drawtitle=0
label(2)=DSS data value (solid)
textlabgap(1)=0.02
labelunits=0
```

The file `style2` contains the following attribute settings:

```
edge(2)=r
gap(2)=0.2
tickall=0
colour=black
style(curve)=2
drawtitle=0
label(2)=IRAS data value (dashed)
textlabgap(1)=0.02
labelunits=0
```

Now comes the complicated bit—the warp lines! The problem is that we used the cursor to define the start and end of these lines when prototyping. This was good enough for an X-window that has relatively low resolution, but will not do for PostScript. We cannot place the cursor accurately enough to ensure that there is no visible gap between the end of the line and the corresponding box corner when using PostScript, so we cannot use the positions found while prototyping. The only way to ensure that the lines join up properly with the box corners is to use the co-ordinates of the box corners to define the lines. First, we need to transform the

box corners into the BASEPIC Frame since both ends of each line must be given within the same co-ordinate Frame.

We start with the 'selection box' drawn over the main image. The corners of this box are currently stored in the `boxpos` positions list. This file contains the RA and DEC at the corners of the box, together with WCS information copied from the NDF `m31`. But this does *not* include the BASEPIC, NDC or CURPIC Frame, which are only available within the WCS information stored with graphics database pictures. In order to find the BASEPIC co-ordinates at the corners of the box, we need to align the RA/DEC values in the positions list with the WCS information stored with the first DATA picture we created (the main image), and then display the corresponding BASEPIC co-ordinates. LISTSHOW can do this, but we need to change the current picture first. At the moment, the current picture is the FRAME containing the line plots. If we did not change the current picture LISTSHOW would assume that the supplied RA/DEC values refer to the line plots (not to the main image), and would consequently give the wrong BASEPIC co-ordinates. We use PICSEL to select the FRAME picture containing the main image (so that it becomes the current picture), and then run LISTSHOW:

```
% picsel main
% listshow boxpos plot=blank frame=BASEPIC

  Title: M31 (Digitised Sky Survey)

   Position        X           Y
  identifier
  ----------------------------------

    #1         0.3657834   0.1911708
    #2         0.4269177   0.245274
```

The parameter assignment `PLOT=BLANK` tells LISTSHOW to search the graphics database for a picture with which the positions in `boxpos` can be aligned, but without marking the positions in any way on the screen. Doing this means that the co-ordinate Frames stored with the picture are available when specifying a value for Parameter FRAME. This is necessary because the BASEPIC Frame is only available within graphics data base pictures since it describes positions on a graphics device.

We now find the BASEPIC co-ordinates at the corners of the magnified image. The most simple way of doing this is to use the same RA/DEC positions. The problem with this approach is that the bounds of the magnified image were rounded to a whole number of pixels and so may not correspond exactly to the RA an DEC at the corners of the selection box. A more accurate method is to use the pixel indices at the corners of the image section to define the box corners. We first need to create a positions list holding the pixel co-ordinate bounds of the image section, remembering to reduce the lower pixel index bounds by 1.0 in order to convert the integer *pixel indices* into floatingpoint *pixel co-ordinates* (see Section 12.1 for more information about pixel co-ordinates and indices):

```
% listmake frame=pixel dim=2 outcat=magpos
  POSITION - A position for the output list /!/ > 172.0 78.0
  POSITION - A position for the output list /!/ > 213.0 114.0
  POSITION - A position for the output list /!/ >
```

We now need to use LISTSHOW to get the corresponding BASEPIC co-ordinates, aligning the above PIXEL co-ordinates with the DATA picture holding the magnified image section. Another slight complication arises here since the required DATA picture is obscured by the DATA picture holding the IRAS contours. Pixel co-ordinates in the IRAS image are totally different to those in the grey-scale image, and so we need to make sure that LISTSHOW is using the correct DATA picture. However, we labelled the required DATA picture when it was created and so we can just use PICSEL to re-select it:

```
% picsel mag
% listshow magpos plot=blank frame=BASEPIC
  Alignment has occurred within the PIXEL Domain.


  Title: Output from LISTMAKE

   Position        X           Y
  identifier
  -----------------------------------

    #1         0.6473798   0.3159827
    #2         0.9524395   0.5838399
```

We now create a positions list containing the BASEPIC co-ordinates at the ends of the upper warp line, and draw it. By default, LISTSHOW draws within the most recent DATA picture contained within the current picture. Since the line is not contained within a single DATA picture, we need to tell LISTSHOW to draw within the BASE picture. This is done by setting Parameter NAME to BASE (we also revert to drawing in black):

```
% listmake frame=basepic outcat=war1pos
  POSITION - A position for the output list /!/ > 0.3657834 0.245274
  POSITION - A position for the output list /!/ > 0.6473798 0.5838399
  POSITION - A position for the output list /!/ >
% listshow war1pos plot=poly style='colour=black' name=base
```

The last drawing we need to do is to create the second warp line in the same way:

```
% listmake frame=basepic outcat=war2pos
  POSITION - A position for the output list /!/ > 0.4269177 0.1911708
  POSITION - A position for the output list /!/ > 0.9524395 0.3159827
  POSITION - A position for the output list /!/ >
% listshow war2pos plot=poly name=base
```

That's it. The final output is left in file `pgplot.ps` and should be visible in the OKULAR window. It should look like Figure 9. However, if we had chosen not to use an accumulating PostScript device, we would now be left with lots of PostScript files—one from each graphics application—that now need to be stacked together to produce the final encapsulated PostScript file:

```
% psmerge -e display1.ps box.ps display2.ps contour.ps title.ps \
            line.ps m31_prof.ps iras_prof.ps war1.ps war2.ps > total.eps
```

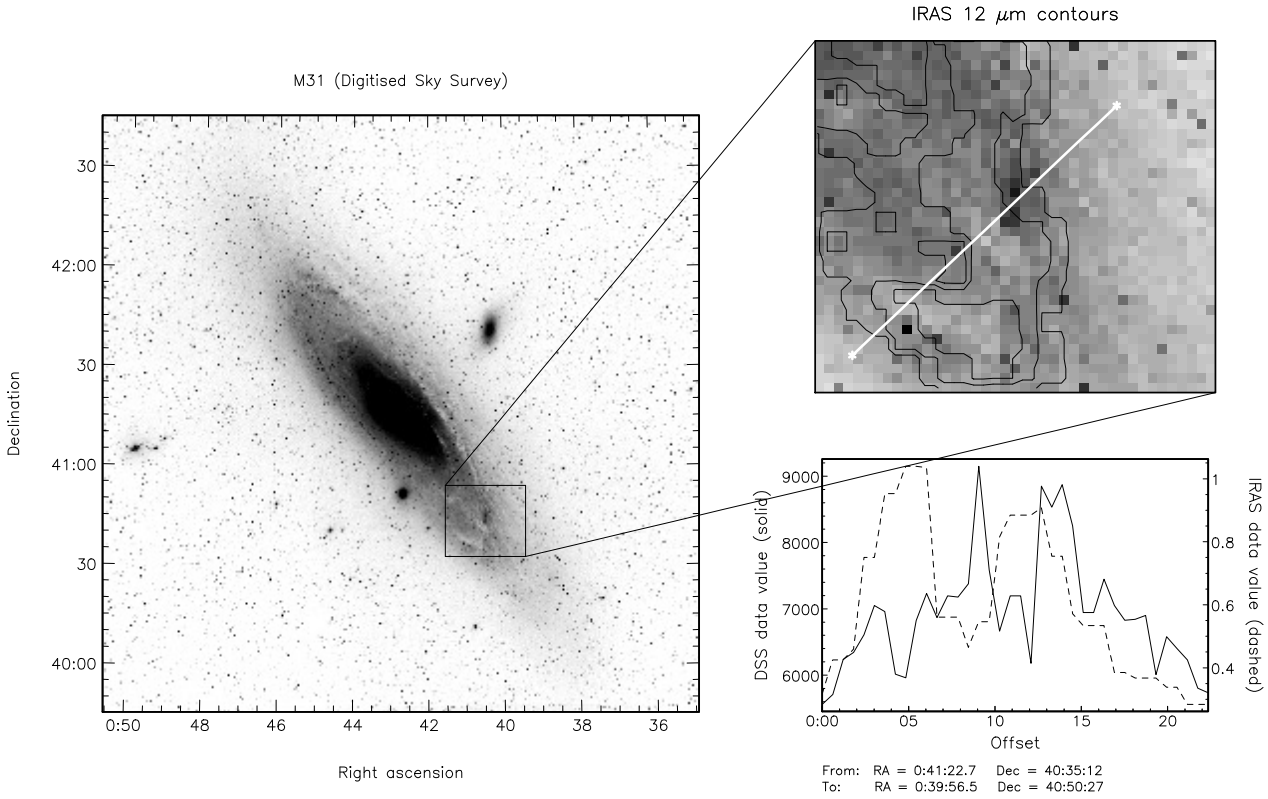


Figure 9: The equivalent plot produced directly in PostScript.

Note, here we assume that each `pgplot.ps` file has been renamed to one of the above names immediately after it has been created, in order to avoid it being over-written by the next graphical application.

12 Using World Co-ordinate Systems

12.1 Pixel Indices, Pixel Co-ordinates, and Grid Co-ordinates

In this sub-section we will look at the definition of the four basic co-ordinate systems available in all NDFs—pixel indices, pixel co-ordinates, grid co-ordinates, and normalised co-ordinates.

Pixel indices are integer values that are used to count the pixels along each axis of an NDF. The first pixel can be given any arbitrary pixel index, and this value is known as the *pixel origin*. When a section is extracted from an NDF, the pixel origin in the extracted section is set so that each pixel retains its original pixel indices (see Figure 10).

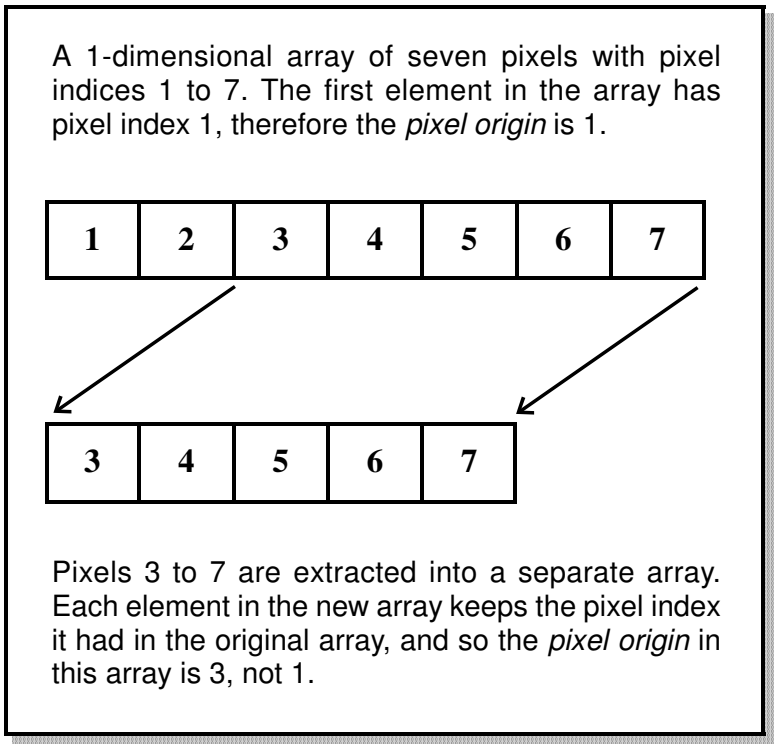


Figure 10: Pixel indices.

Pixel co-ordinates are floating-point values that allow positions to be specified with sub-pixel accuracy. They are related to pixel indices as indicated in Figure 11).

Grid co-ordinates are floating-point values that are similar to pixel co-ordinates except that the origin is fixed so that the first pixel on an axis is centred at a grid co-ordinate value of 1.0, no matter what the pixel origin is. This corresponds to the FITS idea of 'pixel co-ordinates' (FITS makes no provision for an arbitrary pixel origin). When a section is extracted from an array, the grid co-ordinates of the extracted section include no knowledge of where the section was located in the original array. See Figure 12).

Fraction co-ordinates are floating-point values that are normalised pixel or grid co-ordinates such that each axis extends from zero to one. Thus in Figure 13 pixel co-ordinate 2.0 or grid

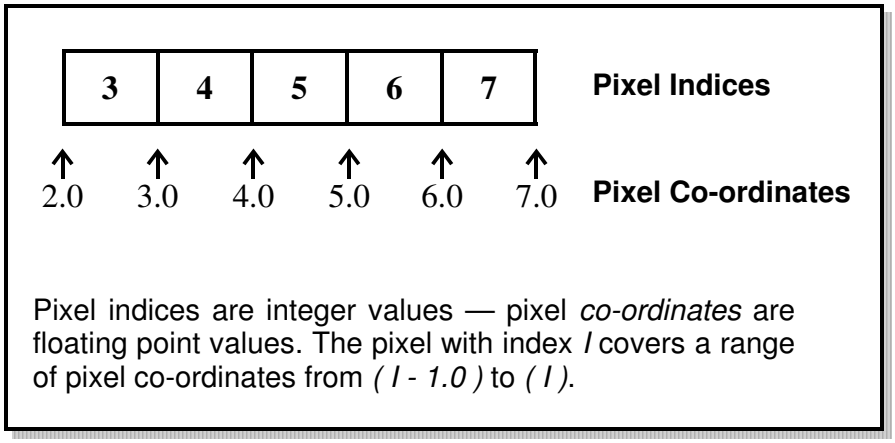


Figure 11: Pixel co-ordinates.

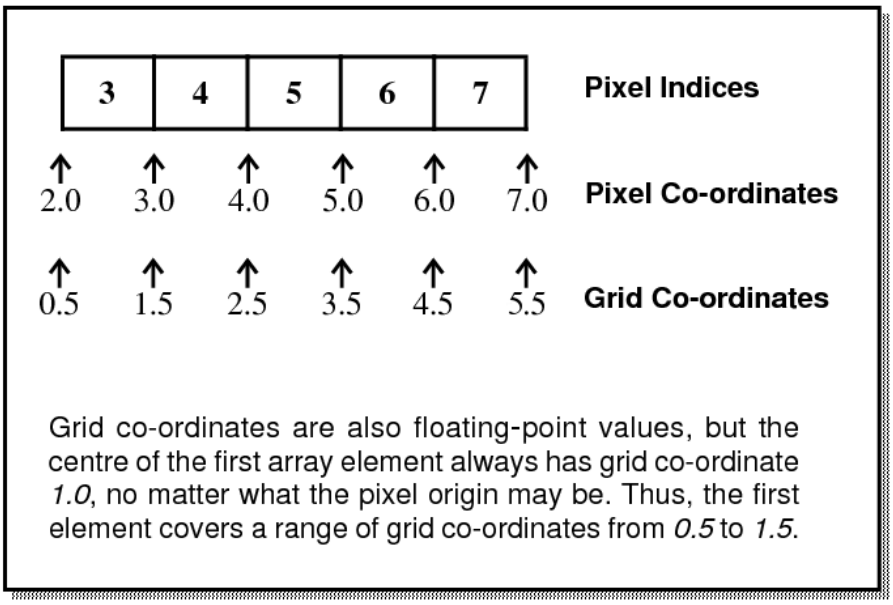


Figure 12: Grid co-ordinates.

co-ordinate 0.5 becomes 0.0 in fraction co-ordinates, and pixel co-ordinate 7.0 or grid co-ordinate 5.5 becomes 1.0 in fraction co-ordinates.

12.2 Co-ordinate Frames, Axes and Domains

A *co-ordinate Frame* is a system of co-ordinate axes which can be used to specify a position within an NDF data array. Within an NDF such co-ordinate Frames also have associated descriptive information such as axis labels, axis units, a Frame title, *etc.* These are called the *attributes* of the Frame, and the most commonly used are listed briefly below (full descriptions of these attributes are included in Appendix D.

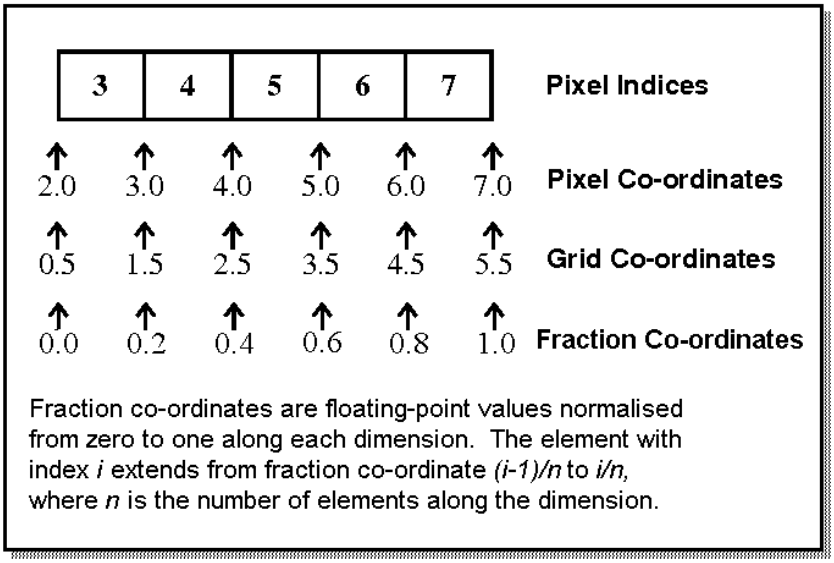


Figure 13: Fraction co-ordinates.

Digits : Number of digits of precision

Domain : Physical domain described by the co-ordinate system

Epoch : A date & time that defines the co-ordinate system

Format(axis) : Format specification for axis values

Label(axis) : Axis label

Naxes : Number of Frame axes

Symbol(axis) : Axis symbol

System : Specific co-ordinate system used to describe the domain

Title : Frame title

Unit(axis) : Axis physical units

The single most important attribute is the Frame Domain, which is an uppercase character string indicating the physical domain in which the co-ordinate system is defined. Frames with the same Domain can, in general, be aligned with each other.

Frames with the following Domain values are defined within every NDF, no matter how the NDF is created.

GRID — Corresponds to grid co-ordinates.

PIXEL — Corresponds to pixel co-ordinates.

FRACTION — Corresponds to normalised pixel or grid co-ordinates where each scaled pixel axis spans the range zero to one.

AXIS — Corresponds to the co-ordinate system defined by the AXIS structures within the NDF.

An AXIS structure is a one-dimensional array that maps pixel index along a given axis on to another related co-ordinate axis (*e.g.* wavelength) – the values on the other axes are assumed to be constant. Each dimension in the NDF should have an associated AXIS structure. Such structures can be created (for instance) using SETAXIS. If the NDF does *not* have defined AXIS structures, then a default AXIS Frame will be used in which the values along each axis correspond to pixel co-ordinates.

AXIS structures can only be used to represent independent axes. For instance, celestial co-ordinates cannot (in general) be described using AXIS structures because celestial longitude and latitude may both vary along any given row or column of an NDF.[20] If an application is used that renders the AXIS structures in an NDF invalid, then the AXIS structures will not be copied to the output NDF. For instance, if ROTATE is used to rotate an NDF by an angle that is not a multiple of 90°, then the output NDF will not have any AXIS structures, and so the AXIS Frame will default to pixel co-ordinates.

The dimensionality of each of these Frames is equal to that of the NDF (*e.g.* a two-dimensional NDF will have two-dimensional PIXEL, FRACTION, GRID, and AXIS Frames). Axes within a Frame are identified by an integer index in the range 1 to the number of axes in the Frame. The above Frames are not stored permanently in the NDF, but are generated automatically by the NDF access library each time a reference is made to them.

In addition to the PIXEL, FRACTION, AXIS, and GRID Frames, an NDF may also contain any number of additional co-ordinates Frames. Descriptions of these extra Frames, together with recipes for converting positions between them, are stored permanently in the NDFs WCS component which may be examined using the NDFTRACE command. Application WCSADD allows new Frames to be added to the WCS component (positions in the new Frame must be related to the corresponding positions in an existing Frame by one of several different supported types of Mapping). WCSREMOVE allows Frames to be removed from the WCS component.

Two common additional Frame Domain options are SKY and SPECTRUM. SKY is reserved to refer to two-dimensional Frames that describe celestial longitude and latitude (known as 'SkyFrames'). SPECTRUM is reserved to refer to one-dimensional Frames that describe position within a spectrum (known as 'SpecFrames'). These sub-classes of Frame have additional attributes indicating the particular celestial or spectral co-ordinate system in use (*e.g.* equatorial, galactic, ecliptic, wavelength, frequency, radio velocity, optical velocity, *etc.*), and any required qualifying parameters (equinox, rest frequency, standard of rest, *etc.*). Instances of these Frames can be added to an NDF by importing suitable FITS headers using FITSDIN, FITSIN or the CONVERT package (see SUN/55). Alternatively, a SkyFrame can be created directly by specifying the pixel co-ordinates of stars with known celestial co-ordinates (see SETSKY and GAIA).

[20] For this and other reasons, new applications will avoid using AXIS structures, and make use of the more versatile WCS component instead.

12.3 FrameSets, and the Current Frame

Any co-ordinate Frames described in the WCS component, together with the standard GRID, FRACTION, PIXEL, and AXIS Frames, form a collection of inter-related Frames called a *FrameSet*. A FrameSet contains descriptions of each Frame, plus recipes (called *Mappings*) describing how to convert positions from one Frame to another.[21]

Frames within a FrameSet are distinguished by their attribute values, primarily their Domain value. In addition, each Frame has an integer *Frame index*. These indices are (in general) allocated sequentially in chronological order as Frames are added into the FrameSet. Applications that require the user to specify a Frame will usually allow the Frame to be specified either by Domain name, or by index. Frame indices can be examined using NDFTRACE (see below).

One Frame within the FrameSet of an NDF is nominated as the *current co-ordinate Frame*. Whenever an application reports positions within an NDF (for instance, when annotating plot axes), the positions reported will refer to the current co-ordinate Frame. Likewise, when-ever an application requests a position from the user (for instance, the position to be placed at the centred of a displayed image), it will expect the position to be given within the current co-ordinate Frame of the NDF,

The contents of the WCS component of an an NDF can be examined using NDFTRACE. By default, this just shows the number of co-ordinates Frames defined within the NDF (including the four standard ones), and the Title and Domain of the current co-ordinate Frame. More detail can be obtained using the keywords FULLWCS and FULLFRAME. The first causes the Title, index and Domain of all Frames to be displayed, together with the index of the current Frame. The second causes a more complete description of each Frame to be displayed, including axis labels, units, celestial co-ordinate system (for SkyFrames), *etc.*

The current co-ordinate Frame can also be examined using WCSFRAME. This is an important application because it also allows you to change the current co-ordinate Frame in the NDF. For instance:

```
% wcsframe m31 pixel
```

will make the PIXEL Frame the current co-ordinate Frame in the NDF `m31`. After this, all references to positions within the NDF `m31` will refer to pixel co-ordinates. You can also change the current co-ordinate system using WCSATTRIB:

```
% wcsattrib myspectrum set system Frequency
% wcsattrib myspectrum set unit GHz
% wcsattrib myspectrum set sor Helio
```

The above changes the values of the 'System', 'Unit' and 'SOR' attributes of the current Frame in the NDF `myspectrum` so that it represents heliocentric frequency in units of `GHz` (this assumes that the current Frame in `myspectrum` is a SpecFrame since only a SpecFrame supports these particular attribute values).

[21] A Mapping need not necessarily be defined in both directions. For instance, a Mapping that goes from a three-dimensional Frame to a two-dimensional Frame (maybe by simply throwing away one of the axis values) may not be defined in the other direction. KAPPA applications will report an error if a Mapping is not defined in the required direction. Another common example of a uni-directional Mapping is that between the GRID and AXIS Frames in cases where the AXIS structures are non-monotonic.

12.4 Reserved Domain Names

New co-ordinates Frames can be created and added into an NDF using WCSADD. When you create a new Frame you should give it a meaningful Domain name which indicates the sort of co-ordinates that it represents. You are free to choose any name you like (white space is removed, and lower case characters are converted to upper case before using the supplied Domain string), but you should usually avoid the following reserved Domain names:

GRID — Reserved to represent grid co-ordinates (in units of pixels).

PIXEL — Reserved to represent pixel co-ordinates (in units of pixels).

AXIS — Reserved to represent AXIS co-ordinates (in arbitrary units).

FRACTION — Reserved to represent normalised pixel/grid co-ordinates from zero to one (unitless).

SKY — Reserved to represent celestial longitude and latitude (in any suitable system, but always stored internally in units of radians).

SPECTRUM — Reserved to represent position within an electro-magnetic spectrum (various spectral systems and units are supported).

TIME — Reserved to represent moments in time (various time-scales and units are supported).

DSBSPECTRUM — Reserved to represent position within an dual-sideband electro-magnetic spectrum.

GRAPHICS — Reserved to represent positions on a graphics device in units of millimetres, measured from the bottom-left corner of the device.

BASEPIC — Reserved to represent positions on a graphics device in normalised units, in which the shorter axis of the graphics device has length 1.0.

NDC — Reserved to represent positions on a graphics device in normalised units, in which the both axes of the graphics device have length 1.0.

CURPIC — Reserved to represent positions in normalised units within each individual graphics database picture. The shorter axis of each picture has length 1.0 in this Frame.

When two Frames are joined together to form a compound Frame describing a co-ordinate space of higher dimensionality, the default Domain name for the compound Frame is formed from the individual Domain names, separated by a hyphen. For instance, the WCS FrameSet associated with a spectral cube will often contain a (compound) Frame with the Domain name "SKY-SPECTRUM". Thus you should also avoid Domain names that contain a hyphen, particularly if they also contain any of the reserved Domain names listed above.

12.5 Specifying a Co-ordinate Frame

Several applications (WCSFRAME, CURSOR, LISTMAKE, *etc.*) have parameters that are used to select a co-ordinate Frame. These parameters are usually called FRAME. When selecting a Frame from an existing FrameSet (*e.g.* read from the WCS component of an NDF), the Frame may be specified in one of the following ways:

- As an integer Frame index within the specified FrameSet starting at 1 for the first Frame.
- As a 'Domain' name (*e.g.* PIXEL, AXIS, SKY), selected from those available in the FrameSet.
- As a *Sky Co-ordinate System* (SCS) specification. This notation has been inherited from the IRAS90 package (see SUN/163, and can be used to indicate a specific celestial co-ordinate system. Simply specifying the Domain name SKY will select any SkyFrame that is present, irrespective of the particular celestial co-ordinate system that the SkyFrame represents. If you want to request a particular celestial co-ordinate system (*e.g.* galactic, equatorial, ecliptic, *etc.*), then use an SCS specification. If the requested system is not present, but a related SkyFrame can be found for a different system, then the SkyFrame will be modified so that it represents the requested system (the Mappings between the SkyFrame and the other Frames in the FrameSet will also be modified appropriately).

  An SCS specification is made up of two parts; a co-ordinate system name, followed by an optional epoch giving the reference equinox. Any of the three system names EQUATORIAL, ECLIPTIC and GALACTIC can be used. Case is insignificant, and abbreviations may be given.

  Ecliptic and equatorial co-ordinates are referred to the mean equinox of a given epoch. This epoch is specified by appending it to the end of the name of the sky co-ordinate system, in parentheses; for instance EQUATORIAL(1983.5) (only the four most significant decimal places are used). The epoch may be preceded by a single character, B or J, indicating if the epoch is a Besselian epoch (B) or a Julian epoch (J). If this character is missing (as in the above example), then the epoch is assumed to be Besselian if it less than 1984.0, and Julian otherwise. If no equinox is specified in this way, then a default of B1950.0 is used.

  If a Julian epoch is used to specify the reference equinox for an equatorial co-ordinate system, then the equatorial co-ordinates are assumed to be in the IAU 1976, FK5, Fricke system. If the equinox is specified using a Besselian epoch, then the co-ordinates are assumed to be in the FK4, Bessel-Newcomb system.

  When a Frame is specified using an SCS specification, it will usually also be necessary to specify the epoch at which the positions were determined. This will be done using the separate Parameter EPOCH. This epoch is required because some celestial co-ordinate systems are non-inertial and rotate slowly with respect to other celestial co-ordinate systems, introducing fictitious proper motions. Knowing the date at which the positions were determined allows the effect of this fictitious proper motion to be eliminated when converting between different systems.

When specifying a new Frame (rather than selecting an existing Frame from a FrameSet), you can either give a Domain name, or an SCS specification, but you cannot give a Frame index. Any string may be used as a Domain name, and you will usually be required to specify the number of axes in the Frame. The exception to this is if you specify one of the Domain names

SKY, GRAPHICS, NDC, CURPIC or BASEPIC, in which case a two-dimensional Frame is always created.

The nature of the current Frame can also be changed using WCSATTRIB, which allows new values to be assigned to specified Frame attributes. For instance, assigning a new value to the System attribute will change the co-ordinate system used to describe positions within the domain covered by the Frame. The main attributes relevant to Frames are described in Appendix D.

12.6 Propagation of WCS Information

KAPPA applications that create an output NDF on the basis of a given input NDF usually copy the contents of the WCS component from the input to the output, modifying it as appropriate to take account of any linear mapping of pixel co-ordinates introducing by the application. The following are exceptions to this rule:

- Applications in which positions in the output NDF are not straight-forwardly related to corresponding positions in the input NDF (*e.g.* ELPROF, CHAIN, RESHAPE).
- Applications in which the output NDF is not a direct representation of the input NDF (*e.g.* FOURIER).

The output NDFs produced by such applications will contain no WCS component (but they will still have the four standard Frames—GRID, FRACTION, PIXEL and AXIS—albeit the AXIS Frame will probably just describe pixel co-ordinates since these applications will not usually copy the AXIS structures either).

The application WCSCOPY can be used to copy the WCS component from one NDF to another, optionally introducing a linear transformation of pixel co-ordinates in the process. This can be used to add WCS information back into an NDF that has been stripped of WCS information by one of the above applications.

12.7 Reading WCS Information Stored in Other Forms

When a KAPPA application requires WCS information, it looks first in the WCS component of the NDF. If no WCS component is defined within the NDF, then it will attempt to obtain WCS information from two other locations, in the following order:

(1) If an IRAS90 astrometry structure (see SUN/163) is present within the NDF, then WCS information will be read from it, and added to the FrameSet holding the GRID, FRACTION, PIXEL, and AXIS Frames. IRAS90 astrometry structures have been used by several applications packages in the past for storing astrometry information. KAPPA contains the SETSKY application which can be used to create such a structure, either by supplying the required numerical parameters (pixel size, image orientation, *etc.*), or by doing a least-squares fit to a set of pixel co-ordinates with corresponding celestial co-ordinates.

(2) If no IRAS90 astrometry structure can be found, an attempt is made to read WCS information from the FITS header cards in the FITS extension.

Note, KAPPA applications always *write* WCS information in the form of a standard WCS component. You should remember that astrometry information stored within an IRAS90 or FITS extension will not be corrected to take account of geometric manipulation produced by applications such as ROTATE, COMPAVE, *etc.* Use of such applications will render IRAS90 and FITS astrometry information incorrect. For this reason, applications always warn the user if astrometry information is being read from an IRAS90 or FITS extension. These extensions can be deleted if necessary, using the ERASE command. For instance:

```
% erase m31.more.iras90
% erase m31.more.fits
```

will erase the IRAS90 and FITS extensions from the NDF `m31`.

12.8 Using SETSKY to Add a Celestial Co-ordinate Frame to an NDF

As mentioned in the previous section, the SETSKY application stores astrometry information within an NDF in the form of either a WCS component or an IRAS90 astrometry structure.

To use SETSKY, you need to know the celestial co-ordinates at a set of points within the image. You may be able to find these by comparing your image with other images, such as those available from the Digitised Sky Survey, which already have astrometry information associated with them. You create a text file holding the pixel and celestial co-ordinates at a single position on each line. For instance, if you have five known RA/DEC (B1950) positions in your image, the file may look like:

```
0 49 05.9,   42 25 30,    32,    266
0 48 31.7,   40 03 36,    39,    29
0 37 03.0,   40 04 48,    258,   31
0 36 54.6,   42 26 47,    257,   268
0 45 47.7,   41 54 03,    93,    213
```

The first column gives the RA values (hours, minutes and seconds), the second gives the DEC values (degrees, arcminutes and arcseconds), the third gives the pixel X co-ordinates, and the fourth gives the pixel Y co-ordinates.

If this file is called `pos.dat`, then the following command can be used to create a WCS component:

```
% setsky m31 ^pos.dat coords='equ(b1950)' epoch=1998.0 projtype=gno

  Trying GNOMONIC projection...

  These parameter values give an RMS positional error of 0.3647723 pixels ...
    Projection type                     : GNOMONIC
    Sky co-ordinates of reference point  : 0h 40m 31.29s, 42d 37m 53.15s
    Image co-ordinates of reference point: (190.3287,285.795)
    Pixel dimensions                    : (36.04451,36.00686) arcsecs
    Position angle of image Y axis      : 359d 38m 35.77s
    Tilt of celestial sphere            : 0d 0m 0.00s
```

Note the up-arrow character ("^") before the file name (pos.dat). This tells SETSKY that the string is a file name. A gnomonic (or *tangent plane*) projection was requested using the PROJTYPE parameter. If no projection type is specified then SETSKY will try four different projections (gnomonic, Aitoff, Lambert equivalent cylindrical, and orthographic) in turn, and choose the one that gives the smallest RMS position error.

An alternative way to add a celestial co-ordinate Frame to an NDF is to use the facilities of GAIA (see SUN/214). This provides much more sophisticated facilities.

12.9 Converting an AXIS structure to a SpecFrame

For many years, the calibration of spectral axes has been recorded in the form of AXIS structures within the NDF. As mentioned above, each pixel axis in an NDF has an associated AXIS structure, which is a one-dimensional array containing an element for each pixel on the associated axis. The value of each element gives the 'axis' value for the pixel.

Spectral axis calibration can now also be recorded in the form of a 'SpecFrame' within the WCS component. A SpecFrame is a Frame that can describe spectral axes in many different forms (wavelength, frequency, various forms of velocity, *etc.*), with many different units, and measured in various rest frames. A SpecFrame 'knows' how to convert between all these different forms. Let's say you have two spectra—in one the current co-ordinate Frame is a SpecFrame representing frequency in GHz as measured in the rest frame of the telescope (*i.e.* 'topocentric frequency'), and in the other the current co-ordinate Frame is a SpecFrame representing radio velocity in km/s measured in the kinematic Local Standard of Rest. You want to overlay plots of these two spectra for comparison purposes, so you display the first using LINPLOT. This produces a plot in which the *x* axis is frequency measured in GHz. You then display the second spectrum, again using LINPLOT but this time specifying clear=no on the command line in order to prevent the previous plot from being erased. The SpecFrame representing radio velocity in the second spectrum automatically adjusts itself to represent the same system as the currently displayed plot (topocentric frequency in this example), so the plots can be compared directly. The conversion includes the effects of the Doppler shift caused by the differing standards of rest used by the two spectra.

So the question arises; "How can I add a SpecFrame to my existing NDFs that only have an AXIS structure?". This is simple to do using the WCSADD command. Let's assume you have a one-dimensional NDF called fred containing an AXIS structure holding the frequency at the centre of each pixel in units of GHz. The following command will add an appropriate SpecFrame to the WCS component of the NDF (and will also make it the *current* Frame):

```
% wcsadd fred frame=axis maptype=unit frmtype=spec domain=SPECTRUM \
        attrs="'System=freq,Unit=GHz'"
```

The FRMTYPE parameter indicates that a SpecFrame should be created. The ATTRS parameter gives the attribute values to be assigned to the SpecFrame (note the quotes to protect the string from interpretation by the UNIX shell). The DOMAIN parameter is given the default domain for a SpecFrame (and should usually not be changed). The MAPTYPE and FRAME parameters indicate that this new SpecFrame should be connected to the existing AXIS Frame using a UnitMap—*i.e.* the frequency values held within the AXIS structure are identical to the frequency values described by the SpecFrame.

The above setting for the ATTRS parameter gives the bare minimum of information—there are several other items of information that *could* have been given. For instance, the above command does not indicate the standard of rest to which the frequency values refer. Neither does it indicate the source position, date of observation, or the observers geographical position (all of which may be needed to enable conversion between different standards of rest). In general you should specify as much information as you can. To do this, you can either include the extra information in the ATTRS parameter value above, or you can add the information later using WCSATTRIB. For example:

```
% wcsattrib fred remap=no set Stdofrest topo
% wcsattrib fred remap=no set RefRA '"10:12:24.2"'
% wcsattrib fred remap=no set RefDec '"-32:10:14"'
% wcsattrib fred remap=no set Epoch '"2003-10-2 12:13:00"'
% wcsattrib fred remap=no set ObsLat '"N19:49:33"'
% wcsattrib fred remap=no set ObsLon '"W155:28:47"'
```

These commands set new values for named attributes within the current Frame of NDF `fred`. These attributes are described in Appendix D. The inclusion of `"remap=no"` is important: it tells the command to change the Frame attribute *without* changing the Mappings between Frames accordingly. If the default value of `"remap=yes"` were used, the Mappings that connect the SpecFrame to the other Frames within the WCS FrameSet would be modified in order to ensure that the position of each pixel is unchanged. The default mode (`"remap=yes"`) should be used if you already have a fully and correctly specified SpecFrame within the WCS component which you want to change to describe a different system. For instance, if you have a SpecFrame describing frequency at each pixel, and you want to change it so that it describes the corresponding wavelength at each pixel, then doing:

```
% wcsattrib fred set System wave
```

will modify the Mapping between the pixel Frame and the SpecFrame using the relationship "wavelength = speed of light/frequency".

However, if you have a partially or incorrectly specified SpecFrame you should usually use `"remap=no"`. For instance, of the above SpecFrame, which gives the frequency at each pixel, was discovered to be incorrect in that the AXIS values were actually wavelength values and not frequency at all, then you would want to *correct* the WCS component by changing the System attribute of the SpecFrame from `"freq"` to `"wave"`. In this situation you want to leave the Mapping from the pixel Frame to the spectral Frame unchanged since the Mapping already gives the correct *wavelength* value (previously, but erroneously, thought to be a frequency value). So here you would do:

```
% wcsattrib fred remap=no set System wave
```

12.10 Specifying Attributes for sub-Frames within Compound Frames

Let's say you have a spectral cube in which the WCS axes are wavelength, RA and Dec. In this case, the current co-ordinate Frame will actually be a 'compound' Frame containing two

'component' Frames; a one-dimensional spectral Frame and a two-dimensional sky Frame. If we consider an attribute such as System (which all classes of Frame have), we now seem to have three different Systems to consider; the System value for the component spectral Frame, the System value for the component sky Frame and the System value for the compound Frame as a whole. So if we want to set the spectral system to `"optical velocity"`, and celestial system to `"Galactic"` what do we do? The answer is to include the index of an axis within the attribute name. If WCS axis 1 is the spectral axis, 2 is the RA axis and 3 is the Dec. axis, then we would do the following:

```
% wcsattrib fred set "System(1)" vopt
% wcsattrib fred set "System(2)" galactic
```

In fact, we could have used `"System(3)"` in place of `"System(2)"` since Axes 2 and 3 are both contained within the same sky Frame and so setting the System attribute for either one will have the same effect. We could now examine the attributes as follows:

```
% wcsattrib fred get "System(1)"
% wcsattrib fred get "System(2)"
```

These commands will display `vopt` and `Galactic`, as expected. Note, the following command:

```
% wcsattrib fred get "System"
```

will display `Compound` since it is displaying the System value of the compound Frame *as a whole* (because no axis index was included).

To continue this example, the SpecFrame class (which represents spectral axes) has an attribute called StdOfRest (standard of rest). This attribute is specific to the SpecFrame class and is not recognised by other classes of Frames. If we do:

```
% wcsattrib fred set "StdOfRest" LSRK
```

we will get an error saying the attribute name is unknown. This is because compound Frames do not have a StdOfRest attribute. If we want to set the standard of rest, we must indicate the index of the spectral axis as follows:

```
% wcsattrib fred set "StdOfRest(1)" LSRK
```

Likewise, if we wanted to set the Equinox attribute of the sky Frame, we could say:

```
% wcsattrib fred set "Equinox(2)" "J2003.5"
```

13 Interaction Mode

We have seen the different co-ordinate systems KAPPA uses. Now we address how the applications obtain co-ordinate information itself. Applications often permit a variety of mechanisms for obtaining those co-ordinates. Typical possibilities are as follows.

Catalogue — In this mode the application reads a file containing a positions list. This can either be a FITS binary table, or a text file in STLformat, and can contain WCS information allowing positions within the catalogue to be aligned with other data. Positions lists can be created by several applications such as CURSOR, LISTMAKE, CENTROID, *etc.*

Cursor — This mode utilises the cursor of the current graphics device. For this to work the array must already be displayed as an image, or a contour plot, or line plot (provided the application handles one-dimensional data), and the picture is stored in the graphics database.

Interface — This mode obtains co-ordinates from the parameter system, usually in response to prompting.

File — In this mode the application reads a text file containing a list of co-ordinates in free format, one object per record. There may be commentary lines in the file beginning with `#` or `!`. The format and syntax of the files are *ad hoc*, and are described in the application documentation.

Applications that permit these options have a parameter, called MODE, by which you can control how positional data are to be acquired. It would be tedious to have to specify a mode for each application, therefore KAPPA has a global parameter—the interaction mode—to which each application's interaction-mode parameter is defaulted. The global value remains in force until you change it by assigning an application's interaction mode on the command line. The following examples shows the effect of the global parameter. For compactness GLOBALS will merely show the interaction mode.

First we display an image on the `xw` windows device.

```
ICL> gdset xw
ICL> display $KAPPA_DIR/ccdframec mode=pe \
Data will be scaled from 2366.001 to 2614.864.
ICL> globals
The current interaction mode is      : <undefined>
```

Now we obtain the centroids of a couple of stellar/galaxian images via each of the interaction modes. First in cursor mode. Note that CENTROID obtains the name of the input NDF from the graphics database in this mode. If you need to preview which NDF is going to be selected use the PICIN command.

```
ICL> centroid mode=c
Current picture has name: DATA, comment: KAPPA_DISPLAY.
Using /star/bin/kappa/ccdframec as the input NDF.
```

```
To select a point press the left button on the mouse or trackerball.
To exit press the right button.
Use the cursor to select one point.

Input guess position was     86.23534, 295.0848
Output centroid position is  86.41057, 295.1141

Use the cursor to select one point.

Input guess position was     73.32529, 318.9757
Output centroid position is  72.76437, 318.9484

Use the cursor to select one point.
```

If we look at the global parameters again, indeed we see that it has become cursor mode.

Now we'll see the effect of changing the mode parameter. Note that unless it is undefined or the application does not support the current mode, you must change the mode on the command line. First we shall prompt for the co-ordinates. A null ends the loop.

```
ICL> centroid mode=i
NDF - Array to be analysed /@/star/bin/kappa/ccdframec/ >
INIT - Guess at co-ordinates of star-like feature /108.8,403.5/ > 86,295

Input guess position was     86, 295
Output centroid position is  86.41057, 295.1141

INIT - Guess at co-ordinates of star-like feature /86,295/ > 73.3,319

Input guess position was     73.3, 319
Output centroid position is  72.76437, 318.9484

INIT - Guess at co-ordinates of star-like feature /73.3,319/ > !
```

Finally, we can create a text file called `starlist.dat` and run CENTROID in file mode.

```
ICL> cat > starlist.dat
86 295
73 320
CTRL/D
ICL> centroid mode=f
COIN - File of initial positions /@centroid.lis/ > starlist.dat
NDF - Array to be analysed /@$KAPPA_DIR/ccdframec/ >

Input guess position was     86, 295
Output centroid position is  86.41057, 295.1141

Input guess position was     73, 320
Output centroid position is  72.76437, 318.9484
```

Such co-ordinate files can also be created interactively with images by CURSOR.

14 Graphics Device Colour Table and Palette

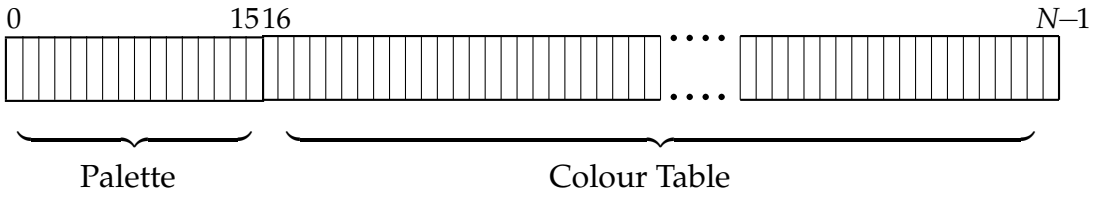
The PGPLOT graphics package, which is used by KAPPA, draws images and line graphics using a set of 'pens'. The number of pens available is limited to 256, even on modern 16- or 24-bit graphics devices which nominally have 'millions' of colours. On older 8-bit graphics devices, the number of available pens may be fewer than 256 if any other applications have 'grabbed' colours for their own use.

Each PGPLOT pen draws in a single colour, but you can choose what that colour will be for each pen. This allocation of colours to pens is called the PGPLOT 'colour table'. Each pen has a corresponding integer *index* within the table. On 8-bit graphics devices you can allocate any arbitrary combination of red, green and blue to a pen (each colour is specified as an 'intensity' in the range zero to one). On 16- and 24-bit devices you can only allocate one of the 'millions' of colours known to the graphics device. For instance, on 16 bit devices it is common to allocate 5 bits each to the red and blue intensity and the remaining 6 bits to the green intensity. This means that red and blue can only be set accurate to 1 part in 32 on these devices, which may result in colours not being exactly as you want them.

On an 8-bit device, any changes thatyou make to the colour table are immediately reflected in the visual appearance of the display. For instance, if you set Pen 1 to red, then draw something using Pen 1, it will appear red on the screen. If you then change Pen 1 to blue, the previously drawn graphics will immediately change colour without you needing to re-draw them. This is *not* usually true for 16- and 24-bit devices. That is, changing pen colours will usually have no effect on previously drawn graphics. In order to see the effects of the changed pen colour, you will need to re-display the graphics.

In many image procesing and visualisation systems the full colour table is used to draw images. This has the disadvantage that if you want to annotate images with captions or axes, plot coloured borders about images, plot graphs *etc*, yet simultaneously display images with certain colour tables, there may be conflict of interests. For instance, a linear grey-scale colour table's first few pens will be almost black. By default, these same pens, particularly Pen 1, are used by the graphics system for line graphics, thus any plots will be invisible. If you reset colour Pen 1 to white, the appearance of your image alters. Whenever you alter the colour table to enhance the look of your image, it will affect the line graphics.

To circumvent this dilemma, KAPPA reserves a portion of the colour table, called the *palette*, that is unaffected by changes to the rest of the colour table. It is shown schematically below. The palette currently contains a fixed 16 pens. n is the total number of pens. In KAPPA the remainder of the pens is called the *colour table*. It is easy to confuse this use of the term 'colour table', with the PGPLOT colour table described above. To sumarize again, in KAPPA the 'colour table' is that part of the PGPLOT colour table that has not been reserved for annotation (*i.e.* the whole colour table minus the first 16 pens which form the annotation *palette*). The context should usually make it obvious which understanding of the phrase 'colour table' is being used.



14.1 Lookup Tables

A list of colours to be allocated to each pen in the colour table is called a *lookup table*. Lookup tables comprise a series of red, green and blue (RGB) intensities, each normalised to 1.0; they may be stored in NDFs—indeed some are provided with KAPPA—or be coded within applications.

A lookup table may be transferred into the display's colour table. However, the number of pens in the colour table is usually not the same as the number of colours in the lookup table and so a simple substitution is not possible. Therefore, KAPPA squeezes or stretches the lookup table to make it fit in the available number of colour-table pens. Normally, linear interpolation between adjacent lookup-table entries defines the resultant colour, though you can select a nearest-neighbour algorithm. The latter is suited to lookup tables with sharp boundaries between contrasting colours, *e.g.* a series of coloured blocks, and the former to smoothly varying lookup tables where there are no obvious discontinuities, *e.g.* spectrum-like.

Let's have a few examples.

```
% lutheat
% lutramps
% lutread pastel
% lutable li ex sawtooth nn
% lutsave pirated
```

LUTHEAT loads the standard 'heat' lookup table into the colour table using linear interpolation, whilst LUTRAMPS loads the standard coloured ramps using the nearest neighbours in the lookup table. LUTREAD reads the lookup table stored in the DATA_ARRAY of the NDF called pastel and maps it on to the colour table via linear interpolation. In the fourth example the lookup table in NDF sawtooth is mapped on to the colour table via a linear nearest-neighbour method. `ex` tells LUTABLE to read an external file. In the final example LUTSAVE saves the current colour table into a lookup-table NDF called pirated. LUTSAVE is quite useful as you can steal other people's attractive colour tables that they've carelessly left in the display's memory! It does not matter should the display not have a palette, since

```
ICL> lutsave pirated full
```

will save the full set of pens (including the first 16) to the NDF.

14.2 Manipulating Colour Tables

LUTEDIT provides a complete graphical-user-interface which allows colour tables to be created or modified in many different ways.

14.3 Creating Lookup Tables

14.3.1 From a Text File

You can make a text file of the RGB intensities and use TRANDAT to create the NDF , or manipulate the colour table and then save it in a lookup-table NDF. If you choose the second option remember that all RGB intensities must lie in the range 0.0–1.0, where 1.0 is the maximum intensity; and that equal red, green, and blue intensities yields a shade of grey. So for example if you want a six equal blocks of red, blue, yellow, pink, sienna and turquoise you could create the text file `col6.dat` with contents

```
# Red, blue, yellow, pink, sienna, and turquoise LUT
1.0 0.0 0.0
0.0 0.0 1.0
1.0 1.0 0.0
0.9 0.56 0.56
0.56 0.42 0.14
0.68 0.92 0.92
```

and then run TRANDAT to make the NDF called collut6.

```
% trandat col6 collut6 shape='[3,6]' auto
```

14.3.2 Running LUTEDIT

There is an interactive task called LUTEDIT for creating and editing lookup tables. The LUTEDIT command fires up a complete graphical-user-interface. This includes its own help system via a "short help" window at the bottom of the interface which describes the control currently under the pointer, and also via the usual "Help" button at the right-hand end of the menu bar.

14.4 Palette

There are four commands for controlling the palette (Section 14), all beginning PAL (in addition, the colours in the current palette can be listed using GDSTATE). If you inherit the graphics device after a non-KAPPA user or after a device reset, you will probably have to reset the palette. You can do this either by loading the default palette—black, white, the primary then secondary colours, and eight equally spaced grey levels—with the command PALDEF; or load a palette you've created yourself via PALREAD. You modify the palette by changing individual colours within it using PALENTRY. The colour specification can be a named colour (see Appendix F for a list), or RGB intensities. For example,

```
% palentry 1 Skyblue
% palentry 14 [1.0,1.0,0.3]
```

would make palette index 1 sky blue and index 14 a pale yellow. Once you have a palette you like, save it in an NDF with PALSAVE.

Palette entry 0 is the background colour. By choosing a palette colour equal to the background colour, features may be 'erased'.

14.5 Persistence of Palettes and Colour Tables

PGPLOT re-initializes the palette and colour table each time it is started up, wiping out the colours you had previously selected with such care! For this reason, KAPPA keeps a copy of the 'current' palette and colour table in a special file. Each time a graphics application is run, the current colour table and palette are read back from this file, and used to reset the pen colours in the PGPLOT colour table before any drawing is performed. Some application change the colour table or palette; PALENTRY, LUTABLE, *etc.*. When such applications terminates, they write the modified colour table or palette back to the file so that it will be used by subsequent graphics applications.

Separate palettes and colour tables are maintained for each known graphics device, so running LUTGREY on an xwindows device will have no effect on the colour table used for PostScript devices (for instance). The palettes for all known devices are stored in a file called `kappa_palette.sdf` located within your ADAM directory (usually `$HOME/adam`). The colour tables for all known devices are stored in a file called `kappa_lut.sdf` located within the same directory.

15 Masking, Bad Values, and Quality

Masking is the process by which you can exclude portions of your data from data processing or analysis. Suppose that you are doing surface photometry of a bright galaxy, part of the data reduction is to measure the background contribution around the galaxy and to subtract it. You usually want to avoid inclusion of light from the galaxy in your estimation of the background. A convenient method for doing this is to mask the galaxy during the background fitting.

There are two techniques used for masking. One employs special *bad* values (also known as *magic* or *invalid* values). These appear within the data or variance arrays in place of the actual values, and indicate that the pixel is to be ignored or is undefined. They are destructive[22] and so some people don't like them, but you can always mask your data into a new, temporary NDF. With a little care, bad values are quite effective and they are used throughout KAPPA. By its nature, a bad value can only indicate a logical, two-state condition about a data element—it is either good or bad—and so this technique is sometimes called *flagging*.

In contrast, the second technique, uses a quality array. This permits many more attributes or qualities of the data to be associated with each pixel. In the current implementation there may be up to 255 integer values, or 8 single-bit logical flags. Thus quality can be regarded as offering 8 logical masks extending over the data or variance arrays, and can signify the presence or absence of a particular property if the bit has value 1 or 0 respectively. An application of quality to satellite data might include the detector used to measure the value, some indicator of the time each pixel was observed, was the observation made within the Earth's radiation belts, and whether or not the pixel contains a reseau mark. By selecting only those data with the appropriate quality values, you process only the data with the desired properties. This can be very powerful. However, it does have the drawback of having to store at least an extra byte per pixel in your NDF.

The two methods are *not* mutually exclusive; the NDF permits their simultaneous use in a dataset.

Now we'll look at both of these techniques in detail and demonstrating the relevant KAPPA tasks.

15.1 Bad-pixel Masking

Bad pixels are flagged with the Starlink standard values (see Section 5 of SUN/39), which for _REAL is the most-negative value possible.

In addition to tasks that routinely create bad values in the output value is undefined, KAPPA offers many applications for flagging pixels with certain properties or locations.

15.1.1 Doing it the ARD Way

To mask a region or a series of regions within an NDF, you can create an ASCII Region Definition (ARD) text file. ARD has a powerful syntax for combining regions and supplying WCS information, described fully in SUN/183. An ARD file comprises keywords that define a region, such as `RECT` to specify a rectangular box; operators that enable regions to be combined, for

[22] That is, the special bad value replaces the original data values, and so the original data values are lost.

instance `.AND.` that will form the intersection of two regions; and statements to define the world co-ordinate system and dimensionality. For further details see the three sections called *Regions*, *Operators*, and *Statements* in SUN/183.

Here is an example of the creation of an ARD file.

```
% cat myard.ard
COFRAME(PIXEL)
PIXEL( 23.5, -17.2 )
ELLIPSE( 75.2, 296.6, 33, 16, 78 )
POLYGON( 109.5, 114.5, 122.5, 131.5, 199.5, 124.5 )
CIRCLE( 10, 10, 40 ) .AND. .NOT. CIRCLE( 10, 10, 30 )
COFRAME(SKY,SYSTEM=FK5,EQUINOX=2000)
CIRCLE( 10:09:12.2, -45:12:13, ::40 )
CTRL/D
```

The COFRAME statements indicate the co-ordinate system in which subsequent positions are supplied. Its first argument is the domain. Here the first `COFRAME(PIXEL)` refers to pixel co-ordinates. Note that these are not the same as pixel indices, as they are displaced by -0.5 with respect to pixel indices. The second `COFRAME` selects a SKY domain using the FK5 system, so that regular equatorial co-ordinates may be supplied as arguments to subsequent keywords. Other possible values for System include `ECLIPTIC` and `GALACTIC`. If no COFRAME or WCS statement is present, the default co-ordinate system is pixel co-ordinates transformed by any COEFFS, OFFSET, TWIST, STRETCH, SCALE statements. Note that the ARDMASK application, used to mask data with an ARD file, has a DEFPIX parameter where you can choose whether the default co-ordinates are pixel or those of the current WCS Frame. if there is no COFRAME or WCS statement in your ARD file. Still you are recommended to supply a COFRAME or WCS statement in your ARD files to avoid accidentally selecting the wrong regions.

In this example, the regions are: the single pixel at co-ordinates (23.5, -17.2); an ellipse centred at (75.2, 296.6) with semi-major axis of 33 pixels and semi-minor axis of 16 pixels, at orientation 78° clockwise from the *x* axis; a triangle with vertices at pixel indices (110, 115), (123, 132), (200, 125); an annulus centred on pixel co-ordinates (10.0, 10.0) between radius 30 and 40 pixels; and a circle centred on RA 10:09:12.2, and DEC -45:12:13 of radius 40 arcseconds.

Operators combine regions using a Fortran-like logical expression, where each keyword acts like a logical operand acted upon by the adjoinning operators. Statements are ignored in such logical expressions. There is an implicit `.OR.` operator for every keyword on a new line. Thus pixels that lie in any of the above regions (the union) are selected.

Where a keyword (such as CIRCLE, RECT, POLYGON) defines an area or volume a pixel is deemed to be part of that region if its *centre* lies on or within the boundary of the region. For regions of zero volume (such as keywords PIXEL, LINE, COLUMN), the pixel is regarded as part of the region when the locus of the region passes through that pixel. So for example, a PIXEL region will be the pixel emcompassing the supplied co-ordinates; and for a LINE, the selected pixels are all that intersect with the line's locus.

Here are some more examples of ARD files.

```
COFRAME(GRID)
ROTBOX( 12, 15, 20, 10, 36.3 ) .AND. .NOT. ( COLUMN( 13 ) .OR. ROW( 8 ) )
```

Now the co-ordinates are Grid co-ordinates. This selects all the pixels with a rotated box except those in thirteenth column or eighth row. Note the use of parentheses to adjust or clarify the precedence. The box is centred on grid pixel (12, 15) has sides of length 20 and 10 pixels. The first side of the box—the one with length 20—is at an angle of 36.3° measured anticlockwise from the X axis.

```
DIMENSION(3)
CIRCLE( 10.3, 21.6, 32.9, 10.4 )
LINE( 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 )
```

This defines a sphere centred at pixel co-ordinates (10.3, 21.6, 32.9) with radius 10.4 pixels, and a line from (1.1, 2.2, 3.3) to (4.4, 5.5, 6.6).

```
CIRCLE( 10.3, 21.6, 32.9, 10.4 )
```

This defines a sphere centred at pixel co-ordinates (10.3, 21.6, 32.9) with radius 10.4 pixels.

```
.NOT. ELLIPSE( 75.2, 296.6, 33, 16, 78 )
```

This selects the whole array except for the ellipse defined as before. Something like this might be useful for excluding a galaxy image before fitting to the background around the galaxy.

There are more details and further ARD facilities described in SUN/183. If you do not wish to read SUN/183, you'll be relieved to learn that there are shortcuts for two-dimensional data...

The first is is provided by GAIA by its `Image Analysis` → `Image Regions...` tool. Here you can select region types and interactively adjust the region locations and shapes, and then record the selected regions in an ARD file. However, it does not provide the boolean operators other than .OR. to combine a series of regions, or use co-ordinates other than pixel.

KAPPA offers its own interactive graphical tool for generating ARD files. To use ARDGEN you must first display your data on a device with a cursor, such as an X-terminal. DISPLAY with a grey-scale lookup table is probably best for doing that. The grey lets you see the coloured overlays clearly. The following example assumes that the current co-ordinate Frame in the NDF is PIXEL (*i.e.* pixel co-ordinates). Consequently all positions are shown below in pixel co-ordinates. If the current co-ordinate Frame in the NDF was not PIXEL but (say) SKY, then ARDGEN would produce positions in SKY co-ordinates. The ARD file generated by ARDGEN always contains a description of the co-ordinate system in which positions are specified, allowing later applications to interpret them correctly, and convert them (if necessary) into other co-ordinate systems.

```
% ardgen demo.ard
Current picture has name: DATA, comment: KAPPA_DISPLAY.
SHAPE - Region shape /'CIRCLE'/ >
```

At this point you can select a shape. Enter `?` to get the list. Once you've selected a shape you'll receive instructions.

```
      SHAPE - Region shape /'COLUMN'/ > ellipse

   Region type is "ELLIPSE".  Identify the centre, then one end of the semi-major
   axis, and finally one other point on the ellipse.

      To select a position press the space bar or left mouse button
      To exit press "." or the right mouse button
```

Once you have defined one ellipse, you can define another or exit to the OPTION prompt. In addition to keyboard 1, pressing the right-hand mouse button has the same effect. Thus in the example, the new shape is a rotated box.

```
Region completed. Identify another 'ELLIPSE' region...
OPTION - Next operation to perform /'SHAPE'/ > shape
SHAPE - Region shape /'ELLIPSE'/ > rotbox


Region type is "ROTBOX". Identify the two end points of any edge and then give
a point on the opposite edge.
Region completed. Identify another 'ROTBOX' region...
```

If you make a mistake, use the 'Undo' option. Alternatively, enter `List` at the OPTION prompt to see a list of the regions. Note the 'Region Index' of the region(s) you wish to remove, and select the `Delete` option. At the REGION prompt, give a list of the regions you want to remove. If you change your mind, enter `!` at the prompt for Parameter REGIONS, and no regions are deleted.

Now suppose you want to combine or invert regions in some way, you supply `Combine` at the OPTION prompt. So suppose we have created the following regions in `$KAPPA_DIR/ccdframe`.

```
Region          Region Description
Index

  1   -  ELLIPSE( 174.1, 234.4, 82.2, -43.5, 65.64783 )
  2   -  ELLIPSE( 168.1, 209.1, 29.4, -19.7, 9.441798 )
  3   -  ELLIPSE( 42.2, 244.1, 13, -10.3, 111.8452 )
  4   -  ROTBOX( 40.5, 219.2, 63.8, 38.3, 37.24281 )
  5   -  RECT( 141.5, 1.4, 143.9, 358.8 )
  6   -  POLYGON( 229.8, 247.7,
                  233.4, 247.7,
                  233.4, 258.6,
                  231, 267,
                  229.8, 265.8,
                  228.6, 256.2 )
```

We want to form the region inside the first ellipse but not inside the second. This done in two stages. First we invert the second ellipse, meaning that pixels are included if they are not inside this ellipse, by combining with the `NOT` operator.

```
OPTION - Next operation to perform /'SHAPE'/ > comb
OPERATOR - How to combine the regions /'AND'/ > not
OPERANDS - Indices of regions to combine or invert /6/ > 2
```

This removes the original Region 2, decrements the region numbers of the other regions following 2 by one, so that Region 3 becomes 2, 4 becomes 3, and so on. A new Region 7 is the inverted ellipse. The renumbering makes it worth listing the regions before combining regions. The second stage is to combine it with Region 1, using the AND operator. This includes pixels if they are in both regions. In this example, that means all the pixels outside the second ellipse but which lie within the first.

```
OPTION - Next operation to perform /'SHAPE'/ > com
OPERATOR - How to combine the regions /'AND'/ >
OPERANDS - Indices of regions to combine or invert /[6,7]/ > 1,6
```

Here is another example of combination. This creates a region for pixels are included provided they are in one of two regions, but not in both. Here we apply the .XOR. operator to the small ellipse and the first rotated box.

```
OPTION - Next operation to perform /'SHAPE'/ > comb
OPERATOR - How to combine the regions /'AND'/ > xor
OPERANDS - Indices of regions to combine or invert /[4,5]/ > 1,2
```

Here is the final set of regions.

```
OPTION - Next operation to perform /'SHAPE'/ > list


  Region          Region Description
  Index

    1   -  RECT( 141.5, 1.4, 143.9, 358.8 )
    2   -  POLYGON( 229.8, 247.7,
                    233.4, 247.7,
                    233.4, 258.6,
                    231, 267,
                    229.8, 265.8,
                    228.6, 256.2 )

    3   -  ( ELLIPSE( 174.1, 234.4, 82.2, -43.5, 65.64783 )
             .AND.
             ( .NOT. ELLIPSE( 168.1, 209.1, 29.4, -19.7, 9.441798 ) ) )

    4   -  ( ELLIPSE( 42.2, 244.1, 13, -10.3, 111.8452 )
             .XOR.
             ROTBOX( 40.5, 219.2, 63.8, 38.3, 37.24281 ) )
```

Once you are done, enter "Exit" at the OPTION prompt, and the ARD file is created. "Quit" also leaves the programme, but the ARD file is not made.

Having created the ARD file it is straightforward to generate a masked image with ARDMASK[23]:

```
% ardmask $KAPPA_DIR/ccdframec demo.ard ardccdmask
```

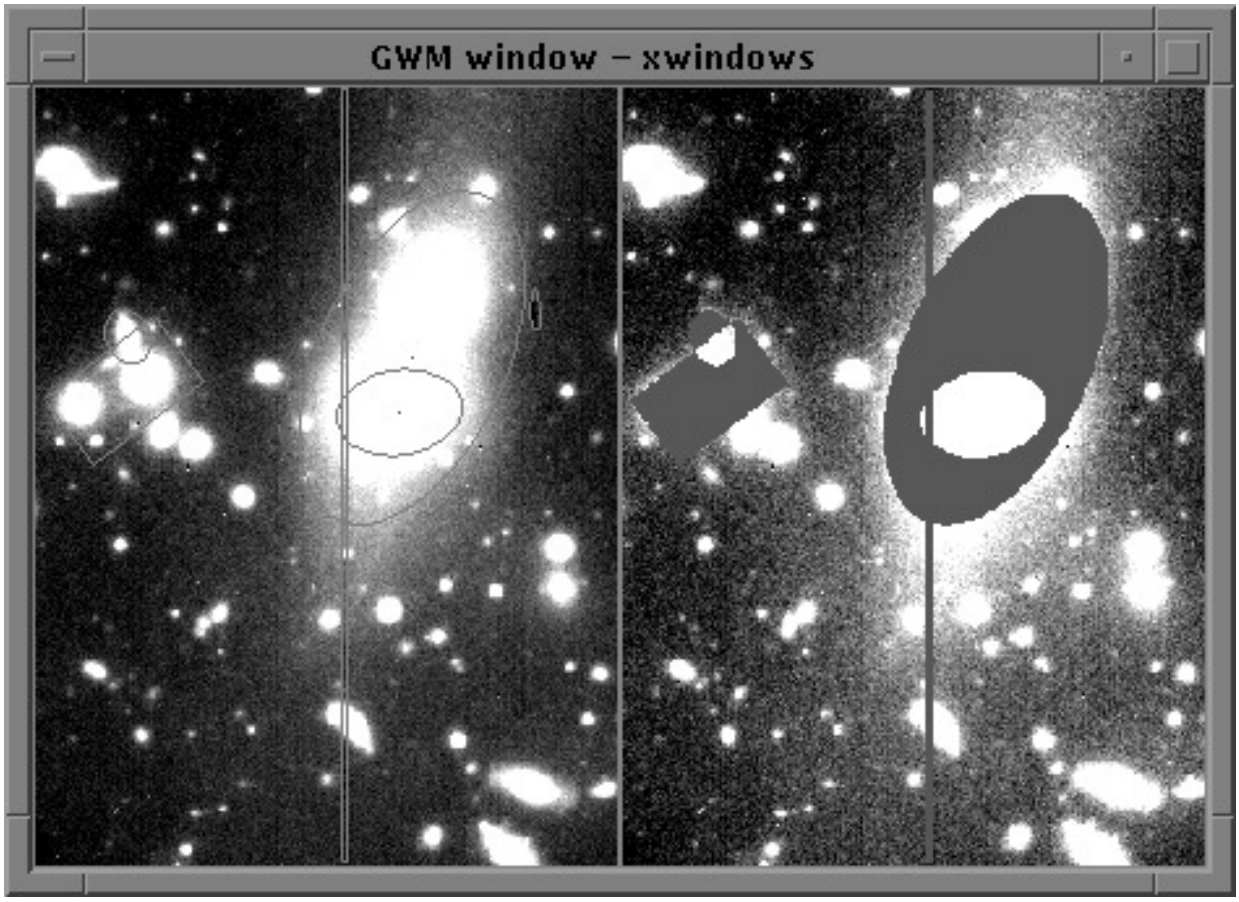


Figure 14: Masking of `$KAPPA_DIR/ccdframec`. To the left shows the original ARDMASK regions, and to the right shows the final masked regions after some have been combined.

Figure 14 shows the image with the original regions outlined to the left. Note only the section (:270, :360) is displayed. To see where you have masked, use DISPLAY, which lets you define a colour for bad pixels using the BADCOL parameter.

```
% display ardccdmask badcol=red \\
```

To the right of Figure 14 is the final masked image.

15.1.2 SEGMENT and ZAPLIN

SEGMENT is ostensibly for copying polygonal regions from one NDF to another. You may also use SEGMENT to copy bad pixels into the polygonal regions by giving the null value for one of the two input NDFs. For instance,

```
% segment in1=! in2=$KAPPA_DIR/ccdframec out=ccdmask
```

[23] You can also plot the outline of the selected regions on top of a display image using ARDPLOT.

NDF ccdmask will have bad values inside the polygons, whereas

```
% segment in2=! in1=$KAPPA_DIR/ccdframec out=ccdmask
```

the pixels exterior to the polygons are flagged. SEGMENT lets you define the polygon vertices interactively, like in ARDGEN, but you can also use text files, or respond to prompting.

ZAPLIN also has an option to fill in rectangular areas when Parameter ZAPTYPE has value `Bad`.

15.1.3 Special Filters for Inserting Bad Values

There are applications that mask pixels if their values meet certain criteria.

SETMAGIC flags those pixels with a nominated value. It is most useful during conversion of imported data whose data system uses bad-pixel values different from Starlink's.

FFCLEAN removes defects smaller than a nominated size from an image or vector NDF . It flags those pixels that deviate from a smoothed version of the NDF by more than some number of standard deviations from the local mean.

ERRCLIP flags pixels that have errors larger than some supplied limit or signal-to-noise ratios below a threshold. The errors come from the VARIANCE component of the NDF. Thus you can exclude unreliable data from analysis.

THRESH flags pixels that have data values within or outside some specified range.

15.2 Quality Masking

All the NDF tasks in KAPPA use quality yet there is no obvious sign in individual applications how particular values of quality are selected. What gives? The meanings attached to the quality bits will inevitably be quite specific for specialist software packages, but KAPPA tasks aim to be general purpose. To circumvent this conflict there is an NDF component called the *bad-bits mask* that forms part of the quality information. Like a QUALITY value, the bad-bits mask is an unsigned byte. Its purpose is to convert the eight quality flags into a single logical value for each pixel, which can then be processed just like a bad pixel.

When data are read from the NDF by mapping into memory, the quality of each pixel is combined with the bad-bits mask; if a result of this quality masking is `FALSE`, that pixel is assigned the bad value for processing. This does not change the original values stored in the NDF; it only affects the mapped data.

So how do the quality and bad-bits mask combine to form a logical value? They form the bit-wise 'AND' and test it for equality for 0. None the wiser? Regard each bit in the bad-bits mask as a switch to activate detection of the corresponding bit in a pixel's quality. The switch is on if it has value `1`, and is off if it has value `0`. Thus if the pixel is flagged only if one or more of the eight bits has both quality and the corresponding bad-bit set to 1. Here are some examples:

QUALITY:	10000001	10000001
Bad-bits:	01000100	01000101
Bits on:		^
Result:	TRUE	FALSE

The application SETBB allows you to modify the bad-bits mask in an NDF. It allows you to specify the bit pattern in a number of ways including decimal and binary as illustrated below.

```
% setbb RO950124 5
% setbb RO950124 b101
```

These both set the bad-bits mask to 00000101 for the NDF RO950124. SETBB also allows you to combine an existing NDF bad-bits mask with another mask using the operators AND and OR. OR lets you switch on additional bits without affecting those already on; AND lets you turn off selected bits leaving the rest unchanged.

```
% setbb RO950124 b00010001 or
% setbb RO950124 b11101110 and
```

The first example sets bits 1 and 5 but leaves the other bits of the mask unaltered, whereas the second switches off the same bits.

Now remembering which bit corresponds to which could be a strain on the memory. It would be better if some meaning was attached to each bit through a name. There are four general tasks that address this. SETQUAL sets quality values and names; SHOWQUAL lists the named qualities; REMQUAL removes named qualities; and QUALTOBAD uses a logical expression containing the named quality properties to create a copy of your NDF in which pixels satisfying the quality expression are set bad. See Section 16 for more information about using these tasks. Once you have defined quality names, you can set the bad-bits mask with SETBB to mask pixels with those named quality attributes.

```
% setbb RO950124 spike
% setbb RO950124 '"spike,back"'
```

The first example might set the bad-bits mask to exclude spike artefacts. The second could mask both spikes and background pixels. Thus it might be used to select the spectral lines not affected by noise spikes in a spectral cube. Other logical combinations are possible using the AND and OR operators.

15.3 Removing bad pixels

Sometimes having bad pixels present in your data is a nuisance, say because some application outside of KAPPA does not recognise them, or you want to integrate the flux of a source. KAPPA offers a number of options for removing bad values. Which of these is appropriate depends on the reason why you want to remove the bad pixels.

First you could replace the bad values with some other reasonable value, such as zero.

```
% nomagic old new 0 comp=all
```

Here dataset new is the same as dataset old except that any bad value in the data or variance array has now become zero.

If you wanted some representative value used based upon neighbouring pixels, use the GLITCH command.

```
% glitch old new mode=bad
```

This replaces the bad values in the data and variance arrays with the median of the eight neighbouring pixels. This works fine for isolated bad pixels but not for large blocks. If your data are generally flat, large areas can be replaced using the FILLBAD task.

```
% fillbad old new size=4
```

The value of Parameter SIZE should be about half the diameter of the largest region of bad pixels. Both the data array and variance arrays are filled.

You may replace individual pixels or rectangular sections using CHPIX.

```
% chpix old new
SECTION - Section to be set to a constant /'55,123'/ >
NEWVAL - New value for the section /'60'/ >
SECTION - Section to be set to a constant /'1:30,-10:24'/ >
NEWVAL - New value for the section /'-1'/ >
SECTION - Section to be set to a constant /'1:30,-10:24'/ > !
```

This replaces pixel (55, 123) with value 60, and the region from (1, −10) to (30, 24) with −1. The final `!` ends the loop of replacements. If you supply NEWVAL on the command line, only one replacement occurs.

It is also possible to paste other datasets where your bad values lie with the PASTE and SEGMENT tasks.

```
% paste old fudge"(10:20,29:30)" out=new
```

The dataset old is a copy of dataset new, except in the 22-pixel region (10, 29) to (20, 30), where the values originate from the fudge dataset.

16 Using Quality Names

16.1 Introduction

As described in Section 15, an NDF may optionally contain a component called QUALITY. If this component exists, it will be an array with the same bounds as the main DATA array. Each element in the QUALITY array can be used to store several flags that are associated with the corresponding element in the DATA array. These flags may be used to indicate that the DATA value holds some specified property. For instance, one of the flags may be used to indicate if the corresponding DATA values are saturated, another may be used to indicate if the DATA value lies within a background area, and so on.

You are free to use the flags in whatever way seems most suited to the particular process being performed. You can set (or reset) any of the flags within any sub-region of the NDF using application SETQUAL. Each of the flags is referred to by a *Quality Name* specified by the user. Names that reflect the nature of the quality should be used, *e.g.* the quality name `SATURATED` could be used to flag saturated data values. These quality names get stored within the NDF and can be used to refer to the quality flag when running later applications. The terminology adopted here is that an element of the DATA array 'holds' the quality `SATURATED` (for instance) if the flag that is associated with the quality name `SATURATED` is set for the corresponding element within the QUALITY array.

The number of quality names that can be stored within an NDF is limited, and therefore it may become necessary to remove quality names that are no longer needed to make room for new ones. The applications SHOWQUAL and REMQUAL allow you to do this. SHOWQUAL displays a list of the quality names currently defined within an NDF, and REMQUAL removes specified quality names from an NDF.

Some applications have a Parameter QEXP, which may be used to specify that the application is only to use data values that hold a specified selection of qualities. As an example, when running QUALTOBAD you could (for instance) specify a value of `BACKGROUND` for the QEXP parameter. This means that only those data values for which the flag associated with the quality name `BACKGROUND` is set, are to be set bad. The quality name `BACKGROUND` must previously have been defined and assigned to the appropriate data values using application SETQUAL.

The specification of the data values to be used by an application can be more complex than this, and can depend on several qualities combined together using 'Boolean' operators. For instance, assigning the value `.NOT. (SOURCE_A .OR. SOURCE_B)` would cause the application to use only those data values that hold neither of the qualities `SOURCE_A` and/or `SOURCE_B`. These sort of strings are known as *quality expressions*.

16.2 Quality Names

Quality names are names by which the user refers to particular flags stored in the QUALITY component of the NDF. They must not be longer than 15 characters. Leading blanks are ignored, and they are always stored in upper case, even if they are supplied by the user in lower case. Embedded blanks are considered to be significant. Quality names must not contain any fullstops (`"."`), and there are three reserved names that cannot be used; these are `ANY`, `IRQ_BAD_SLOT` and `IRQ_FREE_SLOT`.

16.3 Quality Expressions

Quality names may be combined together using Boolean operators into complex *quality expressions*. The quality expression is evaluated at each element within the NDF by substituting a value of true or false for each quality name used in the expression, depending on whether or not that element holds the specified quality. Elements are used if the quality expression evaluates to a true value. Boolean operators are delimited on each side by a period (*e.g.* `.AND.`). The operands on which these operators act must be either a quality name (which must be defined within the NDF), or one of the Boolean constants `.TRUE.` and `.FALSE.` Parentheses can be used to nest expressions.

Quality expressions can be up to 254 characters long, and must not contain more than 40 symbols (Boolean operators, constants, or quality names). Some attempts are made to simplify a quality expression to reduce the run time needed to evaluate the expression for every data value.

The precedence of the Boolean operators decreases in the following order; `.NOT.`, `.AND.`, `.OR.`, `.XOR.`, `.EQV.` (the final two have equal precedence). In an expression such as `( A .XOR. B .EQV. C .XOR. D )` in which all operators have equal precedence, the evaluation proceeds from left to right, *i.e.* the expression is evaluated as `( ( ( A .XOR. B ) .EQV. C ) .XOR. D )`. If there is any doubt about the order in which an expression will be evaluated, parentheses should be used to ensure the required order of evaluation.

.NOT. - The expression `(.NOT.A)` is true only if A is false.

.AND. - The expression `(A.AND.B)` is true only if A and B are both true.

.OR. - The expression `(A.OR.B)` is true if either (or both) of A or B are true.

.XOR. - The expression `(A.XOR.B)` is true if either A is true and B is false, or A is false and B is true.

.EQV. - The expression `(A.EQV.B)` is true if either A is true and B is true, or A is false and B is false.

17 Processing Groups of Data Files

When a KAPPA application requests an input or output data file (either an NDF or a positions list), you may optionally give a group of several data files rather than just one. In this case, the application is automatically re-run until all the supplied data files have been processed. For instance, in the following command:

```
% display in="../a*" mode=perc accept
```

a group of NDFs (all those beginning with "a" in the parent directory) are assigned to the IN parameter, and the DISPLAY command is automatically re-run to display each NDF in the fashion of a (rather slow!) movie.

Another example:

```
% stats ndf=^files
```

will display the pixel statistics of all NDFs listed within the text file `files`. The "^" character indicates that the following string (`files`) is not the name of an NDF, but the name of a text file from which NDF names should be read.

```
% wcsframe '"image_a,image_b,image_c"' sky
```

This will set the current co-ordinate Frame for the three NDFs `image_a`, `image_b` and `image_c` so that celestial sky co-ordinates are used to refer to positions within the NDFs, if possible.

```
% cursor outcat="'first,^list_names'"
```

This will run CURSOR several times, allowing you to select display positions using a cursor. On the first invocation, the selected positions are written to a positions list stored in file `first.FIT`. Positions selected on subsequent invocations are written to positions lists with names read from the text file `list_names`. Finally, if this CURSOR command is followed by:

```
% listshow accept
```

LISTSHOW will display the contents of all the catalogues created previously by CURSOR.

If an application has more than one NDF or positions list parameter, each parameter should be given the same number of values (*i.e.* data files). A warning is issued if any parameter is given too many values, but processing continues normally until the smallest group is exhausted. For instance, the following example adds NDF `a1` to `a2`, and `b1` to `b2`, putting the results in `a3` and `b3`:

```
% add in1="'a1,b1'" in2="'a2,b2'" out="'a3,b3'"
```

If (say) an extra NDF had been specified for Parameter IN2, the application would have been invoked twice to process the first two pairs, and then a warning message would have been displayed saying that too many NDFs were specified for Parameter IN2.

There is a special case in which this rule does not apply. If only a *single* value is given for an data file parameter, the same value is used repeatedly on all invocations of the application. So, for instance, if a single NDF had been given for Parameter IN2 in the above ADD example, the application would have again been run twice, using the same NDF for Parameter IN2 on each invocation.

The OUT parameter in the above example could alternatively have been specified as `out="'*|1|3|'"`. Here, the asterisk (*) represents the base-names, `a1` and `b1`, of the NDFs supplied for the first NDF parameter to be accessed (IN1). The following string, `|1|3|`, means *replace all occurrences of 1 with 3*, thus giving the final NDF names `a3` and `b3`.

17.1 Applications that Process Groups of NDFs

The majority of applications with NDF parameters use each NDF parameter to access a *single* NDF, and supplying more than one NDF will result in the application being re-run as described above. The application then accesses a single NDF on each invocation. Some applications, however, have NDF parameters that are explicitly described in the reference section of this document as being associated with a *group* of NDFs. An example is WCSALIGN that has a Parameter IN to read a *group* of NDFs that are to be aligned with each other. Such applications process all the specified NDFs in a single invocation. For the purposes of the multiple invocation scheme described above, such parameters are not considered to be 'NDF' parameters, and will not cause the application to be re-run.

17.2 What about the other Parameters?

When an application is re-run to process multiple data files, all the parameters not associated with NDFs or positions lists retain their values from one invocation to the next. So, for instance, the assignment for the MODE parameter in the earlier DISPLAY example is retained and used for all subsequent invocations of the application, you are not prompted for a new value each time the application runs.

This is usually what you want, but beware that there *are* times when this behaviour may trip you up. Sometimes an application may prompt for a new parameter value while in the middle of processing a group of NDFs. This can occur for instance, if the initial value you supplied on the first invocation is inappropriate for the NDF currently being processed. For instance, supposing you use WCSFRAME to set the current co-ordinate Frame to SKY for a group of NDFs. To do this, you would set the FRAME parameter to SKY either on the command line or when prompted during the first invocation. This value would be retained for subsequent invocations, but what happens if one of the NDFs does not have a SKY Frame defined in its WCS component? Not surprisingly, you get an error message identifying the NDF, and you are asked to supply a new value for FRAME. You could, for instance supply `PIXEL` as the new value. This changes the current value of the FRAME parameter to PIXEL, and this value will consequently be used for any remaining NDFs.

If you do not specify a value for a parameter, the default value used by the first invocation will be re-used for all subsequent invocations. Note that the default value for some parameters

(for instance the CENTRE parameter of the DISPLAY command) is the null value `!`. This is usually interpreted as a request for the application to find an appropriate value itself for the parameter. In these cases, the parameter *value* is `!` and is re-used on all invocations, resulting in the application finding and using a potentially different value on each invocation. So, for instance, the above DISPLAY example will find and use an appropriate CENTRE value for each displayed image. If you want to use the same CENTRE value for all images you should specify it explicitly on the command line, for instance:

```
% display in="../a*" mode=perc centre="'12:00:00 -32:00:00'" accept
```

17.3 Output Parameters

If you tried out the examples at the start of this section, you may be wondering what happened about the output parameters for STATS. The STATS application writes the various statistics it calculates to lots of output parameters, which can be used by subsequent applications. If an application is re-run several times to process different data files then the values left in the application's output parameters will be the values created on the *last* invocation of the application.

17.4 What Happens if an Error Occurs?

If an application fails to execute successfully, the error report will be displayed, and then cancelled.[24] This means that any remaining NDFs will continued to be processed normally.

The exception to this is that if the error is an 'abort request' (caused by supplying two exclamation marks for a parameter), then the loop exits immediately. That is, no remaining NDFs are processed.

17.5 What about Applications that Re-use Parameters?

Some applications use a single parameter to obtain a series of values from the user. Examples are the INIT parameter of CENTROID and the OPTION parameter of SETEXT. Remembering that parameters that are not associated with either an NDF or a positions list retain their values between invocations, it is not surprising that care is needed when using such application to process groups of NDFs. For instance, when using CENTROID you supply a null parameter value (*i.e.* a single exclamation mark `!`) as the final value for the INIT parameter to indicate that you do not wish to find any more centroids. Since parameter values are retained between invocations when processing groups of NDFs, this null value becomes the first value to be used by any subsequent invocation. The next invocation of CENTROID finds INIT set to a null value, assumes that no more centroids are to be found, and exits immediately! The same goes for all subsequent invocations until the group of NDFs has been exhausted.

The only way (currently) to avoid this behaviour is to specify the INIT parameter value on the command line. CENTROID takes this as an indication that you only want to find a single centroid, and so does not attempt to get a new value for INIT, thus leaving the supplied value

[24] This is called 'flushing' the error.

for the next invocation. The same value for INIT is thus used by all invocations. Of course, this means you can only find a single centroid in each NDF.[25]

Most applications that re-use one or more parameters during a single invocation have some similar means of indicating that you do not want to be prompted for a new value. For some (like CENTROID), putting the parameter value on the command line accomplishes this. Some others (such as SETEXT) have a LOOP parameter that can be set `FALSE` to indicate that parameters should not be accessed more than once. The reference documentation for each command should be consulted for details.

17.6 Introducing a Pause Between Invocations

Sometimes you may want to slow down the speed at which data files are processed. For instance, if you display several small images using a single DISPLAY command, you may want time to examine each image before moving on to the next. You can introduce a delay between invocations by setting the shell environment variable `KAPPA_LOOP_DELAY` to the required delay time (in units of seconds). For instance, in the C-shell:

```
% setenv KAPPA_LOOP_DELAY 2.5
```

causes a delay of 2.5 seconds between invocations of any KAPPA command. To remove the delay, you should undefined `KAPPA_LOOP_DELAY`. In C-shell:

```
% unsetenv KAPPA_LOOP_DELAY
```

17.7 Reporting the Data Files being Processed

When processing a single data file, some applications report the name of the file and some do not. Normally, no extra information is given when processing groups of data files. This means that sometimes you get to see the names of the files as they are processed, and some times you do not. It just depends on the application.

However, it is often very useful to see the names of the files as they are processed. For instance, if an error occurs processing one of the files, it is useful to know which file failed. If the application doesn't display this information, then you can force it to by setting the shell environment variable `KAPPA_REPORT_NAMES` to an arbitrary value.[26] For instance, in the C-shell:

```
% setenv KAPPA_REPORT_NAMES 1
```

This causes the value used for each data file parameter to be displayed in the form `"parameter = value"` on each invocation. To go back to the normal, quiet reporting scheme, you should undefined `KAPPA_REPORT_NAMES`. In C-shell:

```
% unsetenv KAPPA_REPORT_NAMES
```

[25] If this is a problem, you can always put the INIT values into a file or positions list, using a different value for the MODE parameter.

[26] The actual value does not matter.

17.8 The Syntax for Specifying Groups of Data Files

The group of NDFs or positions lists to be used for a given parameter is specified by a *group expression*. This is also the syntax used to give groups of plotting attributes when specifying graphics STYLE parameters. The group expression syntax is described in Section 4.13.

A group of output data files may be specified by modifying the names of a corresponding set of input data files. This is easy enough when the application only has one parameter for input data files, but what happens if more than one parameter is associated with a group of input data files? Which parameter is used to define the group of input data files on which the names of the output data files are based? The answer is "the first one to be accessed". For instance, ADD takes two input NDFs, adds them together and produces a single output NDF. When running ADD, you are prompted first for Parameter IN1, and then for Parameter IN2, and finally for Parameter OUT. Thus, if you give the string `"a_"*` for OUT, the names of the output NDFs will be derived from the NDFs supplied for Parameter IN1, because IN1 is accessed first (*i.e.* prompted for *before* IN2).

Note, when choosing the input parameter on which output data files are based, no significance is attached to whether the input and output file types match. That is, the first input parameter to be accessed if used, irrespective of whether it is associated with an NDF or a positions list.

A feature that may sometimes be useful is the facility for providing a shell command in response to a prompt for a group of data files. To do this, enclose the command within the usual backward quotes (‘), as you would when substituting the output from a command into another shell command. The command should generate a set of explicit file names, with file types. Note, you will need to escape any characters that are normally interpreted as part of the syntax of a group expression, such as `"|"` or `","`, by preceding them with a backslash `"\"`.

17.9 Using non-NDF Data Formats

In addition to processing Starlink NDF structures, KAPPAcan also process many non-NDF ('foreign') data files. This is achieved through 'on-the-fly conversion' (see Section 18.1 and SUN/55).

When this scheme is in use, you need not include explicit file types for all input file names. If no file type is given, the file with the highest priority file type amongst all files with the specified base name will be used. The priority of a file type is determined by its position within the list of file types given by `NDF_FORMATS_IN` environment variable (see Section 18.1). File types near the start of the list have higher priority than those that follow. Note, native NDF files always have the highest priority and will be used (if they exist) in preference to all other files types.

17.10 Disabling Multiple Invocations of Applications

In certain circumstances, you may possibly want to disable the automatic re-invocation of KAPPA applications to process groups of data files. This can be done by setting the environment variable `KAPPA_LOOP_DISABLE` to an arbitrary value.[27] For instance, in the C-shell:

```
% setenv KAPPA_LOOP_DISABLE 1
```

[27] The actual value does not matter.

will cause all NDF and positions list parameters to accept only a single data file, and each application will be run only once. Note, the extra facilities for specifying data files provided by the group expression syntax will not then be available. To re-enable looping, you should undefine KAPPA_LOOP_DISABLE. In C-shell:

```
% unsetenv KAPPA_LOOP_DISABLE
```

18 Getting Data into KAPPA

KAPPA utilises general data structures within an HDS container file, with file extension `.sdf`. Most of the examples in this documentation processing is performed on data in this NDF format generated from within KAPPA. Generally, you will already have data in 'foreign' formats, that is formats other than the Starlink standard, particularly in the FITS (Flexible Image Transport System), IRAF, and FIGARO DST formats.

18.1 Automatic Conversion

Although KAPPA tasks do not work directly with 'foreign' formats, they can made to appear that they do. What happens is that the format is converted 'on-the-fly' to a scratch NDF, which is then processed by KAPPA. If the processing creates an output NDF or modifies the scratch NDF, this may be back-converted 'on-the-fly' too, and not necessarily to the original data format. At the end, the scratch NDF is deleted. So for example you could have an IRAF image file, use BLOCK to filter the array, and output the resultant array as a FITS file.

We must first define the names of the recognised formats and a file extension associated with each format. In practice you'll most likely do this with the `convert` command, which creates these definitions for many popular formats. The file extension determines in which format a file is written. There is an environment variable called `NDF_FORMATS_IN` which defines the allowed formats in a comma-separated list with the file extensions in parentheses. Here is an example.

```
% setenv NDF_FORMATS_IN 'FITS(.fit),IRAF(.imh),FIGARO(.dst)'
```

Once defined it lets you run KAPPA tasks on FITS, IRAF, or FIGARO files, like

```
% stats m51.fit
% stats m51.dst
```

would compute the statistics of a FITS file `m51.fit`, and then a FIGARO file `m51.dst`.

The environment variable also defines a search order. Had you entered

```
% stats m51
```

STATS would first look for an NDF called m51 (stored in file `m51.sdf`). If it could not locate that NDF, STATS would then look for a file called `m51.fit`, and then `m51.imh`, and finally `m51.dst`, stopping once a file was found and associating the appropriate format with it. If none of the files exist, you'll receive a "file not found" error message.

You can still define an NDF section (see Section 9) when you access an existing data file in a foreign format. Thus

```
% stats m51.imh"(100:200,200~81)"
```

would derive the statistics for x pixels between 100 and 200, and y pixels 160 to 240 in the IRAF file `m51.imh`.

The conversion tasks may be your own for some private format, but normally they will come from the CONVERT package (SUN/55). If you want to learn how to add conversions to the standard ones, you should consult SSN/20.

There is an environment variable that defines the format of new data files. This could be assigned the same value as `NDF_FORMATS_OUT`, though they don't have to be.

```
% setenv NDF_FORMATS_OUT 'FITS(.fit),IRAF(.imh),FIGARO(.dst)'
```

If you supply the file extension when a KAPPA task creates a new dataset, and it appears in `NDF_FORMATS_OUT`, you'll get a file in that format. So for instance,

```
% ffclean in=m51.dst out=m51_cleaned.dst \\
```

cleans `m51.dst` and stores the result in `m51_cleaned.dst`. On the other hand, if you only give the dataset name

```
% ffclean in=m51.dst out=m51_cleaned \\
```

the output dataset would be the first in the `NDF_FORMATS_OUT` list. Thus if you want to work predominantly in a foreign format, place it first in the `NDF_FORMATS_IN` and `NDF_FORMATS_OUT` lists.

If you want to create an output NDF, you must insert a full stop at the head of the list.

```
% setenv NDF_FORMATS_OUT '.,FITS(.fit),IRAF(.imh),FIGARO(.dst)'
```

This is the recommended behaviour. If you just want to propagate the input data format, insert an asterisk at the start of the output-format list.

```
% setenv NDF_FORMATS_OUT '*,.,FITS(.fit),IRAF(.imh),FIGARO(.dst)'
```

This only affects applications that create a dataset using information propagated from an existing dataset. For instance, if the above `NDF_FORMATS_OUT` were defined,

```
% ffclean in=m51.dst out=m51_cleaned \\
```

would now create `m51_cleaned.dst`. If there is no propagation in the given application, the asterisk is ignored.

You can retain the scratch NDF by setting the environment variable `NDF_KEEP` to 1. This is useful if you intend to work mostly with NDFs and will save the conversion each time you access the dataset.

The `convert` command, which sets up definitions for the CONVERT package, defines the lists of input and output formats as follows.

```
% setenv NDF_FORMATS_IN \
'FITS(.fit),FIGARO(.dst),IRAF(.imh),STREAM(.das),UNFORMATTED(.unf),UNF0(.dat),
ASCII(.asc),TEXT(.txt),GIF(.gif),TIFF(.tif),GASP(.hdr),COMPRESSED(.sdf.Z),
GZIP(.sdf.gz),FITS(.fits),FITS(.fts),FITS(.FTS),FITS(.FITS),FITS(.FIT),
FITS(.lilo),FITS(.lihi),FITS(.silo),FITS(.sihi),FITS(.mxlo),FITS(.rilo),
FITS(.rihi),FITS(.vdlo),FITS(.vdhi),STREAM(.str)'

% setenv NDF_FORMATS_OUT \
'.,FITS(.fit),FIGARO(.dst),IRAF(.imh),STREAM(.das),UNFORMATTED(.unf),
UNF0(.dat),ASCII(.asc),TEXT(.txt),GIF(.gif),TIFF(.tif),GASP(.hdr),
COMPRESSED(.sdf.Z),GZIP(.sdf.gz)'
```

See the CONVERT documentation for more details of these conversions.

18.2 Other Routes for Data Import

You can run CONVERT (*cf.* SUN/55) directly to perform conversions. There is also TRANDAT, which will read a text file of data values, or co-ordinates and data values into an NDF, and ASCIN in the FIGARO package (SUN/86).

18.3 FITS readers

The automatic conversion does not allow you the full control of the conversion that direct use of a FITS reader offers and it does not deal with the special properties of tape. For full control of the conversion process, you should use the FITS2NDF and MTFITS2NDF commands form the CONVERTpackage. FITS2NDF reads disk FITS files, and MTFITS2NDF reads FITS files from magnetic tape.

For historical reasons, KAPPA contains its own additional FITS readers; FITSIN for reading data from tape, and FITSDIN for reading data from disk. These do not currently have all the features of the corresponding CONVERTcommands (for instance, they do not allow an NDF to be created from a specified FITS extension). For this reason, you should normally use the CONVERT commands described in SUN/55.

Let's see the KAPPA FITS readers in action.

18.3.1 Reading FITS Tapes

FITSIN reads FITS files stored on tape. For efficiency, you should select the 'no-rewind' device for the particular tape drive, for example `/dev/nrmt0h` on OSF/1 and `/dev/rmt/1n` on Solaris.

We ask for the second file on the tape, and the headers are displayed so we can decide whether this is the file we want. It is so we supply a name of an NDF to receive the FITS file. If it wasn't we would enter `!` to the OUT prompt. The FMTCNV parameter asks whether the data are to be converted to _REAL, using the FITS keywords BSCALE and BZERO, if present. If you are wondering why there is `(1)` after the file number, that's present because FITS files can have sub-files, stored as FITS extensions.

```
% fitsin
MT - Tape deck /@/dev/nrmt0h/ >
```

```
The tape is currently positioned at file 1.
FILES - Give a list of the numbers of the files to be processed > 2
File # 2(1)  Descriptors follow:
SIMPLE  =                    T
BITPIX  =                   16
NAXIS   =                    2
NAXIS1  =                  400
NAXIS2  =                  590
DATE    = '03/07/88'                  /Date tape file created
ORIGIN  = 'ING     '                  /Tape writing institution
OBSERVER= 'CL      '                  /Name of the Observer
TELESCOP= 'JKT     '                  /Name of the Telescope
INSTRUME= 'AGBX    '                  /Instrument configuration
OBJECT  = 'SYS:ARCCL.002'             /Name of the Object
BSCALE  =                  1.0        /Multiplier for pixel values
BZERO   =                  0.0        /Offset for pixel values
BUNIT   = 'ADU     '                  /Physical units of data array
BLANK   =                    0        /Value indicating undefined pixel
            :                :              :
            :                :              :
            :                :              :
END
FMTCNV - Convert data? /NO/ >
OUT - Output image > ff1
Completed processing of tape file 2 to ff1.
MORE - Any more files? /NO/ >
```

We can trace the structure to reveal the 2-byte integer CCD image. Notice that the FITS headers are stored verbatim in a component .MORE.FITS. This is the FITS extension. The extension contents can be listed with FITSLIST. There is more on this NDF extension and its purpose in Section 18.4.

```
% hdstrace ff1
FF1  <NDF>

   DATA_ARRAY(400,590)  <_WORD>   216,204,220,221,202,222,220,206,218,221,
                                  ... 216,218,218,204,221,218,219,222,221,218
   TITLE          <_CHAR*13>      'SYS:ARCCL.002'
   UNITS          <_CHAR*3>       'ADU'
   MORE           <EXT>           {structure}
      FITS(84)       <_CHAR*80>      'SIMPLE  =                    T','BI...'
                                     ... '   ...','         ING PACKEND','END'

End of Trace.
```

If you have many FITS files to read there is a quick method for extracting all files or a selection. In automatic mode the output files are generated without manual intervention and the headers aren't reported for efficiency. Should you want to see the headers, write them to a text file via the LOGFILE parameter. The cost of automation is a restriction on the names of the output files, but if you have over a hundred files on a tape are you really going to name them individually?

The following example extracts the fourth to sixth, and eighth files. Note that the `[ ]` are needed because the value for Parameter FILES is a character array.

```
% fitsin auto
MT - Tape deck /@/dev/nrmt0h/ >
FMTCNV - Convert data? /NO/ > y
PREFIX - Prefix for the NDF file names? /'FITS'/ > JKT
FILES - Give a list of the numbers of the files to be processed > [4-6,8]
Completed processing of tape file 4 to JKT4.
Completed processing of tape file 5 to JKT5.
Completed processing of tape file 6 to JKT6.
Completed processing of tape file 8 to JKT8.
MORE - Any more files? /NO/ >
```

You can list selected FITS headers from a FITS tape without attempting to read in the data into NDFs by using FITSHEAD. You can redirect its output to a file to browse at your leisure, and identify the files you want to convert. So for instance,

```
% fitshead /dev/nrmt1h > headers.lis
```

lists all the FITS headers from a FITS tape on device `/dev/nrmt1h` to file `headers.lis`.

After running FITSIN you may notice a file `USRDEVDATASET.sdf` in the current directory. This HDS file records the current position of the tape, so you can use FITSIN to read a few files, and then run it again a little later, and FITSIN can carry on from where you left off. In other words FITSIN does not have to rewind to the beginning of the tape to count files. When you're finished you should delete this file.

18.3.2 Reading FITS Files

For many years there was officially no such thing as disc FITS. However, *ad hoc* implementations have existed for a long time. Of these, FITSDIN will handle files adhering to the FITS rules for blocking (and more), but it doesn't process byte-swapped 'FITS' files. Thus it can process files with fixed-length records of semi-arbitrary length; so, for example, files mangled during network transfer, which have 512-byte records rather than the customary 2880, may be read. However, it will not handle, VAX FITS files as may be produced with FIGARO's WDFITS. FITSDIN will accept a list of files with wildcards. However, a comma-separated list must be enclosed in quotation marks. Also wildcards must be protected. Here are some examples so you get the idea.

```
% fitsdin '*.fit'
% fitsdin \*.fit
ICL> fitsdin *.fit
% fitsdin '"i*.fit,abc123.fts"'
ICL> fitsdin "i*.fit,abc123.fts"
```

In the following example a floating-point file is read (BITPIX=−32) and so FMTCNV is not required.

```
% fitsdin '*.fits'

   2 files to be processed...
```

```
Processing file number 1: /home/scratch/dro/gr.fits.
File /scratch/dro/gr.fits(1)  Descriptors follow:
SIMPLE  =                    T / Standard FITS format
BITPIX  =                  -32 / No. of bits per pixel
NAXIS   =                    2 / No. of axes in image
NAXIS1  =                  512 / No. of pixels
NAXIS2  =                  256 / No. of pixels
EXTEND  =                    T / FITS extension may be present
BLOCKED =                    T / FITS file may be blocked

BUNIT   = 'none given      '   / Units of data values

CRPIX1  =   1.000000000000E+00 / Reference pixel
CRVAL1  =   0.000000000000E+00 / Coordinate at reference pixel
CDELT1  =   1.000000000000E+00 / Coordinate increment per pixel
CTYPE1  = '                '   / Units of coordinate
CRPIX2  =   1.000000000000E+00 / Reference pixel
CRVAL2  =   0.000000000000E+00 / Coordinate at reference pixel
CDELT2  =   1.000000000000E+00 / Coordinate increment per pixel
CTYPE2  = '                '   / Units of coordinate

ORIGIN  = 'ESO-MIDAS'          / Written by MIDAS
OBJECT  = 'artificial image'   / MIDAS desc.: IDENT(1)
        :                :               :
        :                :               :
        :                :               :
HISTORY  ESO-DESCRIPTORS END     ................

END
OUT - Output image > gr
Completed processing of disc file /home/scratch/dro/gr.fits to gr.
File has illegal-length blocks (512).  Blocks should be a multiple (1--10) of the
FITS record length of 2880 bytes.
Processing file number 2: /home/scratch/dro/indef.fits.
File /home/scratch/dro/indef.fits(1)  Descriptors follow:
SIMPLE  =                    T  /  FITS STANDARD
BITPIX  =                   32  /  FITS BITS/PIXEL
NAXIS   =                    2  /  NUMBER OF AXES
NAXIS1  =                  256  /
NAXIS2  =                   20  /
BSCALE  =      3.7252940008E28  /  REAL = TAPE*BSCALE + BZERO
BZERO   =      7.9999999471E37  /
OBJECT  = 'JUNK[1/1]'  /
ORIGIN  = 'KPNO-IRAF'  /
        :                :               :
        :                :               :
        :                :               :
END
OUT - Output image > iraf
Completed processing of disc file /home/scratch/dro/indef.fits to iraf.
```

NDFTRACE shows that the object name is written to the NDF's title, that axes derived from the FITS headers are present, and that gr is a _REAL NDF.

```
% ndftrace gr

   NDF structure /home/scratch/dro/iraf:
      Title:  artificial image
      Units:  none given

   Shape:
      No. of dimensions:  2
      Dimension size(s):  512 x 256
      Pixel bounds     :  1:512, 1:256
      Total pixels     :  131072

   Axes:
      Axis 1:
         Label : Axis 1
         Units : pixel
         Extent: -0.5 to 511.5

      Axis 2:
         Label : Axis 2
         Units : pixel
         Extent: -0.5 to 255.5

   Data Component:
      Type        :  _REAL
      Storage form:  PRIMITIVE
      Bad pixels may be present

   Extensions:
         FITS             <_CHAR*80>
```

Both FITSIN and FITSDIN write the FITS headers into an NDF extension called FITS within your NDF. The extension is a literal copy of all the 80-character 'card images' in order. These can be inspected or written to a file via the command FITSLIST. There is more on this NDF extension and its purpose in Section 18.4.

18.4 The FITS Airlock

18.4.1 NDF Extensions

An important feature of the NDF is that it is designed to be extensible. The NDF has components whose meanings are well defined and universal, and so they can be accessed by general-purpose software, such as KAPPA and CONVERT provide; but the NDF also allows independent *extensions* to be defined and added, which can store auxiliary information to suit the needs of a specialised software package. (Note that the term extension here refers to a structure within the NDF for storing additional data, and is neither the file extension `.sdf` nor extensions like BINTABLE within the FITS file.) An extension is only processed by software that understands the meanings obeys the processing rules of the various components of the extension. Other programmes propagate the extension information unaltered.

The existence of extensions makes it straightforward to write general utilities for converting an arbitrary format into an NDF. The idea being that every specialist package should not have to

have its own conversion tools such as a FITS reader. However, this still leaves the additional data that requires specialist knowledge to move it into the appropriate extension components. The aim is to make the conversions themselves extensible, with add-on operations to move the specialist information to and from the extensions. This is where the FITS 'airlock' comes in.

The FITS data format comprises a header followed by the data array or table. The header contains a series of 80-character lines each of which contains the keyword name, a value and an optional comment. There are also some special keywords for commentary. The meanings of most keywords are undefined, and so can be used to transport arbitrary ancillary information, subject to FITS syntax limitations. There is a special NDF extension called FITS, which mirrors this functionality, and may be added to an NDF. It therefore can act as an airlock between the general-purpose conversion tools and specialist packages.

18.4.2 Importing and Exporting from and to the FITS Extension

The FITS extension comprises a one-dimensional array of 80-character strings that follow FITS-header formatting rules. In the case of FITSIN and FITSDIN, each FITS extension is a verbatim copy of the FITS header of the input file. Other conversion tools like IRAF2NDF and UNF2NDF of CONVERT can also create a FITS extension in the same fashion. On export, standard conversion tools propagate the FITS extension to any FITS headers or equivalent in the foreign format. However, information which is derivable from the standard NDF components, such as the array dimensions, data units, and linear axes, replaces any equivalent headers from FITS extension.

You use your knowledge, or the writer of the specialist package provides import tools, to recognise certain FITS keywords and to attribute meaning to them, and then to move or process their values to make the specialist extensions. One such is the PREPARE task in IRAS90. Similarly, the reverse operation—exporting the extension information—can occur too, prior to converting the NDF into another data format.

KAPPA offers two simple tools for the importing and exporting of extension information: FITSIMP and FITSEXP. They both use a text file, which acts as a translation table between the FITS keyword and extension components. Starting with FITSIMP, its translation table might look like this.

```
ORDER_NUMBER _INTEGER  ORDNUM
PLATE_SCALE  _REAL SCALE         ! The plate scale in arcsec/mm
SMOOTHED  _LOGICAL FILTERED
```

It consists of three fields: the first is the name of the component in the chosen extension, the second is the HDS data type of that component, and the third is the FITS keyword. Optional comments can appear following an exclamation mark. So if we placed these lines in file `imptable`, we could create an extension called MYEXT of data type MJC_EXT (if it did not already exist) containing components ORDER_NUMBER, PLATE_SCALE, and SMOOTHED.

```
% fitsimp mydata imptable myext mjc_ext
```

Should any of the keywords not exist in the FITS extension, you'll be warned. If the extension already exists, you don't need to specify the extension data type. FITSIMP will even handle hierarchical keywords and those much-loved ING packets from La Palma.

Going in the opposite direction, the text translation file could look like this

```
MYEXT.ORDER_NUMBER  ORDNUM(LAST) The spectral order number
MYEXT.PLATE_SCALE   SCALE   The plate scale in arcsec/mm
MYEXT.SMOOTHED  FILTERED
```

where the first column is the 'name' of the extension component to be copied to the FITS extension. The 'name' includes the extension name and substructures. The second column gives the FITS keyword to which to write the value. A further keyword in parentheses instructs FITSEXP to place the new FITS header immediately before the header with that keyword. If the second keyword is absent from the translation-table record or the FITS extension, the new header appears immediately before the END header line in the FITS extension. Thus the value of ORDER_NUMBER in extension MYEXT, creates a new keyword in the FITS extension called ORDNUM, and it is located immediately prior the keyword LAST.

18.4.3 Listing the FITS Extension and keywords

If you don't want to be bothered with NDF extensions, you might just want to know the value of some FITS keyword, say the exposure time, as part of your data processing. FITSLIST lists the contents of the FITS extension of an NDF or file. You can even search for keywords with **grep**.

```
% fitslist myndf | grep "ELAPSED ="
```

This would find the keyword ELAPSED in the FITS extension of NDF myndf. (Keywords are 8 characters long and those with values are immediately followed by an equals sign.) However, the recommended way is to use the FITSVAL command. Since this command only reports the value, it is particularly useful in scripts that need ancillary-data values during processing. The following obtains the value of keyword ELAPSED.

```
% fitsval myndf ELAPSED
```

In a script you may need to know whether the keyword exists and take appropriate action.

```
filterpre = `fitsexist myndf filter`
if ( $filterpre == "TRUE" ) then
   filter = `fitsval myndf filter`
else
   prompt -n "Filter > "
   set filter = $<
endif
```

Shell variable `filterpres` would be assigned `"TRUE"` when the FILTER card is present, and `"FALSE"` otherwise. (The ` ` quotes cause the enclosed command to be executed.) So the user of the script would be prompted for a filter name whenever the NDF did not contain that information.

18.4.4 Creating and Editing the FITS Extension

Besides the conversion utilities, you can import your own FITS extension using FITSTEXT. You first prepare a FITS-like header in a text file. For example,

```
% fitstext myndf myfile
```

places the contents of `myfile` in the NDF called myndf. This is not advised unless you are familiar with the rules for writing FITS headers. See the NOST *A User's Guide to FITS* (URL `http://archive.stsci.edu/fits/users_guide/`). Other useful FITS documents, test files, and software are available at the FITS Support Office Home Page (URL `http://fits.gsfc.nasa.gov/`).

FITSTEXT does perform some limited validation of the FITS headers, and informs you of any problems it detects. See the FITSHEAD Notes in Appendix C for details.

A safer bet for a hand-crafted FITS extension is to edit an existing FITS extension to change a value, or use existing lines as templates for any new keywords you wish to add. FITSEDIT lets you do this with your favourite text editor. Define the environment variable `EDITOR` to your editor, say

```
% setenv EDITOR jed
```

to choose **jed**. If you don't do this, and EDITOR is unassigned, FITSEDIT selects the **vi** editor. Then to edit the NDF extension is simple.

```
% fitsedit myndf
```

This edits the FITS extension of the NDF called myndf. FITSEDIT extracts the file into a temporary file (`zzfitsedit.tmp`) which you edit, and then uses FITSTEXT to restore the FITS extension. It therefore has the same parsing of the edited FITS headers as FITSTEXT provides.

18.4.5 Easy way to create and edit the FITS Extension

Should you wish to write a new value without knowing about FITS, or in a script where manual editing is undesirable, the FITSWRITE command does the job. So for example,

```
% fitswrite myndf filter value=K
```

will create a keyword FILTER with value `K` in the FITS extension of the NDF called myndf. If the extension does not exist, this command will first create it.

The FITSMOD command has several editing options including the ability to delete a keyword:

```
% fitsmod myndf airmass edit=delete
```

here it removes the AIRMASS header; or rename a keyword:

```
% fitsmod myndf band rename newkey=filter
```

as in this example, where keyword BAND becomes keyword FILTER; or update an existing keyword:

```
% fitsmod myndf filter edit=u value=\$V comment='"Standard filter name"'
```

this example modifies the comment string associated with the FILTER keyword, leaving the value unchanged.

For routine operations requiring many operations on a dataset, FITSMOD lets you specify the editing instructions in a text file.

19 Procedures

Applications from KAPPA and other packages can be combined in procedures and scripts to customise and automate data processing. In addition to giving literal values to application parameters, you can include ICL or C-shell variables on the command line, whose values are substituted at run time. It is also possible to write parameter data into variables, and hence pass them to another application, or use the variables to control subsequent processing.

19.1 C-shell scripts

The *C-shell Cookbook* contains many ingredients and recipes, and features many KAPPA commands. So there is little point repeating them here other than to direct you to a documented script in `$KAPPA_DIR/multiplot.csh`.

19.2 ICL Procedures

You should consult the *ICL Users' Guide* for details about writing ICL syntax, procedures, and functions, but you're a busy researcher...For a quick overview the *two-page* summary on "Writing ICL command files and procedures" in SUN/101 is recommended reading, even though much of the document is dated and still refers to VMS. Here we'll just show some example procedures that can be adapted and cover points not mentioned in SUN/101.

Let's start with something simple. You want to 'flash' a series of images, each with a yellow border. First you write the following procedure called FLASH. It has one argument INPIC, that passes the name of the NDF you want to display. When you substitute an ICLvariable for a parameter value you enclose it in parentheses. The lines beginning with `{` are comments.

```
PROC FLASH INPIC
{
{ Procedure for displaying an image without scaling.
{
   DISPLAY IN=(INPIC) MODE=FL
END PROC
```

To make ICL recognise your procedure you must 'load' it. The command

```
ICL> LOAD FLASH
```

will load the file `FLASH.ICL`. Thereafter in the ICL session you can invoke FLASH for many NDFs. The following will display the NDFs called GORDON and FLOOD side-by-side.

```
ICL> PICGRID 2 1
ICL> FLASH GORDON
ICL> PICSEL 2
ICL> FLASH FLOOD
```

It would be tedious to have to load lots of individual procedures, but you don't. If you have related procedures that you regularly require they can be concatenated into a single file which you load. Better still is to add definitions for each of the procedures in your ICL login file. This is defined as the value of the `ICL_LOGIN` environment variable. A reasonable place is in your home directory and you'd define it like this.

```
% setenv ICL_LOGIN $HOME/login.icl
```

However, the file doesn't have to be in your home directory, or called `login.icl`, but it's convenient to do so. Suppose you have three procedures: FLASH, PICGREY in file `$MY_DIR/display_proc.icl`, and FILTER in `/home/user1/dro/improc.icl`. In your `$HOME/login.icl` you could add the following

```
defproc  flash     $MY_DIR/display_proc.icl
defproc  sfilt     $HOME/user1/dro/improc.icl filter
defproc  picgr(ey) $MY_DIR/display_proc.icl
```

which defines three commands that will be available each time you use ICL: FLASH which will run your FLASH procedure, PICGREY to execute the PICGREY procedure, and SFILT which runs the FILTER procedure. In addition PICGREY can be abbreviated to PICGR or PICGRE. So now you can load and run your procedure. Let's have some more example procedures.

Suppose you have a series of commands to run on a number of files. You could create a procedure to perform all the stages of the processing, deleting the intermediate files that it creates.

```
PROC UNSHARPMASK NDFIN CLIP NDFOUT

{ Insert ampersands to tell the command-line interpreter than these
{ strings are file names.
   IF SUBSTR( NDFIN, 1, 1 ) <> '@'
      NDFIN = '@' & (NDFIN)
   END IF
   IF SUBSTR( JUNK, 1, 1 ) <> '@'
      NDFOUT = '@' & (NDFOUT)
   END IF

{ Clip the image to remove the cores of stars and galaxies above
{ a nominated threshold.
   THRESH (NDFIN) TMP1 THRHI=(CLIP) NEWHI=(CLIP) \

{ Apply a couple of block smoothings with boxsizes of 5 and 13
{ pixels.  Delete the temporary files as we go along.
   BLOCK tmp1 tmp2 BOX=5
   ! rm tmp1.sdf
   BLOCK tmp2 tmp3 BOX=13
   ! rm tmp2.sdf

{ Multiply the smoothed image by a scalar.
   CMULT tmp3 0.8 tmp4
   ! rm tmp3.sdf
```

```
{ Subtract the smoothed and renormalised image from the input image.
{ The effect is to highlight the fine detail, but still retain some
{ of the low-frequency features.
   SUB (NDFIN) tmp4 (NDFOUT)
   ! rm tmp4.sdf
END PROC
```

There is a piece of syntax to note which often catches people out. Filenames, data objects, and devices passed via ICL variables to applications, such as NDFIN and NDFOUT in the above example, must be preceded by an `@`.

A common use of procedures is likely to be to duplicate processing for several files. Here is an example procedure that does that. It uses some intrinsic functions which look just like Fortran.

```
PROC MULTISTAT

{ Prompt for the number of NDFs to analyse.  Ensure that it is positive.
   INPUTI Number of frames:  (NUM)
   NUM = MAX( 1, NUM )

{ Find the number of characters required to format the number as
{ a string using a couple of ICL functions.
   NC = INT( LOG10( I ) ) + 1

{ Loop NUM times.
   LOOP FOR I=1 TO (NUM)

{ Generate the name of the NDF to be analysed via the ICL function
{ SNAME.
     FILE = '@' & SNAME('REDX',I,NC)

{ Form the statistics of the image.
     STATS NDF=(FILE)
   END LOOP
END PROC
```

If NUM is set to 10, the above procedure obtains the statistics of the images named REDX1, REDX2, ... REDX10. The ICL variable FILE is in parentheses because its value is to be substituted into Parameter NDF.

Here is another example, which could be used to flat field a series of CCD frames. Instead of executing a specific number of files, you can enter an arbitrary sequence of NDFs. When processing is completed a !! is entered rather than an NDF name, and that exits the loop. Note the ~ continuation character (it's not required but it's included for pedagogical reasons).

```
PROC FLATFIELD

{ Obtain the name of the flat-field NDF.  If it does not have a
{ leading @ insert one.
   INPUT "Which flat field frame?: " (FF)
   IF SUBSTR( FF, 1, 1 ) <> '@'
```

```
      FF = '@' & (FF)
   END IF

{ Loop until there are no further NDFs to flat field.
   MOREDATA = TRUE
   LOOP WHILE MOREDATA

{ Obtain the frame to flat field.  Assume that it will not have
{ an @ prefix.  Generate a title for the flattened frame.
      INPUT "Enter frame to flat field (!! to exit): " (IMAGE)
      MOREDATA = IMAGE <> '!!'
      IF MOREDATA
         TITLE = 'Flat field of ' & (IMAGE)
         IMAGE = '@' & (IMAGE)

{ Generate the name of the flattened NDF.
         IMAGEOUT = (IMAGE) & 'F'
         PRINT Writing to (IMAGEOUT)

{ Divide the image by the flat field.
         DIV IN1=(IMAGE) IN2=(FF) OUT=(IMAGEOUT) ~
           TITLE=(TITLE)
      END IF
   END LOOP
END PROC
```

Some KAPPA applications, particularly the statistical ones, produce output parameters, which can be passed between applications via ICL variables. Here is an example to draw a contour plot centred about a star in a nominated data array from only the star's approximate position. The region about the star is stored in an output NDF file. Note the syntax required to define the value of Parameter INIT; the space between the left bracket and parenthesis is essential.

```
PROC COLSTAR FILE,X,Y,SIZE,OUTFILE

{+
{  Arguments:
{     FILE = FILENAME (Given)
{        Input NDF containing one or more star images.
{     X = REAL (Given)
{        The approximate x position of the star.
{     Y = REAL (Given)
{        The approximate y position of the star.
{     SIZE = REAL (Given)
{        The half-width of the region about the star's centroid to be
{        plotted and saved in the output file.
{     OUTFILE = FILENAME (Given)
{        Output primitive NDF of 2*%SIZE+1 pixels square (unless
{        constrained by the size of the data array or because the location
{        of the star is near an edge of the data array.
{-

{ Ensure that the filenames have the @ prefix.
   IF SUBSTR( FILE, 1, 1 ) <> '@'
```

```
      NDF = '@' & (FILE)
   ELSE
      NDF = (FILE)
   END IF
   IF SUBSTR( OUTFILE, 1, 1 ) <> '@'
      NDFOUT = '@' & (OUTFILE)
   ELSE
      NDFOUT = (OUTFILE)
   END IF

{ Search for the star in a 21x21 pixel box.  The centroid of the
{ star is stored in the ICL variables XC and YC.
   CENTROID NDF=(NDF) INIT=[ (X&','&Y)] XCEN=(XC) YCEN=(YC) ~
     MODE=INTERFACE SEARCH=21 MAXSHIFT=14

{ Convert the co-ordinates to pixel indices.
   IX = NINT( XC + 0.5 )
   IY = NINT( YC + 0.5 )

{ Find the upper and lower bounds of the data array to plot.  Note
{ this assumes no origin information in stored in the data file.
   XL = MAX( 1, IX - SIZE )
   YL = MAX( 1, IY - SIZE )
   XU = MAX( 1, IX + SIZE )
   YU = MAX( 1, IY + SIZE )

{ Create a new NDF centred on the star.
   NDFCOPY IN=(NDF)((XL):(XU),(YL):(YU)) OUT=(NDFOUT)

{ Draw a contour plot around the star on the current graphics device
{ at the given percentiles.
   CONTOUR NDF=(NDFOUT) MODE=PE PERCENTILES=[80,90,95,99]

{ Exit if an error occurred, such as not being able to find a star
{ near the supplied position, or being unable to make the plot.
   EXCEPTION ADAMERR
      PRINT Unable to find or plot the star.
   END EXCEPTION
END PROC
```

20 Problems Problems

20.1 Errors

A detailed list of error codes and their meanings is not available. KAPPA produces descriptive contextual error messages, which are usually straightforward to comprehend. Some of these originate in the underlying infrastructure software. Error messages from KAPPA begin with the name of the application reporting the error. The routine may have detected the error, or it has something to say about the context of the error.

The remainder of the section describes some difficulties you may encounter and how to overcome them. Please suggest additions to this compilation.

20.2 No Match

When running KAPPA from the UNIX shell, your command fails with a "No Match" error message. This means you have forgotten to protect a wildcard character, such as `*`, `?`, so they they are passed to the KAPPA command and not interpreted by the UNIX shell. You can precede the wildcard character with `\`, or surround the wildcard characters in `" "` quotes. Here are some examples.

```
% ndftrace ccd\?_ff
% stats \*118_"[b-d]?"
```

20.3 Unable to Obtain Work Space

Error messages like "Unable to create a work array" may puzzle you. They are accompanied by additional error messages that usually pinpoint the reason for the failure of the application to complete. Many applications require temporary or work space to perform their calculations. This space is stored in an HDS file within directory `$HDS_SCRATCH` and most likely is charged to your disc quota. (If you have not redefined this environment variable, it will point to your current directory.) So one cause for the message is insufficient disc quota available to store the work space container file or to extend it. A second reason for the message is that your computer cannot provide sufficient virtual memory to map the workspace. In this case you can try increasing your process limits using the C-shell built-in function `limit`. You can find your current limits by entering `limit`. You should see a list something like this.

```
cputime         unlimited
filesize        unlimited
datasize        131072 kbytes
stacksize       2048 kbytes
coredumpsize    unlimited
memoryuse       89232 kbytes
vmemoryuse      1048576 kbytes
descriptors     4096
```

The relevant keywords are `datasize` and the `vmemoryuse`. In effect `datasize` specifies the maximum total size of data files you can map at one time in a single programme. The default should be adequate for most purposes and only need be modified for those working with large images or cubes. The `vmemoryuse` specifies the maximum virtual memory you can use.

```
% limit datasize 32768
```

sets the maximum size of mapped data to 32 megabytes. Values cannot exceed the system limits. You can list these with the `-h` qualifier.

```
% limit -h
cputime         unlimited
filesize        unlimited
datasize        1048576 kbytes
stacksize       32768 kbytes
coredumpsize    unlimited
memoryuse       89232 kbytes
vmemoryuse      1048576 kbytes
descriptors     4096
```

Although you can set your limits to the system maxima, it doesn't mean that you should just increase your quotas to the limits. You might become unpopular with some of your colleagues, especially if you accidentally try to access a huge amount of memory. If you cannot accommodate your large datasets this way, you should fragment your data array, and process the pieces separately.

After receiving this error message in an ICLsession you may need to delete the scratch file by hand. The file is called `txxx.sdf`, where `xxxx` is a process identifier. A normal exit from ICLwill delete the work-space container file.

20.4 Application Automatically Picks up the Wrong NDF

Some applications read the name of the NDF used to create a plot or image from the graphics database in order to save typing. Once in a while you'll say "that's not the one I wanted". This is because AGI finds the last `DATA` picture situated within the current picture. Abort the application via `!!`, then use PICCUR or PICLIST to select the required `FRAME` picture enclosing the `DATA` picture, or even select the latter directly. You can override the AGI NDF also by specifying the required NDF on the command line, provided it has pixels whose indices lies within the world co-ordinates of the `DATA` picture. Thus

```
% inspect myndf
```

will inspect the NDF called myndf. The command PICIN will show the last DATA picture and its associated NDF.

20.5 Unable to Store a Picture in the Graphics Database

You may receive an error message, which says failed to store such-and-such picture in the graphics database. For some reason the database was corrupted due to reasons external to KAPPA. Don't worry, usually your plot will have appeared, and to fix the problem run GDCLEAR or delete the database file (`$AGI_USER/agi_`*<node>*`.sdf`, where you substitute your system's node name for *<node>*). You will need to redraw the last plot if you still require it, say for interaction.

20.6 Line Graphics are Invisible on an Graphics Device

The reason for invisible line graphics on your graphics device is because it is drawn in black or a dark grey. Most likely is that some person has been using other software on your graphics device or that is has been reset. PALDEF will set up the default colours for the palette, and so most line graphics will then appear in white. Alternatively,

```
% palentry 1 white
```

will normally suffice.

20.7 Error Obtaining a Locator to a Slice of an HDS array

If the above error appears from DAT_SLICE and you are (re)prompted for an NDF , the most likely cause is that you have asked an IMAGE application to process an NDF section. Use NDFCOPY to make a subset before running the application in question, or process the whole NDF.

20.8 Badly placed ()'s

This means that you have forgotten to 'escape' parentheses, probably when defining an NDF section in the UNIX shell. Try inserting a backslash before each parenthesis or enclosing all the special characters inside " " quotes.

```
% stats myndf\(100:200,\)
% linplot spectrum"(5087.0~30)"
```

20.9 Attempt to use 'positional' parameter value (x) in an unallocated position

Check the usage of the application you are running. One way of adding positional parameters unintentionally, is to forget to escape the " from the shell when supplying a string with spaces or wildcards. For example, this error would arise if we entered

```
% settitle myndf "A title"
```

instead of say

```
% settitle myndf '"A title"'
```

which protects all special characters between the single quotes.

20.10 The choice x is not in the menu. The options are...

You have either made an incorrect selection, or you have forgotten to escape a metacharacter. For the former, you can select a new value from the list of valid values presented in the error message. For the latter, part of another value is being interpreted as a positional value for the parameter the task is complaining about.

```
% linplot $KAPPA_DIR/spectrum style="Title=Red-giant plot"
!! The choice plot is not in the menu.  The options are
!     Data,Quality,Error,Variance.
!  Invalid selection for Parameter COMP.
```

Here it thinks that `plot` is a positional value. Escape the `"` to cure the problem.

```
% linplot $KAPPA_DIR/spectrum style='"Title=Red-giant plot"'
```

20.11 Annotated axes show the wrong co-ordinate system

Each NDF has an associated *current co-ordinate system* which is used when reporting positions within the NDF, or when obtaining positions from the user. If you want to either see, or give, positions in a *different* co-ordinate system, you need to change the current co-ordinate system (more often called the current co-ordinate *frame*) of the NDF by using command WCSFRAME. For instance,

```
% wcsframe m57 pixel
```

will cause all subsequent commands to use pixel co-ordinates when reporting positions, or obtaining positions.

20.12 "I've Got This FITS Tape"

Certain combinations of magnetic tape produced on one model of tape drive but read on a different model seem to generate parity errors that are detected by the MAG_ library that FITSIN uses. However, this doesn't mean that you won't be able to read your FITS tape. The UNIX tape-reading commands seem less sensitive to these parity errors.

Thus you should first attempt to convert the inaccessible FITS files on tape to disc files using the UNIX **dd** command, and then use the FITSDIN application to generate the output NDF or foreign format. For example to convert a FITS file from device `/dev/nrst0` to an NDF called ndfname, you might enter

```
% dd if=/dev/nrst0 ibs=2880 of=file.fits
% fitsdin files=file.fits out=ndfname
% rm file.fits
```

where `file.fits` is the temporary disc-FITS file. The 2880 is the length of a FITS record in bytes. Repeated **dd** commands to a no-rewind tape device (those with the `n` prefix on OSF/1 and the `n` suffix on Solaris) will copy successive files. To skip over files or rewind the tape, use the **mt** command. For example,

```
% mt -f /dev/rmt/1n fsf 3
      :       :       :
% mt -f /dev/rmt/1n asf 4
```

moves the tape on device `/dev/rmt/1n` forward three files, then moves to the fourth file,

```
% mt bsf 2
```

moves back two files on the default tape drive (defined by the environment variable `TAPE`), and

```
% mt -f /dev/nrmt0h rewind
```

rewinds to the start of the tape on device `/dev/nrmt0h`. Thus it is possible to write a script for extracting and converting a series of files including ranges, just like FITSIN does.

If the above approach fails, try another tape drive.

20.13 FITSIN does not Recognise my FITS Tape

If you attempt to read a FITS magnetic tape with FITSIN, you might receive an error like this

```
% fitsin
% MT - Tape deck /@/dev/nrmt1h/ > /dev/nrmt3l
!! Object '/DEV/NRMT3L' not found.
!  DAT_FIND: Error finding a named component in an HDS structure.
!  /dev/nrmt3l: MAG__UNKDV, Unable to locate device in DEVDATASET
```

when you enter the device name. The magnetic-tape system uses an HDS file called the device dataset (DEVDATASET) to store the position of the tape between invocations of Starlink applications.

When FITSIN is given a name, the magnetic-tape system validates the name to check that it is a known device. There should be a `devdataset.sdf` file (within `/star/etc` at Starlink sites) containing a list of at least all the available drives at your site. What FITSIN is complaining about above, is that the device you have given is not included in the DEVDATASET file. Now this might be because you mistyped the device name, or that the particular device is not accessible on the particular machine, or because your computer manager has not maintained the DEVDATASET when a drive was added. You can look at the contents of the DEVDATASET with this command.

```
% hdstrace /star/etc/devdataset
```

Oh and one other point: make sure the tape is loaded in the drive. Yes this mistake has happened (not mentioning any names) and it is very hard to diagnose remotely.

20.14 It Used to Work. . . and Weird Errors

There is a class of error that arises when an HDS file is corrupted. The specific message will depend on the file concerned and where in the file the corruption occurred. The most likely reason for file corruption is breaking into a task at the wrong moment, or trying to write to a file at the same time.

If you want to process simultaneously from different sessions—say one interactive and another in batch—it is wise to redefine the environment variables `ADAM_USER`, and `AGI_USER` if you want graphics on the same machine. The environment variables should point to a separate existing directory for each additional session. This will keep the global and application parameters, and the graphics database separate for each session.

The way to look for corrupted HDS files is trace them. Assuming that $ADAM_USER and $AGI_USER are defined,

```
% hdstrace $ADAM_USER/GLOBALS full
% hdstrace $ADAM_USER/ardmask full
% hdstrace $AGI_USER/agi_cacvad full
```

traces the `GLOBALS` file, the application you were running when the weird error occurred (here ARDMASK), and the graphics database for machine `cacvad`. Once you have identified the problem file, delete it. If that proves to be the globals file, you might want to retain the output from HDSTRACE, so that you can restore their former values. Deleting the graphics database is something you should do regularly, so that's not a problem.

If you have been running KAPPA from ICL, you will need to check of the integrity of the monolith parameter file, instead the individual parameter file. It will be one of these depending on the type of task that failed: graphics, NDF components, or the rest (mostly image processing) corresponding to these three monolith interface files.

```
% hdstrace $ADAM_USER/kapview_mon full
% hdstrace $ADAM_USER/ndfpack_mon full
% hdstrace $ADAM_USER/kappa_mon full
```

If that doesn't cure the problem, send a log of the session demonstrating the problem to the Starlink Software support mailing list (`starlink@jiscmail.ac.uk`), and we shall endeavour to interpret it for you, and find out what's going wrong.

21 Custom KAPPA

21.1 Tasks

KAPPA applications can be modified to suit your particular requirements. Since this document is not a programmer's guide, instructions are not given here. Programmers should contact the author for details until a new Programmer's Guide appears to replace the old SUN/101, which *was* a good summary of Starlink infrastructure libraries and programming.

All the source files can be found in `/star/kappa/*.tar` on Starlink machines. The `/star` path may be different outside of Starlink, so check with your computer manager. There is a separate tar file for each KAPPA subroutine library (with a `_sub` suffix) and the interface files, with obvious names. The remaining files: the monolith routines, link scripts, include files, the help source, shell scripts, ICL procedures, and test data are in `kappa_source.tar`. There is also a Starlink standard `makefile` and `mk` script.

Many of the general-purpose subroutines which previously formed part of KAPPA have now been moved into a separate software item called KAPLIBS (see SUN/238). KAPPA itself now links against the libraries in KAPLIBS .

Here is a worked example. Suppose that you have _REAL-type datasets for which you want to compute statistics including the skewness and kurtosis. One way is to modify STATS. First to save typing define environment variables, say `STAR` and `KAPPA` and `KAPLIBS` to point to where the Starlink software, KAPPA and KAPLIBS source is stored. Next we extract the source files to change.

```
% setenv STAR /star
% setenv KAPPA /star/sources/kappa
% setenv KAPLIBS /star/sources/kaplibs
% tar xf $KAPPA/kappa_sub.tar stats.
% tar xf $KAPPA/kappa_ifls.tar stats.ifl
% tar xf $KAPLIBS/kapgen_sub.tar kpg1_statr.f
% tar xf $KAPLIBS/kapgen_sub.tar kpg1_stdsr.f
% tar xf $KAPPA/kappa_source.tar kappa_link_adam
```

We modify `kpg1_statr.f` to compute the additional statistics; `kpg1_stdsr.f` to list the statistics; `stats.f` to update the documentation, to use the revised argument lists of the subroutines, and to output the new statistics to parameters; and `stats.ifl` to add the output parameters. `kappa_link_adam` need not be modified, but it is needed during linking.

Next some soft links to include files need to be made.

```
% star_dev
% ndf_dev
% prm_dev
% par_dev
% kaplibs_dev
```

For some other application and subroutines, you can find what is needed by trying to compile them and see which include files the compiler cannot locate. You then invoke the appropriate

package definitions: *pkg*_dev, where *pkg* is the three-letter package abbreviation. Now compile the modified code. This is for OSF/1:

```
% f77 -O -c -nowarn stats.f kpg1_statr.f kpg1_stdsr.f
```

and this is for Solaris:

```
% f77 -O -PIC -c -w stats.f kpg1_statr.f kpg1_stdsr.f
```

The -nowarn and -w prevent warning messages appearing.

And this is for Linux:

```
% g77 -fno-second-underscore -O -c stats.f kpg1_statr.f kpg1_stdsr.f
```

Now link the task to produce a new stats executable.

```
% alink stats.o -o stats kpg1_statr.o kpg1_stdsr.o \
-L$STAR/lib `./kappa_link_adam`
```

If you want to use KAPPA subroutines for your own application here are words of warning: *the code may undergo alterations of subroutine name or argument lists, and those without a* pkg_ *prefix will either be replaced or renamed.* Therefore, you should copy the modules you need.

21.2 Parameters

If you don't like KAPPA's parameter defaults, or its choice of which parameters get prompted for and which get defaulted, then you can change them. Extract the interface file from /star/kappa/kappa_ifls.tar to your work directory and make the required modifications, and then recompile it. See SUN/115 on the meanings and possible values of the fieldnames, and how to recompile the interface file. If you use ICL, you'll need to modify a monolith interface file: $KAPPA_DIR/kappa_mon.ifl, kapview_mon.ifl or ndfpack_mon.ifl. Finally, you will need to specify a search path that includes the directory containing your modified interface file.

```
% setenv ADAM_IFL /home/scratch/dro/ifls:/usr/local/kappa
```

This asks Starlink programmes to look in /home/scratch/dro/ifls to find the interface file, and if there isn't one to look in /usr/local/kappa. If the interface file search is unsuccessful, the directory containing the executable is assumed. Thus if you've not created your own interface file for a task, you'll get the released version. Of course, once you have done this, the documentation in Appendix C will no longer be correct.

21.3 Commands

There is an easier method of tailoring KAPPA to your requirements. If you frequently use certain commands, especially those with a long list of keywords and fixed values, you can define some C-shell aliases or ICL symbols for the commands. Like the shell's `$HOME/.login`, ICL has a *login file*. (See "ICL for Unix" Appendix in SUN/144, or SSN/64 for details.) If you add symbols to this file, each time you activate ICL these abbreviations will be available to you without further typing. What you should do is to create a `login.icl` in a convenient directory, and assign the environment variable `ICL_LOGIN` to that directory in your `$HOME/.login` file.

It is possible to have several ICL login files, each for different work in different directories. Now to abbreviate a command you put a DEFSTRING entry into the ICL login file. For example,

```
DEFSTRING MYC{ON} CONTOUR CLEAR=F PENROT MODE=AU NCONT=7
```

defines `MYC` or `MYCO` or `MYCON` to run CONTOUR without clearing the screen, with pen rotation and seven equally spaced contour heights. The symbols are not limited to KAPPA. Indeed they can include shorthands for shell commands. For example,

```
DEFSTRING DA ls -al
```

would make DA produce a directory listing of all files with sizes and modification dates.

You can check what the current login files are as follows.

```
% printenv | grep ICL_LOGIN
```

For shell usage similar definitions can be made with aliases. For example,

```
% alias mycon contour clear=f penrot mode=au ncont=7
```

is the equivalent of the DEFSTRING above, except that in keeping with UNIX tradition the command is in lowercase, and the alias cannot be abbreviated.

22 Acknowledgments

Several people have contributed complete KAPPAprogrammes, or have upgraded earlier versions, or have written original code (which eventually became included in KAPPA after reworking). Mark Taylor and Rodney Warren-Smith both re-wrote some of the old IMAGE applications to use the NDF_ library. Rodney also supplied several other programmes, especially ones that now form the basis of NDFPACK. Other contributions have come from Alasdair Allan, Steven Beard, Wei Gong, Rhys Morris, Jo Murray, Grant Privett, and Richard Saxton. The original KAPPA was derived from Mark McCaughrean's RAPI2D and Ken Hartley's ASPIC Kernel, though little remains. Thanks also to Rodney Warren-Smith for permitting this document to include a few modified pages of his SUN/33 on NDF sections and co-ordinate systems; and to many useful suggestions from users and programmers over the years including Chris Clayton, Peter Draper, Jim Emerson, Horst Meyerdierks, Andy Scott, and Martin Shaw. Mike Lawden helped produce the quick-reference card.

23 Acknowledging this Software

Please acknowledge the use of this software in any publications arising from research in which it has played a significant rôle. Please also acknowledge the use of any other Starlink software in such publications. The following is suggested as a suitable form of words:

> *The authors acknowledge the use of the following software provided by the UK Starlink Project: KAPPA,... Starlink is run by CCLRC on behalf of PPARC.*

A Classified KAPPA commands

KAPPA applications may be classified in terms of their functions as follows.

A.1 DATA IMPORT & EXPORT

A.1.1 Image generation and input

CREFRAME	Generates a test two-dimensional NDF from a selection of several types.
FITSDIN	Reads a FITS disc file composed of simple, group or table objects.
FITSHEAD	Lists the headers of FITS files.
FITSIMP	Imports FITS information into an NDF extension.
FITSIN	Reads a FITS tape composed of simple, group or table files.
MATHS	Evaluates mathematical expressions applied to NDF data structures.
TRANDAT	Converts free-format data into an NDF.

A.1.2 Preparation for output

FITSEDIT	Edits the FITS extension of an NDF.
FITSEXP	Exports NDF-extension information into an NDF FITS extension.
FITSMOD	Edits an NDF FITS extension via a text file or parameters.
FITSTEXT	Creates an NDF FITS extension from a text file.
FITSWRITE	Writes a new keyword to the FITS extension.

A.2 DATA DISPLAY

A.2.1 Detail enhancement

CARPET	Creates a cube representing a carpet plot of an image.
COLCOMP	Produces a colour composite image from 1, 2 or 3 individual NDFs.
HISTEQ	Performs an histogram equalisation on an NDF.
LAPLACE	Performs a Laplacian convolution as an edge detector in a two-dimensional NDF.
SHADOW	Enhances edges in a two-dimensional NDF using a shadow effect.
THRESH	Edits an NDF such that array values below and above two thresholds take constant values.

A.2.2 Device selection

GDNAMES Shows which graphics devices are available.

GDSET Selects a current graphics device.

A.2.3 Display control

CURSOR Reports the co-ordinates of points selected using the cursor.

GDCLEAR Clears a graphics device and purges its database entries.

GDSTATE Shows the current status of a graphics device.

A.2.4 Graphics Database

PICBASE Selects the BASE picture from the graphics database.

PICCUR Uses a cursor to select the current picture.

PICDATA Selects the last DATA picture from the graphics database.

PICDEF Defines a new graphics-database FRAME picture or an array of FRAME pictures.

PICEMPTY Finds the first empty FRAME picture in the graphics database.

PICENTIRE Finds the first unobscured and unobscuring FRAME picture in the graphics database.

PICFRAME Selects the last FRAME picture from the graphics database.

PICGRID Creates an array of FRAME pictures.

PICIN Finds the attributes of a picture interior to the current picture.

PICLABEL Labels the current graphics-database picture.

PICLAST Selects the last picture from the graphics database.

PICLIST Lists the pictures in the graphics database for a device.

PICSEL Selects a graphics-database picture by its label.

PICTRANS Transforms co-ordinates between the current and BASE pictures.

PICVIS Finds the first unobscured FRAME picture in the graphics database.

PICXY Creates a new picture defined by co-ordinate bounds.

A.2.5 Lookup/Colour tables

LUTABLE	Manipulates an graphics device colour table.
LUTBGYRW	Loads the *BGYRW* lookup table.
LUTCOL	Loads the standard colour lookup table.
LUTCOLD	Loads the *cold* lookup table.
LUTCONT	Loads a lookup table to give the display the appearance of a contour plot.
LUTEDIT	Creates or edits an graphics device colour table.
LUTFC	Loads the standard false-colour lookup table.
LUTGREY	Loads the standard grey-scale lookup table.
LUTHEAT	Loads the *heat* lookup table.
LUTIKON	Loads the default *Ikon* lookup table.
LUTNEG	Loads the standard negative grey-scale lookup table.
LUTRAMPS	Loads the coloured-ramps lookup table.
LUTREAD	Loads an graphics device lookup table from an NDF.
LUTSAVE	Saves the current colour table of an graphics device in an NDF.
LUTSPEC	Loads a spectrum-like lookup table.
LUTVIEW	Draws a colour-table key.
LUTWARM	Loads the *warm* lookup table.
LUTZEBRA	Loads a pseudo-contour lookup table.

A.2.6 Output

ARDPLOT	Plots the boundaries of regions described in an ARD file over an existing picture.
CLINPLOT	Draws a spatial grid of line plots for an axis of a cube NDF.
CONTOUR	Contours a two-dimensional NDF.
DISPLAY	Displays a one- or two-dimensional NDF.
DRAWNORTH	Draws arrows parallel to the axes.
DRAWSIG	Draws $\pm n$ standard-deviation lines on a line plot.
ELPROF	Creates a radial or azimuthal profile of a two-dimensional image.
LINPLOT	Draws a line plot of the data values in a one-dimensional NDF.

LISTSHOW	Displays the positions stored in a positions list.
LOOK	Outputs the values of specified NDF pixels to the screen or a text file.
MLINPLOT	Draws a multi-line plot of the data values in a two-dimensional NDF.
OUTLINE	Draws the outline of a two-dimensional NDF.
SCATTER	Displays a scatter plot between data in two NDFs.
VECPLOT	Plots a two-dimensional vector map.

A.2.7 Palette

PALDEF	Loads the default palette to a colour table.
PALENTRY	Enters a colour into an graphics device's palette.
PALREAD	Fills the palette of a colour table from an NDF.
PALSAVE	Saves the current palette of a colour table to an NDF.

A.3 DATA MANIPULATION

A.3.1 Arithmetic

ADD	Adds two NDF data structures.
CADD	Adds a scalar to an NDF data structure.
CDIV	Divides an NDF by a scalar.
CMULT	Multiplies an NDF by a scalar.
CSUB	Subtracts a scalar from an NDF data structure.
CUMULVEC	Sums the values cumulatively in a one-dimensional NDF.
DIV	Divides one NDF data structure by another.
EXP10	Takes the base-10 exponential of each pixel of an NDF.
EXPE	Takes the exponential of each pixel of an NDF (base e).
EXPON	Takes the exponential (specified base) of each pixel of am NDF.
LOG10	Takes the base-10 logarithm of each pixel of an NDF.
LOGAR	Takes the logarithm of each pixel of an NDF (specified base).
LOGE	Takes the natural logarithm of each pixel of an NDF.
MAKESNR	Creates a signal-to-noise array from an NDF with Variance.
MATHS	Evaluates mathematical expressions applied to NDF data structures.

MULT	Multiplies two NDF data structures.
POW	Takes the specified power of each pixel of a data array.
SUB	Subtracts one NDF data structure from another.
TRIG	Performs a trigonometric transformation on an NDF.

A.3.2 Combination

CALPOL	Calculates polarisation parameters.
COLCOMP	Produces a colour composite image from 1, 2 or 3 individual NDFs.
COMPLEX	Converts between representations of complex data.
INTERLEAVE	Forms a higher-resolution NDF by interleaving a set of NDFs.
KSTEST	Compares data sets using the Kolmogorov-Smirnov test.
NORMALIZE	Normalises one NDF to a similar NDF by calculating a scale factor and zero difference.
WCSMOSAIC	Tiles a group of NDFs using World Co-ordinate System information.

A.3.3 Compression and expansion

CARPET	Creates a cube representing a carpet plot of an image.
CHANMAP	Creates a channel map from a cube NDF by compressing slices along a nominated axis
COLLAPSE	Reduces the number of axes in an NDF by collapsing it along a nominated axis.
COMPADD	Reduces the size of an NDF by adding values in rectangular boxes.
COMPAVE	Reduces the size of an NDF by averaging values in rectangular boxes.
COMPICK	Reduces the size of an NDF by picking equally spaced pixels.
INTERLEAVE	Forms a higher-resolution NDF by interleaving a set of NDFs.
NDFCOMPRESS	Compresses an NDF so that it occupies less disk space.
PIXDUPE	Expands an NDF by pixel duplication.
PLUCK	Plucks slices from an NDF at arbitrary positions.
REGRID	Uses an arbitrary mapping to regrid an NDF.
SQORST	Squashes or stretches an NDF.
WCSALIGN	Aligns a group of NDFs using WCS information.

A.3.4 Configuration change

CHAIN	Concatenates a series of vectorized NDFs.
FLIP	Reverses an NDF's pixels along a specified dimension.
MANIC	Converts all or part of an NDF from one dimensionality to another.
NDFCOPY	Copies an NDF (or NDF section) to a new location.
PERMAXES	Permutes the axes of an NDF.
PIXBIN	Places each pixel value in an input NDF into an output bin.
PLUCK	Plucks slices from an NDF at arbitrary positions.
PROFILE	Creates a one-dimensional profile through an *n*-dimensional NDF.
REGRID	Uses an arbitrary mapping to regrid an NDF.
RESHAPE	Reshapes an NDF, treating its arrays as vectors.
ROTATE	Rotates a two-dimensional NDF about its centre through any angle.
SETBOUND	Sets new bounds for an NDF.
SLIDE	Shifts pixels in an NDF by a given amount along each axis.
WCSSLIDE	Applies a translational correction to the WCS in an NDF.

A.3.5 Filtering

BLOCK	Smooths an NDF using an n-dimensional rectangular box filter.
CONVOLVE	Convolves a pair of one- or two-dimensional NDFs together.
FFCLEAN	Removes defects from a substantially flat one- or two-dimensional NDF.
FOURIER	Performs forward and inverse Fourier transforms of one- or two-dimensional NDFs.
GAUSMOOTH	Smooths a one- or two-dimensional image using a Gaussian filter.
LUCY	Performs a Richardson-Lucy deconvolution of a one- or two-dimensional array.
MEDIAN	Smooths a two-dimensional data array using a weighted median filter.
MEM2D	Performs a Maximum-Entropy deconvolution of a two-dimensional NDF.
ODDEVEN	Removes odd-even defects from a one-dimensional NDF.
WIENER	Applies a Wiener filter to a one- or two-dimensional array.

A.3.6 HDS components

ERASE	Erases an HDS object.
NATIVE	Converts an HDS object to native machine data representation.

A.3.7 NDF array components

NDFCOMPRESS	Compresses an NDF so that it occupies less disk space.
NDFCOPY	Copies an NDF (or NDF section) to a new location.
PERMAXES	Permutes the axes of an NDF.
QUALTOBAD	Assigns bad values to pixels with given qualities.
REMQUAL	Removes named qualities stored in an NDF QUALITY component.
SETBAD	Sets new bad-pixel flag values for an NDF.
SETBB	Sets a new value for the quality bad-bits mask of an NDF.
SETBOUND	Sets new bounds for an NDF.
SETORIGIN	Sets a new pixel origin for an NDF.
SETQUAL	Assigns a specified quality to selected pixels within an NDF.
SETTYPE	Sets a new numeric type for the DATA and VARIANCE components of an NDF.
SETVAR	Sets new values for the VARIANCE component of an NDF data structure.
SHOWQUAL	Displays the named qualities stored in an NDF QUALITY component.

A.3.8 NDF axis components

AXCONV	Expands spaced axes in an NDF into the primitive form.
AXLABEL	Sets a new label value for an axis within an NDF data structure.
AXUNITS	Sets a new units value for an axis within an NDF data structure.
PERMAXES	Permutes the axes of an NDF.
SETAXIS	Sets values for an axis array component within an NDF data structure.
SETNORM	Sets a new value for one or all of an NDF's axis-normalisation flags.

A.3.9 NDF character components

SETLABEL Sets a new label for an NDF data structure.

SETTITLE Sets a new title for an NDF data structure.

SETUNITS Sets a new units value for an NDF data structure.

A.3.10 NDF extensions

FITSEDIT Edits the FITS extension of an NDF.

FITSEXIST Inquires whether or not a keyword exists in a FITS extension.

FITSEXP Exports NDF-extension information into an NDF FITS extension.

FITSLIST Lists the FITS extension of an NDF.

FITSMOD Edits an NDF FITS extension via a text file or parameters.

FITSTEXT Creates an NDF FITS extension from a text file.

FITSVAL Reports the value of a keyword in the FITS extension.

FITSWRITE Writes a new keyword to the FITS extension.

SETEXT Manipulates the contents of a specified NDF extension.

SETSKY Stores WCS Information in an NDF.

A.3.11 NDF History

HISCOM Adds commentary to the history of an NDF.

HISLIST Lists NDF history records.

HISSET Sets the NDF history update mode.

A.3.12 NDF Provenance

PROVADD Stores provenance information in an NDF.

PROVMOD Modifies provenance information for an NDF.

PROVREM Removes selected provenance information from an NDF.

PROVSHOW Displays provenance information for an NDF.

A.3.13 NDF World Co-ordinate Systems

PERMAXES	Permutes the axes of an NDF.
WCSADD	Creates a Mapping and optionally adds a new co-ordinate Frame into the WCS component of an NDF.
WCSALIGN	Aligns a group of NDFs using WCS information.
WCSATTRIB	Manages attribute values associated with the WCS component of an NDF.
WCSCOPY	Copies WCS information from one NDF to another.
WCSFRAME	Changes the current co-ordinate Frame in the WCS component of an NDF.
WCSMOSAIC	Tiles a group of NDFs using World Co-ordinate System information.
WCSREMOVE	Removes co-ordinate Frames from the WCS component of an NDF.
WCSSHOW	Examines the internal structure of a WCS description.
WCSSLIDE	Applies a translational correction to the WCS in an NDF.
WCSTRAN	Transforms a position from one NDF co-ordinate Frame to another.

A.3.14 Pixel editing and masking

ARDGEN	Creates a text file describing selected regions of an image.
ARDMASK	Uses an ARD file to set some pixels of an NDF to be bad.
ARDPLOT	Plots the boundaries of regions described in an ARD file over an existing picture.
CHPIX	Replaces the values of selected pixels in an NDF.
COPYBAD	Copies the bad-pixel mask from one NDF to another.
ERRCLIP	Removes pixels with large errors from an NDF.
EXCLUDEBAD	Copies a two-dimensional NDF excluding any bad rows or columns.
FFCLEAN	Removes defects from a substantially flat one- or two-dimensional NDF.
FILLBAD	Removes regions of bad values from an NDF.
GLITCH	Replaces bad pixels in a two-dimensional image with the local median.
MFITTREND	Fits independent trends to data lines that are parallel to an axis.
MOCGEN	Creates a Multi-Order Coverage (MOC) map describing regions of an image.

NOMAGIC	Replaces all occurrences of magic-value pixels in an NDF array with a new value.
OUTSET	Sets pixels outside a specified circle in a two-dimensional NDF to a specified value.
PASTE	Pastes a series of NDFs upon each other.
REGIONMASK	Applies a mask to a region of an NDF.
RIFT	Adds a scalar to a section of an NDF data structure to correct rift-valley defects.
SEGMENT	Copies polygonal segments from one NDF to another.
SETMAGIC	Replaces all occurrences of a given value in an NDF array with the bad value.
SUBSTITUTE	Replaces all occurrences of a given value in an NDF array with another value.
THRESH	Edits an NDF such that array values below and above two thresholds take
ZAPLIN	Replaces regions in a two-dimensional NDF by bad values or by linear interpolation.

A.3.15 Polarimetry

CALPOL	Calculates polarisation parameters.

A.3.16 Resampling and transformations

ALIGN2D	Aligns a pair of two-dimensional NDFs by minimising the residuals between them.
PIXBIN	Places each pixel value in an input NDF into an output bin.
PLUCK	Plucks slices from an NDF at arbitrary positions.
REGRID	Uses an arbitrary mapping to regrid an NDF.
WCSALIGN	Aligns a group of NDFs using WCS information.
WCSMOSAIC	Tiles a group of NDFs using World Co-ordinate System information.

A.3.17 Surface and vector fitting

FITSURFACE	Fits a polynomial surface to two-dimensional data array.
MAKESURFACE	Creates a two-dimensional NDF from the coefficients of a polynomial surface.
MFITTREND	Fits independent trends to data lines that are parallel to an axis.
SURFIT	Fits a polynomial or spline surface to a two-dimensional data array using blocking.

A.4 DATA ANALYSIS

A.4.1 Statistics

APERADD	Derives statistics of pixels within a specified aperture of an NDF.
HISTAT	Computes ordered statistics for an NDF's pixels using an histogram.
HISTOGRAM	Computes an histogram of an NDF's values.
MSTATS	Does cumulative statistics over a sequence of NDFs.
NUMB	Counts the number of elements of an NDF with values or absolute values above or below a threshold.
STATS	Computes simple statistics for an NDF's pixels.

A.4.2 Other

BEAMFIT	Fits beam features in a two-dimensional NDF.
CENTROID	Finds the centroids of star-like features in an NDF.
NORMALIZE	Normalises one NDF to a similar NDF by calculating a scale factor and zero-point difference.
PSF	Determines the parameters of a model star profile by fitting star images in a two-dimensional NDF.
SURFIT	Fits a polynomial or spline surface to a two-dimensional data array.

A.5 SCRIPTING TOOLS

CALC	Evaluates a mathematical expression.
CONFIGECHO	Displays a named parameter from a group of configuration parameters.
NDFECHO	Expands a group expression into a list of explicit NDF names.
PARGET	Obtains the value or values of an application parameter.

A.6 INQUIRIES & STATUS

GLOBALS	Displays the values of the KAPPA global parameters.
FITSEXIST	Inquires whether or not a keyword exists in a FITS extension.
FITSLIST	Lists the FITS extension of an NDF.
FITSVAL	Reports the value of a keyword in the FITS extension.
NDFCOMPARE	Compares a pair of NDFs for equivalence.
NDFTRACE	Displays the attributes of an NDF data structure.
NOGLOBALS	Resets the KAPPA global parameters.

A.7 MISCELLANEOUS

COMPLEX	Converts between representations of complex data.
KAPHELP	Gives help about KAPPA.
LISTMAKE	Creates a catalogue holding a positions list.
KAPVERSION	Checks the version number of the installed package.

B Quotas to run KAPPA

No special quotas are needed to run KAPPA. If you have large datasets you might need to increase the datasize limit in the C-shell.

```
% limit datasize 65336
```

sets the maximum size of a data file to 64 megabytes. To list the current values use the **limit** command without any arguments.

C Specifications of KAPPA applications

C.1 Explanatory Notes

The specification of parameters has the following format.

```
name  =  type (access)
   description
```

This format also includes a *Usage* entry. This shows how the application is invoked from the command line. It lists the positional parameters in order followed by any prompted keyword parameters using a "KEYWORD=?" syntax. Defaulted keyword parameters do not appear. Positional parameters that are normally defaulted are indicated by being enclosed in square brackets. Keyword (*i.e.* not positional) parameters are needed where the number of parameters are large, and usually occur because they depend on the value of another parameter. These are denoted by a curly brace; the parameters on each line are related, and each line is mutually exclusive. An example should clarify.

```
contour ndf [comp] mode ncont [key] [device]   { low =? high =?
                                               { percentiles =?
                                               { sigmas =?
                                                 mode
```

NDF, COMP, MODE, NCONT, KEY, DEVICE, and SMOOTHING are all positional parameters. Only NDF, MODE, and NCONT would be prompted if not given on the command line. The remaining parameters depend on the value of MODE. If the mode is to nominate a list of contour heights, HEIGHTS will be needed (MODE = `"Free"`); alternatively, if the mode requires a start height and spacing between contours FIRSTCNT and STEPCNT should be specified (MODE = `"Linear"` or `"Magnitude"`). Note that there are other modes that do not require additional information, and hence no more parameters.

There is also an *Examples* section. This shows how to run the application from the command line. More often you'll enter the command name and just some of the parameters, and be prompted for the rest. *Note that the examples are the strings expected by the tasks.* They are operating-system neutral as KAPPA has run on several different operating systems. *UNIX shells or operating-system command languages will often interpret as special characters some or all of* `[] ()\^~"'$*?` *that may form part of the* KAPPA *command-line syntax. So in practice you should escape any such special characters that appear in these examples, as appropriate to your command language or shell.* For instance, from the C-shell the fourth example of COMPAVE could be written like the following.

```
compave cosmos galaxy '[4,3]' weight title='"COSMOS compressed"'
compave cosmos galaxy \[4,3\] weight title=\"COSMOS compressed\"
```

Backslash escapes individual special characters, whereas quotes placed around text escape all occurrences of special characters within the quotes.

Some parameters will only be used when another parameter has a certain value or mode. These are indicated by the name of the mode in parentheses at the end of the parameter description,

but before any default, *e.g.* Parameter DEVICE in CENTROID is only relevant when Parameter MODE is `"Cursor"`.

`%name` means the value of parameter *name*.

The description entry has a notation scheme to indicate normally defaulted parameters, *i.e.* those for which there will be no prompt. For such parameters a matching pair of square brackets (`[]`) terminates the description. The content between the brackets mean

`[]` Empty brackets means that the default is created dynamically by the application, and may depend on the values of other parameters. Therefore, the default cannot be given explicitly.

`[,]` As above, but there are two default values that are created dynamically.

[**default**] Occasionally, a description of the default is given in normal type, *e.g.* the size of the plotting region in a graphics application, where the exact default values depend on the device chosen.

`[default]` If the brackets contain a value in the teletype typeface, this is the explicit default value.

ADD
Adds two NDF data structures

Description:

The routine adds two NDF data structures pixel-by-pixel to produce a new NDF.

Usage:

```
add in1 in2 out
```

Parameters:

IN1 = NDF (Read)
First NDF to be added.

IN2 = NDF (Read)
Second NDF to be added.

OUT = NDF (Write)
Output NDF to contain the sum of the two input NDFs.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN1 to be used instead. `[!]`

Examples:

```
add a b c
```

This adds the NDF called b to the NDF called a, to make the NDF called c. NDF c inherits its title from a.

```
add out=c in1=a in2=b title="Co-added image"
```

This adds the NDF called b to the NDF called a, to make the NDF called c. NDF c has the title `"Co-added image"`.

Notes:

If the two input NDFs have different pixel-index bounds, then they will be trimmed to match before being added. An error will result if they have no pixels in common.

Related Applications :

KAPPA: CADD, CDIV, CMULT, CSUB, DIV, MATHS, MULT, SUB.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.

- The UNITS component is propagated only if it has the same value in both input NDFs.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Huge NDFs are supported.

ALIGN2D
Aligns a pair of two-dimensional NDFs by minimising the residuals between them.

Description:

This application attempts to align a two-dimensional input NDF with a two-dimensional reference NDF in pixel co-ordinates, using an affine transformation of the form:

$$X_{\mathrm{in}} = C_1 + C_2 X_{\mathrm{ref}} + C_3 Y_{\mathrm{ref}}$$

$$Y_{\mathrm{in}} = C_4 + C_5 X_{\mathrm{ref}} + C_6 Y_{\mathrm{ref}}$$

where (X_{in},Y_{in}) are pixel co-ordinates in the input NDF, and (X_{ref},Y_{ref}) are pixel co-ordinates in the reference NDF. The coefficient values (C_1–C_6) are determined by doing a least-squares fit that minimises the sum of the squared residuals between the reference NDF and the transformed input NDF. If variance information is present in either NDF, it is used to determine the SNR of each pixel which is used to weight the residuals within the fit, so that noisy data values have less effect on the fit. The best fit coefficients are displayed on the screen and written to an output parameter. Optionally, the transformation may be applied to the input NDF to create an output NDF (see Parameter OUT). It is possible to restrict the transformation in order to prevent shear, rotation, scaling, *etc*. (see Parameter FORM).

It is possible to exclude from the fitting process areas of the input NDF that are poorly correlated with the corresponding areas in the reference NDF (*e.g.* flat background areas that contain only noise). See Parameter CORLIMIT.

Usage:

```
align2d in ref out
```

Parameters:

BOX = _INTEGER (Read)

The box size, in pixels, over which to calculate the correlation coefficient between the input and reference images. This should be set to an estimate of the maximum expected shift between the two images, but should not be less than typical size of features within the two images. See also Parameter CORLIMIT. [5]

CONSERVE = _LOGICAL (Read)

If set TRUE, then the output pixel values will be scaled in such a way as to preserve the total data value in a feature on the sky. The scaling factor is the ratio of the output pixel size to the input pixel size. This option can only be used if the Mapping is successfully approximated by one or more linear transformations. Thus an error will be reported if it used when the TOL parameter is set to zero (which stops the use of

linear approximations), or if the Mapping is too non-linear to be approximated by a piece-wise linear transformation. The ratio of output to input pixel size is evaluated once for each panel of the piece-wise linear approximation to the Mapping, and is assumed to be constant for all output pixels in the panel. This parameter is ignored if the NORM parameter is set `FALSE`. `[TRUE]`

CORLIMIT = _REAL (Read)

If CORLIMIT is not null (`!`), each pixel in the input image is checked to see if the pixel values in its locality are well correlated with the corresponding locality in the reference image. The input pixel is excluded from the fitting process if the local correlation is below CORLIMIT. The supplied value should be between zero and 1.0. The size of the locality used around each input pixel is given by Parameter BOX. `[!]`

In addition, if a value is supplied for CORLIMIT, the input and reference pixel values that pass the above check are scaled so that they have a mean value of zero and a standard deviation of unity before being used in the fitting process. `[!]`

FITVALS = _LOGICAL (Read)

If `TRUE`, the fitting process will adjust the scale and offset of the input data values, in addition to the geometric position of the the input values, in order to minimise the sum of the squared residuals. `[FALSE]`

FORM = _INTEGER (Read)

The form of the affine transformation to use:

- `0` — full unrestricted six-coefficient fit;
- `1` — shift, rotation and a common X/Y scale but no shear;
- `2` — shift and rotation but no scale or shear; or
- `3` — shift but not rotation, scale or shear.

`[0]`

IN = NDF (Read)

NDF to be transformed.

METHOD = LITERAL (Read)

The method to use when sampling the input pixel values (if resampling), or dividing an input pixel value between a group of neighbouring output pixels (if rebinning). For details of these schemes, see the descriptions of routines AST_RESAMPLEx and AST_REBINSEQx in SUN/210. METHOD can take the following values.

- `"Linear"` — When resampling, the output pixel values are calculated by bi-linear interpolation among the four nearest pixels values in the input NDF. When rebinning, the input pixel value is divided bi-linearly between the four nearest output pixels. Produces smoother output NDFs than the nearest-neighbour scheme, but is marginally slower.
- `"Nearest"` — When resampling, the output pixel values are assigned the value of the single nearest input pixel. When rebinning, the input pixel value is assigned completely to the single nearest output pixel.
- `"Sinc"` — Uses the $\mathrm{sinc}(\pi x)$ kernel, where x is the pixel offset from the interpolation point (resampling) or transformed input pixel centre (rebinning), and $\mathrm{sinc}(z) = \sin(z)/z$. Use of this scheme is not recommended.

- `"SincSinc"` — Uses the $\mathrm{sinc}(\pi x)\mathrm{sinc}(k\pi x)$ A valuable general-purpose scheme, intermediate in its visual effect on NDFs between the bi-linear and nearest-neighbour schemes.
- `"SincCos"` — Uses the $\mathrm{sinc}(\pi x)\cos(k\pi x)$ kernel. Gives similar results to the `"SincSinc"` scheme.
- `"SincGauss"` — Uses the $\mathrm{sinc}(\pi x)e^{-kx^2}$ kernel. Good results can be obtained by matching the FWHM of the envelope function to the point-spread function of the input data (see Parameter PARAMS).
- `"Somb"` — Uses the $\mathrm{somb}(\pi x)$ kernel, where x is the pixel offset from the interpolation point (resampling), or transformed input pixel centre (rebinning), and $\mathrm{somb}(z) = 2 * J_1(z)/z$. J_1 is the first-order Bessel function of the first kind. This scheme is similar to the `"Sinc"` scheme.
- `"SombCos"` — Uses the $\mathrm{somb}(\pi x)\cos(k\pi x)$ kernel. This scheme is similar to the `"SincCos"` scheme.
- `"Gauss"` — Uses the e^{-kx^2} kernel. The FWHM of the Gaussian is given by Parameter PARAMS(2), and the point at which to truncate the Gaussian to zero is given by Parameter PARAMS(1).
- `"BlockAve"` — Block averaging over all pixels in the surrounding N-dimensional cube. This option is only available when resampling (*i.e.* if REBIN is set to `FALSE`).

All methods propagate variances from input to output, but the variance estimates produced by interpolation schemes other than nearest neighbour need to be treated with care since the spatial smoothing produced by these methods introduces correlations in the variance estimates. Also, the degree of smoothing produced varies across the NDF. This is because a sample taken at a pixel centre will have no contributions from the neighbouring pixels, whereas a sample taken at the corner of a pixel will have equal contributions from all four neighbouring pixels, resulting in greater smoothing and lower noise. This effect can produce complex Moiré patterns in the output variance estimates, resulting from the interference of the spatial frequencies in the sample positions and in the pixel-centre positions. For these reasons, if you want to use the output variances, you are generally safer using nearest-neighbour interpolation. The initial default is `"Nearest"`. [current value]

NORM = _LOGICAL (Read)

In general, each output pixel contains contributions from multiple input pixel values, and the number of input pixels contributing to each output pixel will vary from pixel to pixel. If NORM is set `TRUE` (the default), then each output value is normalised by dividing it by the number of contributing input pixels, resulting in each output value being the weighted mean of the contributing input values. However, if NORM is set `FALSE`, this normalisation is not applied. See also Parameter CONSERVE. `[TRUE]`

OUT = NDF (Write)

An optional output NDF to contain a copy of IN aligned with OUT. No output is created if null (`!`) is supplied. If FITVALS is `TRUE`, the output data values will be scaled so that they have the same normalisation as the reference values.

PARAMS(2) = _DOUBLE (Read)

An optional array which consists of additional parameters required by the Sinc, SincSinc, SincCos, SincGauss, Somb, SombCos, and Gauss methods.

PARAMS(1) is required by all the above schemes. It is used to specify how many pixels are to contribute to the interpolated result on either side of the interpolation

or binning point in each dimension. Typically, a value of 2 is appropriate and the minimum allowed value is 1 (i.e. one pixel on each side). A value of zero or fewer indicates that a suitable number of pixels should be calculated automatically. [0]

PARAMS(2) is required only by the SombCos, Gauss, SincSinc, SincCos, and SincGauss schemes. For the SombCos, SincSinc, and SincCos schemes, it specifies the number of pixels at which the envelope of the function goes to zero. The minimum value is 1.0, and the run-time default value is 2.0. For the Gauss and SincGauss scheme, it specifies the full-width at half-maximum (FWHM) of the Gaussian envelope measured in output pixels. The minimum value is 0.1, and the run-time default is 1.0. On astronomical images and spectra, good results are often obtained by approximately matching the FWHM of the envelope function, given by PARAMS(2), to the point-spread function of the input data. []

REBIN = _LOGICAL (Read)

Determines the algorithm used to calculate the output pixel values. If a TRUE value is given, a rebinning algorithm is used. Otherwise, a resampling algorithm is used. See the "Choice of Algorithm" topic below. [current value]

REF = NDF (Read)

NDF to be used as a refernece.

TOL = _DOUBLE (Read)

The maximum tolerable geometrical distortion that may be introduced as a result of approximating non-linear Mappings by a set of piece-wise linear transforms. Both algorithms approximate non-linear co-ordinate transformations in order to improve performance, and this parameter controls how inaccurate the resulting approximation is allowed to be, as a displacement in pixels of the input NDF. A value of zero will ensure that no such approximation is done, at the expense of increasing execution time. [0.05]

WLIM = _REAL (Read)

This parameter is only used if REBIN is set TRUE. It specifies the minimum number of good pixels which must contribute to an output pixel for the output pixel to be valid. Note, fractional values are allowed. A null (!) value causes a very small positive value to be used resulting in output pixels being set bad only if they receive no significant contribution from any input pixel. [!]

Results Parameters:

RMS = _DOUBLE (Write)

An output parameter to which is written the RMS residual between the aligned data and the reference data.

TR(6) = _DOUBLE (Write)

An output parameter to which are written the coefficients of the fit. If FITVALS is TRUE, then this will include the scale and offset (written to the seventh and eighth entries).

Examples:

```
align2d my_data orionA my_corrected form=2
```

Aligns the two-dimensional NDF called my_data with the two-dimensional NDF called orionA, putting the aligned image in a new NDF called my_corrected. The transformation is restricted to a shift of origin and a rotation.

Related Applications :

KAPPA: WCSALIGN.

Implementation Status:

- This routine correctly processes the DATA, VARIANCE, WCS, LABEL, TITLE, and UNITS components of an NDF data structure.
- All non-complex numeric data types can be handled.

Choice of Algorithm :

The algorithm used to produce the output image is determined by the REBIN parameter, and is based either on resampling the output image or rebinning the input image.

The resampling algorithm steps through every pixel in the output image, sampling the input image at the corresponding position and storing the sampled input value in the output pixel. The method used for sampling the input image is determined by the METHOD parameter. The rebinning algorithm steps through every pixel in the input image, dividing the input pixel value between a group of neighbouring output pixels, incrementing these output pixel values by their allocated share of the input pixel value, and finally normalising each output value by the total number of contributing input values. The way in which the input sample is divided between the output pixels is determined by the METHOD parameter.

Both algorithms produce an output in which the each pixel value is the weighted mean of the near-by input values, and so do not alter the mean pixel values associated with a source, even if the pixel size changes. Thus the total data sum in a source will change if the input and output pixel sizes differ. However, if the CONSERVE parameter is set `TRUE`, the output values are scaled by the ratio of the output to input pixel size, so that the total data sum in a source is preserved.

A difference between resampling and rebinning is that resampling guarantees to fill the output image with good pixel values (assuming the input image is filled with good input pixel values), whereas holes can be left by the rebinning algorithm if the output image has smaller pixels than the input image. Such holes occur at output pixels which receive no contributions from any input pixels, and will be filled with the value zero in the output image. If this problem occurs the solution is probably to change the width of the pixel spreading function by assigning a larger value to PARAMS(1) and/or PARAMS(2) (depending on the specific METHOD value being used).

Both algorithms have the capability to introduce artefacts into the output image. These have various causes described below.

- Particularly sharp features in the input can cause rings around the corresponding features in the output image. This can be minimised by suitable settings for the METHOD and PARAMS parameters. In general such rings can be minimised by using a wider interpolation kernel (if resampling) or spreading function (if rebinning), at the cost of degraded resolution.

- The approximation of the Mapping using a piece-wise linear transformation (controlled by Parameter TOL) can produce artefacts at the joints between the panels of the approximation. These can occur when using the rebinning algorithm, or when using the resampling algorithm with CONSERVE set to `TRUE`. They are caused by the discontinuities between the adjacent panels of the approximation, and can be minimised by reducing the value assigned to the TOL parameter.

APERADD
Integrates pixel values within an aperture of an NDF

Description:

This routine displays statistics for pixels that lie within a specified aperture of an NDF . The aperture can either be circular (specified by Parameters CENTRE and DIAM), or arbitrary (specified by Parameter ARDFILE). If the aperture is specified using Parameters CENTRE and DIAM, then it must be either one- or two-dimensional.

The following statistics are displayed:

- The total number of pixels within the aperture
- The number of good pixels within the aperture
- The total data sum within the aperture
- The standard deviation on the total data sum (that is, the square root of the sum of the individual pixel variances)
- The mean pixel value within the aperture
- The standard deviation on the mean pixel value (that is, the standard deviation on the total data sum divided by the number of values)
- The standard deviation of the pixel values within the aperture

If individual pixel variances are not available within the input NDF (*i.e.* if it has no VARIANCE component), then each pixel is assumed to have a constant variance equal to the variance of the pixel values within the aperture. There is an option to weight pixels so that pixels with larger variances are given less weight (see Parameter WEIGHT). The statistics are displayed on the screen and written to output parameters. They may also be written to a log file.

A pixel is included if its centre is within the aperture, and is not included otherwise. This simple approach may not be suitable for accurate aperture photometry, especially where the aperture diameter is less than about ten times the pixel size. A specialist photometry package should be used if accuracy, rather than speed, is paramount.

Usage:

```
aperadd ndf centre diam
```

Parameters:

ARDFILE = FILENAME (Read)

The name of an ARD file containing a description of the aperture. This allows apertures of almost any shape to be used. If a null (`!`) value is supplied then the aperture is assumed to be circular with centre and diameter given by Parameters CENTRE and DIAM. ARD files can be created either 'by hand' using an editor, or using a specialist application such as ARDGEN.

The co-ordinate system in which positions within the ARD file are given should be indicated by including suitable COFRAME or WCS statements within the file (see SUN/183), but will default to pixel co-ordinates in the absence of any such statements. For instance, starting the file with a line containing the text `"COFRAME(SKY,System=FK5)"` would indicate that positions are specified in RA/DEC (FK5,J2000). The statement `"COFRAME(PIXEL)"` indicates explicitly that positions are specified in pixel co-ordinates. `[!]`

CENTRE = LITERAL (Read)
The co-ordinates of the centre of the circular aperture. Only used if Parameter ARDFILE is set to null. The position must be given in the current co-ordinate Frame of the NDF (supplying a colon `":"` will display details of the current co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas. See also Parameter USEAXIS. The current co-ordinate Frame can be changed using application WCSFRAME.

DIAM = LITERAL (Read)
The diameter of the circular aperture. Only used if Parameter ARDFILE is set to null. If the current co-ordinate Frame of the NDF is a SKY Frame (*e.g.* RA and DEC), then the value should be supplied as an increment of celestial latitude (*e.g.* DEC). Thus, `"10.2"` means 10.2 degrees, `"0:30"` would mean 30 arcminutes, and `"0:0:1"` would mean 1 arcsecond. If the current co-ordinate Frame is not a SKY Frame, then the diameter should be specified as an increment along Axis 1 of the current co-ordinate Frame. Thus, if the current Frame is PIXEL, the value should be given simply as a number of pixels.

LOGFILE = FILENAME (Read)
Name of the text file to log the results. If null, there will be no logging. Note this is intended for the human reader and is not intended for passing to other applications. `[!]`

MASK = NDF (Write)
An output NDF containing the pixel mask used to evaluate the reported statistics. The NDF will contain a positive integer value for pixels that are included in the statistics, and bad values for all other pixels. The pixel bounds of the NDF will be the smallest needed to encompass all used pixels. `[!]`

MEAN = _DOUBLE (Write)
The mean of the pixel values within the aperture.

NDF = NDF (Read)
The input NDF.

NGOOD = _INTEGER (Write)
The number of good pixels within the aperture.

NUMPIX = _INTEGER (Write)
The total number of pixels within the aperture.

SIGMA = _DOUBLE (Write)
The standard deviation of the pixel values within the aperture.

SIGMEAN = _DOUBLE (Write)
The standard deviation on the mean pixel value. If variances are available this is the RMS value of the standard deviations associated with each included pixel value. If

variances are not available, it is the standard deviation of the pixel values divided by the square root of the number of good pixels in the aperture.

SIGTOTAL = _DOUBLE (Write)
The standard deviation on the total data sum. Only created if variances are available this is the RMS value of the standard deviations associated with each included pixel value. If variances are not available, it is the standard deviation of the pixel values divided by the square root of the number of good pixels in the aperture.

TOTAL = _DOUBLE (Write)
The total of the pixel values within the aperture.

USEAXIS = GROUP (Read)
USEAXIS is only accessed if the current co-ordinate Frame of the NDF has too many axes. A group of strings should be supplied specifying the axes which are to be used when specifying the aperture using Parameters ARDFILE, CENTRE, and DIAM. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (`!`) value is supplied, the axes with the same indices as the two used pixel axes within the NDF are used. `[!]`

WEIGHT = _LOGICAL (Read)
If a `TRUE` value is supplied, and the input NDF has a VARIANCE component, then pixels with larger variances will be given smaller weight in the statistics. The weight associated with each pixel is proportional to the reciprocal of its variance. The constant of proportionality is chosen so that the mean weight is unity. The pixel value and pixel variance are multiplied by the pixels weight before being used to calculate the statistics. The calculation of the statistics remains unchanged in all other respects. `[FALSE]`

Examples:

```
aperadd neb1 "13.5,201.3" 20
```

This calculates the statistics of the pixels within a circular aperture of NDF neb1. Assuming the current co-ordinate Frame of neb1 is PIXEL, the aperture is centred at pixel co-ordinates (13.5, 201.3) and has a diameter of 20 pixels.

```
aperadd neb1 "15:23:43.2 -22:23:34.2" "10:0"
```

This also calculates the statistics of the pixels within a circular aperture of NDF neb1. Assuming the current co-ordinate Frame of neb1 is a SKY Frame describing RA and DEC, the aperture is centred at RA 15:23:43.2 and DEC -22:23:34.2, and has a diameter of 10 arcminutes.

```
aperadd ndf=neb1 ardfile=outline.dat logfile=obj1
```

This calculates the statistics of the pixels within an aperture of NDF neb1 described within the file `outline.dat`. The file contains an ARD description of the required aperture. The results are written to the log file `obj1`.

Notes:

- The statistics are not displayed on the screen when the message filter environment variable MSG_FILTER is set to `QUIET`. The creation of output parameters and the log file is unaffected by MSG_FILTER.

ASCII-region-definition Descriptors :

The ARD file may be created by ARDGEN or written manually. In the latter case consult SUN/183 for full details of the ARD descriptors and syntax; however, much may be learnt from looking at the ARD files created by ARDGEN and the ARDGEN documentation. There is also a in Section 15.1.1.

Related Applications :

KAPPA: STATS, MSTATS, ARDGEN, ARDMASK, ARDPLOT, WCSFRAME.

Implementation Status:

- This routine correctly processes the WCS, AXIS, DATA, and VARIANCE components of an NDF data structure.
- Processing of bad pixels and automatic quality masking are supported.
- Bad pixels and quality masking are supported.
- All non-complex numeric data types can be handled.

ARDGEN
Creates a text file describing selected regions of an image

Description:

This is an interactive tool for selecting regions of a displayed image using a cursor, and then storing a description of the selected regions in a text file in the form of an 'ARD Description' (see SUN/183). This text file may subsequently be used in conjunction with packages such as CCDPACK or ESP.

The application initially obtains a value for the SHAPE parameter and then allows you to identify either one or many regions of the specified shape, dependent on the value of Parameter STARTUP. When the required regions have been identified, a value is obtained for Parameter OPTION, and that value determines what happens next. Options include obtaining further regions, changing the current region shape, listing the currently defined regions, leaving the application, *etc*. Once the selected action has been performed, another value is obtained for OPTION, and this continues until you choose to leave the application.

Instructions on the use of the cursor are displayed when the application is run. The points required to define a region of the requested shape are described whenever the current region shape is changed using Parameter SHAPE. Once the points required to define a region have been given an outline of the entire region is drawn on the graphics device using the pen specified by Parameter PALNUM.

In the absence of any other information, subsequent application will use the union (*i.e.* the logical OR) of all the defined regions. However, regions can be combined in other ways using the COMBINE option (see Parameter OPTION). For instance, two regions originally defined using the cursor could be replaced by their region of intersection (logical AND), or a single region could be replaced by its own exterior (logical NOT). Other operators can also be used (see Parameter OPERATOR).

Usage:

```
ardgen ardout shape option [device] [startup] [palnum] [poicol]
    { operands=?  operator=?
    { regions=?
  option
```

Parameters:

ARDOUT = FILENAME (Write)
Name of the text file in which to store the description of the selected regions.

DEVICE = DEVICE (Read)
The graphics device on which the regions are to be selected. [Current graphics device]

OPERANDS() = _INTEGER (Read)
A pair of indices for the regions which are to be combined together using the operator specified by Parameter OPERATOR. If the operator is "NOT", then only one region

index need be supplied. Region indices are displayed by the "List" option (see Parameter OPTION).

OPERATOR = LITERAL (Read)

The operator to use when combining two regions into a single region. The pixels included in the resulting region depend on which of the following operators is selected.

- "AND" — Pixels are included if they are in both of the regions specified by Parameter OPERANDS.
- "EQV" — Pixels are included if they are in both or neither of the regions specified by Parameter OPERANDS.
- "NOT" — Pixels are included if they are not inside the region specified by Parameter OPERANDS.
- "OR" — Pixels are included if they are in either of the regions specified by Parameter OPERANDS. Note, an OR operator is implicitly assumed to exist between each pair of adjacent regions unless some other operator is specified.
- "XOR" — Pixels are included if they are in one, but not both, of the regions specified by Parameter OPERANDS.

OPTION= LITERAL (Read)

A value for this parameter is obtained when you choose to end cursor input (by pressing the relevant button as described when the application starts up). It determines what to do next. The following options are available:

- "Combine" — Combine two previously defined regions into a single region using a Boolean operator, or invert a previously defined region using a Boolean .NOT. operator. See Parameters OPERANDS and OPERATOR. The original regions are deleted and the new combined (or inverted) region is added to the end of the list of defined regions.
- "Delete" — Delete previously defined regions, see Parameter REGIONS.
- "Draw" — Draw the outline of the union of one or more previously defined regions, see Parameter REGIONS.
- "Exit" — Write out the currently defined regions to a text file and exit the application.
- "List" — List the textual descriptions of the currently defined regions on the screen. Each region is described by an index value, a *keyword* corresponding to the shape, and various arguments describing the extent and position of the shape. These arguments are described in the "Notes" section below.
- "Multi" — The cursor is displayed and you can then identify multiple regions of the current shape, without being re-prompted for OPTION after each one. These regions are added to the end of the list of currently defined regions. If the current shape is "Polygon", "Frame" or "Whole" (see Parameter SHAPE) then multiple regions cannot be defined and the selected option automatically reverts to "Single".
- "Single" — The cursor is displayed and you can then identify a single region of the current shape. You are re-prompted for Parameter OPTION once you

have defined the region. The identified region is added to the end of the list of currently defined regions.

- `"Shape"` — Change the shape of the regions created by the `"Single"` and `"Multi"` options. This causes a new value for Parameter SHAPE to be obtained.
- `"Style"` — Change the drawing style by providing a new value for Parameter STYLE.
- `"Quit"` — Quit the application without saving the currently defined regions.
- `"Undo"` — Undo the changes made to the list of ARD regions by the previous option. Note, the undo list can contain upto 30 entries. Entries are only stored for options which actually produce a change in the list of regions.

REGIONS() = LITERAL (Read)

The list of regions to be deleted or drawn. Regions are numbered consecutively from 1 and can be listed using the `"List"` option (see Parameter OPTION). Single regions or a set of adjacent regions may be specified, *e.g.* assigning `[4,6-9,12,14-16]` will delete regions 4,6,7,8,9,12,14,15,16. (Note that the brackets are required to distinguish this array of characters from a single string including commas. The brackets are unnecessary when there is only one item.) The numbers need not be in ascending order.

If you wish to delete or draw all the regions enter the wildcard `*`. For instance, `5-*` will delete or draw from 5 to the last region.

SHAPE = LITERAL (Read)

The shape of the regions to be defined using the cursor. After selecting a new shape, you are immediately requested to identify multiple regions as if `"Multi"` had been specified for Parameter OPTION. The currently available shapes are listed below.

- `"Box"` — A rectangular box with sides parallel to the co-ordinate axes, defined by its centre and one of its corners.
- `"Circle"` — A circle, defined by its centre and radius.
- `"Column"` — A single value on Axis 1, spanning all values on Axis 2.
- `"Ellipse"` — An ellipse, defined by its centre, one end of the major axis, and one other point which can be anywhere on the ellipse.
- `"Frame"` — The whole image excluding a border of constant width, defined by a single point on the frame.
- `"Point"` — A single pixel.
- `"Polygon"` — Any general polygonal region, defined by up to 200 vertices.
- `"Rectangle"` — A rectangular box with sides parallel to the co-ordinate axes, defined by a pair of diagonally opposite corners.
- `"Rotbox"` — A rotated box, defined by both ends of an edge, and one point on the opposite edge.
- `"Row"` — A single value on Axis 2, spanning all values on Axis 1.
- `"Whole"` — The whole of the displayed image.

STARTUP = LITERAL (Read)

Determines if the application starts up in `"Multi"` or `"Single"` mode (see Parameter OPTION). `["Multi"]`

UNDO = _LOGICAL (Read)
Used to confirm that it is OK to proceed with an "Undo" option. The consequences of proceeding are described before the parameter is obtained.

Examples:

```
ardgen extract.txt circle exit startup=single
```

This example allows you to create a text file (extract.txt) describing a single circular region of the image displayed on the current graphics device. The application immediately exits after the region has been identified. This example may be useful in scripts or command procedures since there is no prompting.

Notes:

- An image must previously have been displayed on the graphics device.
- The arguments for the textual description of each shape are as follows :
  - "Box" — The co-ordinates of the centre, followed by the lengths of the two sides.
  - "Circle" — The co-ordinates of the centre, followed by the radius.
  - "Column" — The Axis 1 co-ordinate of the column.
  - "Ellipse" — The co-ordinates of the centre, followed by the lengths of the semi-major and semi-minor axes, followed by the angle between Axis 1 and the semi-major axis (in radians).
  - "Frame" — The width of the border.
  - "Point" — The co-ordinates of the pixel.
  - "Polygon" — The co-ordinates of each vertex in the order given.
  - "Rectangle" — The co-ordinates of two diagonally opposite corners.
  - "Rotbox" — The co-ordinates of the box centre, followed by the lengths of the two sides, followed by the angle between the first side and Axis 1 (in radians).
  - "Row" — The Axis 2 co-ordinate of the row.
  - "Whole" — No arguments.
- The shapes are defined within the current co-ordinate Frame of the displayed NDF. For instance, if the current co-ordinate Frame of the displayed NDF is RA/DEC, then "COLUMN" regions will be curves of constant DEC, "ROW" regions will be curves of constant RA (assuming Axis 1 is RA and Axis 2 is DEC), straight lines will correspond to geodesics, *etc*. Numerical values will be stored in the output text file in the current co-ordinate Frame of the NDF. WCS information will also be stored in the output text file allowing the stored positions to be converted to other systems (pixel co-ordinates, for instance).

Related Applications :

KAPPA: ARDPLOT, ARDMASK, LOOK, REGIONMASK; CCDPACK; ESP.

ARDMASK
Uses an ARD file to set some pixels of an NDF to be bad

Description:

This task allows regions of an NDF to be masked, so that they can (for instance) be excluded from subsequent data processing. ARD (ASCII Region Definition) descriptions (SUN/183) stored in a text file define which pixels of the data array are masked. An output NDF is created which is the same as the input file except that all pixels specified by the ARD file have been assigned either the bad value or a specified constant value. This value can be assigned to either the inside or the outside of the specified ARD region.

If positions in the ARD description are given using a co-ordinate system that has one fewer axes than the input NDF, then each line or plane in the NDF will be masked independently using the supplied ARD description. For instance, if a two-dimensional ARD description that uses (RA,Dec) to specify positions is used to mask a three-dimensional (ra,dec,velocity) NDF, then each velocity plane in the NDF will be masked independently.

Usage:

```
ardmask in ardfile out
```

Parameters:

ARDFILE = FILENAME (Read)

The name of the ARD file containing a description of the parts of the image to be masked out, *i.e.* set to bad. The co-ordinate system in which positions within this file are given should be indicated by including suitable COFRAME or WCS statements within the file (see SUN/183), but will default to pixel co-ordinates or current WCS Frame co-ordinates in the absence of any such statements (see Parameter DEFPIX). For instance, starting the file with a line containing the text `"COFRAME(SKY,System=FK5)"` would indicate that positions are specified in RA/DEC (FK5,J2000). The statement `"COFRAME(PIXEL)"` indicates explicitly that positions are specified in pixel co-ordinates.

COMP = LITERAL (Read)

The NDF array component to be masked. It may be `"Data"`, or `"Variance"`, or `"Error"`, or `"All"`, (where `"Error"` is equivalent to `"Variance"`). `["All"]`

CONST = LITERAL (Given)

The constant numerical value to assign to the region, or the string `"bad"`. `["bad"]`

DEFPIX = _LOGICAL (Read)

If a `TRUE` value is supplied for DEFPIX, then co-ordinates in the supplied ARD file will be assumed to be pixel co-ordinates. Otherwise, they are assumed to be in the current WCS co-ordinate system of the supplied NDF. `[TRUE]`

IN = NDF (Read)

The name of the source NDF.

INSIDE = _LOGICAL (Read)

If a `TRUE` value is supplied, the constant value is assigned to the inside of the region specified by the ARD file. Otherwise, it is assigned to the outside. `[TRUE]`

OUT = NDF (Write)
The name of the masked NDF.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (`!`) propagates the title from the input NDF to the output NDF. `[!]`

Examples:

```
ardmask a1060 galaxies.ard a1060_sky title="A1060 galaxies masked"
```

This flags pixels defined by the ARD file galaxies.ard within the NDF called a1060 to create a new NDF called a1060_sky. a1060_sky has a title=`"A1060 galaxies masked"`. This might be to flag the pixels where bright galaxies are located to exclude them from sky-background fitting.

```
ardmask in=ic3374 ardfil=ardfile.txt out=ic3374a
```

This example uses as the source image the NDF called ic3374 and sets the pixels specified by the ARD description contained in `ardfile.txt` to the bad value. The resultant image is output to the NDF called ic3374a. The title is unchanged.

ASCII-region-definition Descriptors :

The ARD file may be created by ARDGEN or written manually. In the latter case consult SUN/183 for full details of the ARD descriptors and syntax; however, much may be learnt from looking at the ARD files created by ARDGEN and the ARDGEN documentation. There is also a in Section 15.1.1.

Related Applications :

KAPPA: ARDGEN, ARDPLOT, LOOK, REGIONMASK.

Implementation Status:

- This routine correctly processes the WCS, AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All numeric data types can be handled.

ARDPLOT
Plot regions described in an ARD file

Description:

This application draws the outlines of regions described in a supplied two-dimensional ARD file (an 'ARD Description' (see SUN/183). If there is an existing picture on the graphics device, the outlines are drawn over the top of the previously displayed picture, aligned (if possible) in the current co-ordinate Frame of the previously drawn picture. If the graphics device is empty (or if the CLEAR parameter is set `TRUE`) the outlines are drawn using a default projection—the size of the area plotted can be controlled by the SIZE parameter. Note, the facility to plot on an empty device is currently only available for two-dimensional regions specified using Parameter REGION.

Usage:

```
ardplot ardfile [device] [regval]
```

Parameters:

ARDFILE = FILENAME (Read)

The name of a file containing an 'ARD Description' of the regions to be outlined. The co-ordinate system in which positions within this file are given should be indicated by including suitable COFRAME or WCS statements within the file (see SUN/183), but will default to pixel co-ordinates in the absence of any such statements. For instance, starting the file with a line containing the text `"COFRAME(SKY,System=FK5)"` would indicate that positions are specified in RA/DEC (FK5,J2000). The statement `"COFRAME(PIXEL)"` indicates explicitly that positions are specified in pixel co-ordinates. The ARDFILE parameter is only accessed if Parameter REGION is given a null (`!`) value.

CLEAR = _LOGICAL (Read)

`TRUE` if the current picture is to be cleared before the Region is display. `[FALSE]`

DEVICE = DEVICE (Read)

The plotting device. [Current graphics device]

REGION = FILENAME (Read)

The name of a file containing an AST Region to be outlined, or null (`!`) if the ARD region defined by Parameter ARDFILE is to be outlined. Suitable files can be created using the ATOOLS package. `[!]`

REGVAL = _INTEGER (Read)

Indicates which regions within the ARD description are to be outlined. If zero (the default) is supplied, then the plotted boundary encloses all the regions within the ARD file. If a positive value is supplied, then only the region with the specified index is outlined (the first region in the ARD file has index 2, for historical reasons). If a negative value is supplied, then all regions with indices greater than or equal to the absolute value of the supplied index are outlined. See SUN/183 for further information on the numbering of regions within an ARD description. The REGVAL parameter is only accessed if Parameter REGION is given a null (`!`) value. `[0]`

SIZE = _REAL (Read)

The size of the plot to create, given as a multiple of the size of the Region being plotted. This parameter is only accessed if no DATA picture can be found on the graphics device, or CLEAR is `TRUE`. A SIZE value of `1.0` causes the plot to be the same size as the Region being plotted. A value of `2.0` causes the plot to be twice the size of the Region, *etc*. `[2.0]`

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the curves.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the plotted curves is controlled by the attributes Colour(Curves), Width(Curves), *etc*. [current value]

Examples:

```
ardplot bulge
```

Draws an outline around all the regions included in the ardfile named `bulge`. The outline is drawn on the current graphics device and is drawn in alignment with the previous picture.

Notes:

- A DATA picture must already exist on the selected graphics device before running this command. An error will be reported if no DATA picture can be found.
- The application stores a new DATA picture in the graphics database. On exit the current database picture for the chosen device reverts to the input picture.

Related Applications :

KAPPA: ARDGEN, ARDMASK, LOOK.

AXCONV
Expands spaced axes in an NDF into the primitive form

Description:

This application routine converts *in situ* an NDF's axis centres in the 'spaced' form into 'simple' form. Applications using the NDF_ library, such as KAPPA, are not currently capable of supporting spaced arrays, but there are packages that produce NDF files with this form of axis, notably ASTERIX. This application provides a temporary method of allowing KAPPA *et al.* to handle these NDF datasets.

Usage:

```
axconv ndf
```

Parameters:

NDF = NDF (Read and Write)
The NDF to be modified.

Examples:

```
axconv rosat256
```

This converts the spaced axes in the NDF called rosat256 into simple form.

Related Applications :

KAPPA: SETAXIS.

Implementation Status:

- Only axes with a real data type are created.

AXLABEL
Sets a new label value for an axis within an NDF data structure

Description:

This routine sets a new value for a LABEL component of an existing NDF AXIS data structure. The NDF is accessed in update mode and any pre-existing LABEL component is over-written with a new value. Alternatively, if a 'null' value (`!`) is given for the LABEL parameter, then the NDF's axis LABEL component will be erased. If an AXIS structure does not exist, a new one whose centres are pixel co-ordinates is created.

Usage:

```
axlabel ndf label dim
```

Parameters:

DIM = _INTEGER (Read)

The axis dimension for which the label is to be modified. There are separate labels for each NDF dimension. The value must lie between 1 and the number of dimensions of the NDF. This defaults to 1 for a one-dimensional NDF. The suggested default is the current value. `[]`

NDF = NDF (Read and Write)

The NDF data structure in which an axis LABEL component is to be modified.

LABEL = LITERAL (Read)

The value to be assigned to the NDF's axis LABEL component (*e.g.* `"Wavelength"` or `"Fibre index"`). LABEL describes the quantity measured along the axis. This value may later be used by other applications for labelling graphs or as a heading for columns in tabulated output. The suggested default is the current value.

Examples:

```
axlabel ngc253 "Offset from nucleus" 2
```

Sets the LABEL component of the second axis dimension of the NDF structure ngc253 to have the value `"Offset from nucleus"`.

```
axlabel ndf=spect label=Wavelength
```

Sets the axis LABEL component of the one-dimensional NDF structure spect to have the value `"Wavelength"`.

```
axlabel datafile label=!  dim=3
```

By specifying a null value (`!`), this example erases any previous value of the LABEL component for the third dimension in the NDF structure datafile.

Related Applications :

KAPPA: AXUNITS, SETAXIS, SETLABEL.

AXUNITS
Sets a new units value for an axis within an NDF data structure

Description:

This routine sets a new value for a UNITS component of an existing NDF AXIS data structure. The NDF is accessed in update mode and any pre-existing UNITS component is over-written with a new value. Alternatively, if a 'null' value (`!`) is given for the UNITS parameter, then the NDF's axis UNITS component will be erased. If an AXIS structure does not exist, a new one whose centres are pixel co-ordinates is created.

Usage:

```
axunits ndf units dim
```

Parameters:

DIM = _INTEGER (Read)

The axis dimension for which the units is to be modified. There are separate units for each NDF dimension. The value must lie between 1 and the number of dimensions of the NDF. This defaults to 1 for a one-dimensional NDF. The suggested default is the current value. `[]`

NDF = NDF (Read and Write)

The NDF data structure in which an axis UNITS component is to be modified.

UNITS = LITERAL (Read)

The value to be assigned to the NDF's axis UNITS component (*e.g.* `"Pixels"` or `"km/s"`). UNITS describes the physical units of the quantity measured along the axis. This value may later be used by other applications for labelling graphs and other forms of display where the NDF's axis co-ordinates are shown. The suggested default is the current value.

Examples:

```
axunits ngc253 "arcsec" 2
```

Sets the UNITS component of the second axis dimension of the NDF structure ngc253 to have the value `"arcsec"`.

```
axunits ndf=spect units=Angstrom
```

Sets the axis UNITS component of the one-dimensional NDF structure spect to have the value `"Angstrom"`.

```
axunits datafile units=!  dim=3
```

By specifying a null value (`!`), this example erases any previous value of the UNITS component for the third dimension in the NDF structure datafile.

Related Applications :

KAPPA: AXLABEL, SETAXIS, SETUNITS.

BEAMFIT
Fits beam features in a two-dimensional NDF

Description:

This fits generalised Gaussians (*cf.* PSF) to beam features within the data array of a two-dimensional NDF given approximate initial co-ordinates. It uses an unconstrained least-squares minimisation involving the residuals and a modified Levenberg-Marquardt algorithm. The beam feature is a set of connected pixels which are either above or below the surrounding background region. The errors in the fitted coefficients are also calculated.

You may apply various constraints. These are either fixed, or relative. Fixed values include the FWHM, background level, or the shape exponent that defaults to 2 thus fits a normal distribution. Relative constraints define the properties of secondary beam features with respect to the primary (first given) feature, and can specify amplitude ratios, and beam separations in Cartesian or polar co-ordinates.

Four methods are available for obtaining the initial positions, selected using Parameter MODE:

- from the parameter system (see Parameters POS, POS2–POS5);
- using a graphics cursor to indicate the feature in a previously displayed data array (see Parameter DEVICE);
- from a specified positions list (see Parameter INCAT); or
- from a simple text file containing a list of co-ordinates (see Parameter COIN).

In the first two modes the application loops, asking for new feature co-ordinates until it is told to quit or encounters an error or the maximum number of features is reached. The last is five, unless Parameters POS2—POS5 define the location of the secondary beams and then only the primary beam's position is demanded.

BEAMFIT both reports and stores in parameters its results. These are fit coefficients and their errors, the offsets and position angles of the secondary beam features with respect to the primary beam, and the offset of the primary beam from a reference position. Also a listing of the fit results may be written to a log file geared more towards human readers, including details of the input parameters (see Parameter LOGFILE).

Usage:

```
                          { incat=?
                          {
                          { [beams]
beamfit ndf [mode]        {
                          { coin=?
                          {
                          { [beams] pos pos2-pos5=?
                     mode
```

Parameters:

AMPRATIO( ) = _REAL (Read)
If number of beam positions given by BEAMS is more than one, this specifies the ratio of the amplitude of the secondary beams to the primary. Thus you should supply one fewer value than the number of beams. If you give fewer than that the last ratio is copied to the missing values. The ratios would normally be negative, usually -1 or -0.5. AMPRATIO is ignored when there is only one beam feature to fit. [!]

BEAMS = _INTEGER (Read)
The number of beam positions to fit. This will normally be 1, unless a chopped observation is supplied, when there may be two or three beam positions. This parameter is ignored for "File" and "Catalogue" modes, where the number comes from the number of beam positions read from the files; and for "Interface" mode when the beam positions POS, POS2, *etc.* are supplied in full on the command line without BEAMS. In all modes there is a maximum of five positions, which for "File" or "Catalogue" modes will be the first five. [1]

CIRCULAR = _LOGICAL (Read)
If set TRUE only circular beams will be fit. [FALSE]

COIN = FILENAME (Read)
Name of a text file containing the initial guesses at the co-ordinates of beams to be fitted. It is only accessed if Parameter MODE is given the value "File". Each line should contain the formatted axis values for a single position, in the current Frame of the NDF. Axis values can be separated by spaces, tabs or commas. The file may contain comment lines with the first character # or !.

DESCRIBE = _LOGICAL (Read)
If TRUE, a detailed description of the co-ordinate Frame in which the beam positions will be reported is displayed before the positions themselves. [current value]

DEVICE = DEVICE (Read)
The graphics device which is to be used to give the initial guesses at the beam positions. Only accessed if Parameter MODE is given the value "Cursor". [Current graphics device]

FITAREA() = _INTEGER (Read)
Size in pixels of the fitting area to be used. This should fully encompass the beam and also include some background signal. If only a single value is given, then it will be duplicated to all dimensions so that a square region is fitted. Each value must be at least 9. A null value requests that the full data array is used. [!]

FIXAMP = _DOUBLE (Read)
This specifies the fixed amplitude of the first beam. Secondary sources arising from chopped data use FIXAMP multiplied by the AMPRATIO. A null value indicates that the amplitude should be fitted. [!]

FIXBACK = _DOUBLE (Read)
If a non-null value is supplied then the model fit will use that value as the constant background level otherwise the background is a free parameter of the fit. [!]

FIXFWHM = LITERAL (Read)
If this is set TRUE then the model fit will use the full-width half-maximum values for the beams supplied through Parameter FWHM. FALSE demands that the FWHM values are free parameters of the fit. [FALSE]

FIXPOS = _LOGICAL (Read)

If TRUE, the supplied position of each beam is used and the centre co-ordinates of the beam features are not fit. FALSE causes the initial estimate of the location of each beam to come from the source selected by Parameter MODE, and all these locations are part of the fitting process (however note the exception when FIXSEP=TRUE. It is advisable not to use this option in the inaccurate "Cursor" mode. [FALSE]

FIXSEP = _LOGICAL (Read)

If TRUE, the separations of secondary beams from the primary beam are fixed, and this takes precedence over Parameter FIXPOS. If FALSE, the beam separations are free to be fitted (although it is actually the centres being fit). It is advisable not to use this option in the inaccurate "Cursor" mode. [FALSE]

FWHM = LITERAL (Read)

The initial full-width half-maximum (FWHM) values for each beam. These become fixed values if FIXFWHM is set TRUE.

A number of options are available.

- A single value gives the same circular FWHM for all beams.
- When Parameter CIRCULAR is TRUE, supply a list of values one for each of the number of beams. These should be supplied in the same order as the corresponding beam positions.
- A pair of values sets the major- and minor-axis values for all beams, provided Parameter CIRCULAR is FALSE.
- Major- and minor-axis pairs, whose order should match that of the corresponding beams. Again CIRCULAR should be FALSE.

Multiple values are separated by commas. An error is issued should none of these options be offered.

If the current co-ordinate Frame of the NDF is a SKY Frame (*e.g.* right ascension and declination), then the value should be supplied as an increment of celestial latitude (*e.g.* declination). Thus, "5.7" means 5.7 arcseconds, "20:0" would mean 20 arcminutes, and "1:0:0" would mean 1 degree. If the current co-ordinate Frame is not a SKY Frame, then the widths should be specified as an increment along Axis 1 of the current co-ordinate Frame. Thus, if the Current Frame is PIXEL, the value should be given simply as a number of pixels.

Null requests that BEAMFIT itself estimates the initial FWHM values. [!]

GAUSS = _LOGICAL (Read)

If TRUE, the shape exponent is fixed to be 2; in other words the beams are modelled as two-dimensional normal distributions. If FALSE, the shape exponent is a free parameter in each fit. [TRUE]

INCAT = FILENAME (Read)

A catalogue containing a positions list giving the initial guesses at the beam positions, such as produced by applications CURSOR, LISTMAKE, *etc.* It is only accessed if Parameter MODE is given the value "Catalogue".

LOGFILE = FILENAME (Read)

Name of the text file to log the results. If null, there will be no logging. Note this is intended for the human reader and is not intended for passing to other applications. [!]

MARK = LITERAL (Read)
Only accessed if Parameter MODE is given the value `"Cursor"`. It indicates which positions are to be marked on the screen using the marker type given by Parameter MARKER. It can take any of the following values.

- `"Initial"` — The position of the cursor when the mouse button is pressed is marked.
- `"Fit"` — The corresponding fit position is marked.
- `"Ellipse"` — As "Fit" but it also plots an ellipse at the HWHM radii and orientation.
- `"None"` — No positions are marked.

[current value]

MARKER = INTEGER (Read)
This parameter is only accessed if Parameter MARK is set `TRUE`. It specifies the type of marker with which each cursor position should be marked, and should be given as an integer PGPLOT marker type. For instance, `0` gives a box, `1` gives a dot, `2` gives a cross, `3` gives an asterisk, `7` gives a triangle. The value must be larger than or equal to −31. [current value]

MODE = LITERAL (Read)
The mode in which the initial co-ordinates are to be obtained. The supplied string can be one of the following values.

- `"Interface"` — positions are obtained usingparameters POS, POS2–POS5.
- `"Cursor"` — positions are obtained using the graphics cursor of the device specified by Parameter DEVICE.
- `"Catalogue"` — positions are obtained from a positions list using Parameter INCAT.
- `"File"` — positions are obtained from a text file using Parameter COIN. [current value]

NDF = NDF (Read)
The NDF structure containing the data array to be analysed. In cursor mode (see Parameter MODE), the run-time default is the displayed data, as recorded in the graphics database. In other modes, there is no run-time default and the user must supply a value. `[]`

PLOTSTYLE = GROUP (Read)
A group of attribute settings describing the style to use when drawing the graphics markers specified by Parameter MARK.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported). [current value]

POLAR = _LOGICAL (Read)

If TRUE, the co-ordinates supplied through POS2–POS5 are interpreted in polar co-ordinates (offset, position angle) about the primary beam. The radial co-ordinate is a distance measured in units of the latitude axis if the current WCS Frame is a SKY DOMAIN or the first axis for other Frames. For a SKY current WCS Frame, position angle follows the standard convention of North through East. For other Frames the angle is measured from the second axis anticlockwise, *e.g.* for a PIXEL Frame it would be from y through negative x, not the standard x through y.

If FALSE, the co-ordinates are the regular axis co-ordinates in the current Frame.

POLAR is only accessed when there is more than one beam to fit. [TRUE]

POS = LITERAL (Read)

When MODE = "Interface" POS specifies the co-ordinates of the primary beam position. This is either merely an initial guess for the fit, or if Parameter FIXPOS is TRUE, it defines a fixed location. It is specified in the current co-ordinate Frame of the NDF (supplying a colon ":" will display details of the current co-ordinate Frame). A position should be supplied as a list of formatted WCS axis values separated by spaces or commas, and should lie within the bounds of the NDF.

If the initial co-ordinates are supplied on the command line without BEAMS the number of contiguous POS, POS2,... parameters specifies the number of beams to be fit. If the initial co-ordinates are supplied on the command line without BEAMS specified only one beam will be fit.

POS2-POS5 = LITERAL (Read)

When MODE = "Interface" these parameters specify the co-ordinates of the secondary beam positions. These should lie within the bounds of the NDF. For each parameter the supplied location may be merely an initial guess for the fit, or if Parameter FIXPOS is TRUE, it defines a fixed location, unless Parameter FIXSEP is TRUE, whereupon it defines a fixed separation from the primary beam.

For POLAR = FALSE each distance should be given as a single literal string containing a space- or comma-separated list of formatted axis values measured in the current co-ordinate Frame of the NDF. The allowed formats depends on the class of the current Frame. Supplying a single colon ":" will display details of the current Frame, together with an indication of the format required for each axis value, and a new parameter value is then obtained.

If Parameter POLAR is TRUE, POS2–POS5 may be given as an offset followed by a position angle. See Parameter POLAR for more details of the sense of the angle and the offset co-ordinates.

The parameter name increments by 1 for each subsequent beam feature. Thus POS2 applies to the first secondary beam (second position in all), POS3 is for the second

secondary beam, and so on. As the total number of parameters required is one fewer than the value of Parameter BEAMS, POS2–POS5 are only accessed when BEAMS exceeds 1.

REFPOS = LITERAL (Read)
The reference position. This is often the desired position for the beam. The offset of the primary beam with respect to this point is reported and stored in Parameter REFOFF. It is only accessed if the current WCS Frame in the NDF is not a SKY Domain containing a reference position.

The co-ordinates are specified in the current WCS Frame of the NDF (supplying a colon `":"` will display details of the current co-ordinate Frame). A position should be supplied either as a list of formatted WCS axis values separated by spaces or commas. A null value (`!`) requests that the centre of the supplied map is deemed to be the reference position.

RESID = NDF (Write)
The map of the residuals (data minus model) of the fit. It inherits the properties of the input NDF, except that its data type is _DOUBLE or _REAL depending on the precision demanded by the type of IN, and no variance is propagated. A null (`!`) value requests that no residual map be created. `[!]`

TITLE = LITERAL (Read)
The title for the NDF to contain the residuals of the fit. If null (`!`) is entered the NDF will not contain a title. `["KAPPA - BEAMFIT"]`

VARIANCE = _LOGICAL (Read)
If `TRUE`, then any VARIANCE component present within the input NDF will be used to weight the fit; the weight used for each data value is the reciprocal of the variance. If set to `FALSE` or there is no VARIANCE present, all points will be given equal weight. `[FALSE]`

Results Parameters:

AMP(2 * BEAMS) = _DOUBLE (Write)
The amplitude and its error for each beam.

BACK(2 * BEAMS) = _DOUBLE (Write)
The background level and its error at each beam position.

CENTRE(2 * BEAMS) = LITERAL (Write)
The formatted co-ordinates and their errors of each beam in the current co-ordinate Frame of the NDF.

GAMMA(2 * BEAMS) = _DOUBLE (Write)
The shape exponent and its error for each beam.

MAJFWHM(2 * BEAMS) = _DOUBLE (Write)
The major-axis FWHM and its error, measured in the current co-ordinate Frame of the NDF, for each beam. Note that the unit for sky co-ordinate Frames is radians.

MINFWHM(2 * BEAMS) = _DOUBLE (Write)
The minor-axis FWHM and its error, measured in the current co-ordinate Frame of the NDF, for each beam. Note that the unit for sky co-ordinate Frames is radians.

OFFSET( ) = LITERAL (Write)
The formatted offset and its error of each secondary beam feature with respect to the primary beam. They are measured in the current Frame of the NDF along a latitude axis if that Frame is in the SKY Domain, or the first axis otherwise. The number of values stored is twice the number of beams. The array alternates an offset, then its corresponding error, appearing in beam order starting with the first secondary beam.

ORIENT(2 * BEAMS) = _DOUBLE (Write)
The orientation and its error, measured in degrees for each beam. If the current WCS Frame is a SKY Frame, the angle is measured from North through East. For other Frames the angle is from the *x*-axis through *y*.

PA() = _REAL (Write)
The position angle and its errors of each secondary beam feature with respect to the primary beam. They are measured in the current Frame of the NDF from North through East if that is a SKY Domain, or anticlockwise from the *y* axis otherwise. The number of values stored is twice the number of beams. The array alternates a position angle, then its corresponding error, appearing in beam order starting with the first secondary beam.

REFOFF(2) = LITERAL (Write)
The formatted offset followed by its error of the primary beam's location with respect to the reference position (see Parameter REFPOS). The offset might be used to assess the optical alignment of an instrument. The ofset and its error are measured in the current Frame of the NDF along a latitude axis if that Frame is in the SKY Domain, or the first axis otherwise. The error is derived entirely from the uncertainities in the fitted position of the primary beam, *i.e.* the reference position has no error attached to it. By definition the error is zero when FIXPOS is `TRUE`.

RMS = _REAL (Write)
The primary beam position's root mean-squared deviation from the fit.

SUM = _DOUBLE (Write)
The total data sum of the multi-Gaussian fit above the background. The fit is evaluated at the centre of every pixel in the input NDF (including bad-valued pixels). The fitted background level is then removed from the fit value, and the sum of these is written to this output parameter.

Examples:

```
beamfit mars_3pos i 1 "5.0,-3.5"
```

This finds the Gaussian coefficients of the primary beam feature in the NDF called mars_3pos, using the supplied beam's centre. The co-ordinates are measured in the NDF's current co-ordinate Frame. In this case they are offsets in arcseconds.

```
beamfit ndf=mars_3pos mode=interface beams=1 init1="5.0,-3.5" fixback=0
```

As above but now the background is fixed to be zero.

```
beamfit mars_3pos i pos="5.0,-3.5"
```

As the first example. The presence of POS indicates a single is required.

```
beamfit ndf=mars_3pos mode=interface beams=1 pos="5.0,-3.5" fixfwhm
fwhm=16.5 gauss=f
```

As above but now the Gaussian is constrained to have a FWHM of 16.5 arcseconds and be circular, but the shape exponent is not constrained to be 2.

```
beamfit mars_3pos in beams=1 fwhm=16.5 fitarea=51 pos="5.,-3.5"
```

As above but now the fitted data is restricted to areas 51×51 pixels about the initial guess positions. All the other examples use the full array. Also the FWHM value is now just an initial guess.

```
beamfit mars_3pos int 3 "5.0,-3.5" ampratio=-0.5 resid=mars_res
```

As the first example except this finds the Gaussian coefficients of the primary beam feature and two secondary features. The secondary features have fixed amplitudes that are half that of the primary feature and of the opposite polarity. The residuals after subtracting the fit are stored in NDF mars_res. In all the other examples no residual map is created.

```
beamfit mars_3pos int 2 "5.0,-3.5" pos2="60.0,90" fixpos
```

This finds the Gaussian coefficients of the primary beam feature and a secondary feature in the NDF called mars_3pos. The supplied co-ordinates (5.0,−3.5) define the centre, *i.e.* they are not fitted. Also the secondary beam is fixed at 60 arcseconds towards the East (position angle 90 degrees).

```
beamfit mars_3pos int 2 "5.0,-3.5" pos2="60.0,90" fixsep
```

As the previous example, except now the separation of the second position is fixed at 60 arcseconds towards the East from the primary beam, instead of being an absolute location.

```
beamfit mars_3pos int 2 "5.0,-3.5" pos2="-60.5,0.6" polar=f fixpos
```

As the last-but-one example, but now location of the secondary beam is fixed at (−55.5,−2.9).

```
beamfit s450 int beams=2 fwhm="7.9,25" ampratio=0.06 circular
pos='"0:0:0,0:0:0"' nopolar pos2="0:0:0,0:0:0"
```

This fits two superimposed circular Gaussians in the NDF called s450, whose current WCS is SKY. The beam second being fixed at 6 percent the strength of the first, with initial widths of 7.9 and 25 arcseconds.

```
beamfit mode=cu beams=1
```

This finds the Gaussian coefficients of the primary beam feature of an NDF, using the graphics cursor on the current graphics device to indicate the approximate centre of the feature. The NDF being analysed comes from the graphics database.

```
beamfit uranus cu 2 mark=ce plotstyle='colour=red' marker=3
```

This fits to two beam features in the NDF called uranus via the graphics cursor on the current graphics device. The beam positions are marked using a red asterisk.

```
beamfit uranus file 4 coin=features.dat logfile=uranus.log
```

This fits to the beam features in the NDF called uranus. The initial positions are given in the text file `features.dat` in the current co-ordinate Frame. Only the first four positions will be used. The last three positions are in polar co-ordinates with respect to the primary beam. A log of selected input parameter values, and the fitted coefficients and errors is written to the text file `uranus.log`.

```
beamfit uranus mode=cat incat=uranus_beams polar=f
```

This example reads the initial guess positions from the positions list in file `uranus_beams.FIT`. The number of beam features fit is the number of positions in the catalogue subject to a maximum of five. The input file may, for instance, have been created using the application CURSOR.

Notes:

- All positions are supplied and reported in the current co-ordinate Frame of the NDF. A description of the co-ordinate Frame being used is given if Parameter DESCRIBE is set to a `TRUE` value. Application WCSFRAME can be used to change the current co-ordinate Frame of the NDF before running this application if required.
- The uncertainty in the positions are estimated iteratively using the curvature matrix derived from the Jacobian, itself determined by a forward-difference approximation.
- The fit parameters are not displayed on the screen when the message filter environment variable MSG_FILTER is set to `QUIET`.
- If the fitting fails there are specific error codes that can be tested and appropriate action taken in scripts: `PDA__FICMX` when it is impossible to derive fit errors, and `KAP__LMFOJ` when the fitted functions from the Levenberg-Marquardt minimisation are orthogonal to the Jacobian's columns (usually indicating that FITAREA is too small).

Related Applications :

KAPPA: PSF, CENTROID, CURSOR, LISTSHOW, LISTMAKE; ESP: GAUFIT; FIGARO: FITGAUSS.

Implementation Status:

- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using double-precision floating point.

BLOCK
Smooths an NDF using an n-dimensional rectangular box filter.

Description:

This application smooths an n-dimensional NDF using a rectangular box filter, whose dimensionality is the same as that of the NDF being smoothed. Each output pixel is either the mean or the median of the input pixels within the filter box. The mean estimator provides one of the fastest methods of smoothing an image and is often useful as a general-purpose smoothing algorithm when the exact form of the smoothing point-spread function is not important.

It is possible to smooth in selected dimensions by setting the boxsize to 1 for the dimensions not requiring smoothing. For example you can apply two-dimensional smoothing to the planes of a three-dimensional NDF (see Parameter BOX). If it has three dimensions, then the filter is applied in turn to each plane in the cube and the result written to the corresponding plane in the output cube.

Usage:

```
block in out box [estimator]
```

Parameters:

BOX() = _INTEGER (Read)

The sizes (in pixels) of the rectangular box to be applied to smooth the data. These should be given in axis order. A value set to `1` indicates no smoothing along that axis. Thus, for example, BOX=`[3,3,1]` for a three-dimensional NDF would apply a 3x3-pixel filter to all its planes independently.

If fewer values are supplied than the number of dimensions of the NDF, then the final value will be duplicated for the missing dimensions.

The values given will be rounded up to positive odd integers, if necessary, to retain symmetry.

ESTIMATOR = LITERAL (Read)

The method to use for estimating the output pixel values. It can be either `"Mean"` or `"Median"`. `["Mean"]`

IN = NDF (Read)

The input NDF to which box smoothing is to be applied.

OUT = NDF (Write)

The output NDF which is to contain the smoothed data.

TITLE = LITERAL (Read)

The title for the output NDF. A null value will cause the title of the input NDF to be used. `[!]`

WLIM = _REAL (Read)

If the input image contains bad pixels, then this parameter may be used to determine the number of good pixels which must be present within the smoothing box before a valid output pixel is generated. It can be used, for example, to prevent output

pixels from being generated in regions where there are relatively few good pixels to contribute to the smoothed result.

By default, a null (!) value is used for WLIM, which causes the pattern of bad pixels to be propagated from the input image to the output image unchanged. In this case, smoothed output values are only calculated for those pixels which are not bad in the input image.

If a numerical value is given for WLIM, then it specifies the minimum fraction of good pixels which must be present in the smoothing box in order to generate a good output pixel. If this specified minimum fraction of good input pixels is not present, then a bad output pixel will result, otherwise a smoothed output value will be calculated. The value of this parameter should lie between 0.0 and 1.0 (the actual number used will be rounded up if necessary to correspond to at least one pixel). [!]

Examples:

```
block aa bb 9
```

Smooths the two-dimensional image held in the NDF structure aa, writing the result into the structure bb. The smoothing box is 9 pixels square. If any pixels in the input image are bad, then the corresponding pixels in the output image will also be bad. Each output pixel is the mean of the corresponding input pixels.

```
block spectrum spectrums [5,1] median title="Smoothed spectrum"
```

Smooths the one-dimensional data in the NDF called spectrum using a box size of 5 pixels, and stores the result in the NDF structure spectrums. Each output pixel is the median of the corresponding input pixels. If any pixels in the input image are bad, then the corresponding pixels in the output image will also be bad. The output NDF has the title "Smoothed spectrum".

```
block ccdin(123,) ccdcol [1,9]
```

Smooths the 123rd column in the two-dimensional NDF called ccdin using a box size of 9 pixels, and stores the result in the NDF structure ccdcol. The first value of the smoothing box is ignored as the first dimension has only one element. Each output pixel is the mean of the corresponding input pixels.

```
block in=image1 out=image2 box=[5,7] estimator=median
```

Smooths the two-dimensional image held in the NDF structure image1 using a rectangular box of size 5×7 pixels. The smoothed image is written to the structure image2. Each output pixel is the median of the corresponding input pixels.

```
block etacar etacars box=[7,1] wlim=0.6
```

Smooths the specified image data using a rectangular box 7×1 pixels in size. Smoothed output values are generated only if at least 60% of the pixels in the smoothing box are good, otherwise the affected output pixel is bad.

```
block in=cubein out=cubeout box=[3,3,7]
```

Smooths the three-dimensional NDF called cubein using a box that has three elements along the first two axes and seven along the third. The smoothed cube is written to NDF cubeout.

```
block in=cubein out=cubeout box=[3,1,7]
```

As the previous example, except that planes comprising the first and third axes are smoothed independently for all lines.

Timing :

When using the mean estimator, the execution time is approximately proportional to the number of pixels in the image to be smoothed and is largely independent of the smoothing box size. This makes the routine particularly suitable for applying heavy smoothing to an image. Execution time will be approximately doubled if a variance array is present in the input NDF.

The median estimator is much slower than the mean estimator, and is heavily dependent on the smoothing box size.

Related Applications :

KAPPA: CONVOLVE, FFCLEAN, GAUSMOOTH, MEDIAN; FIGARO: ICONV3, ISMOOTH, IXSMOOTH, MEDFILT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions. In addition, if the mean estimator is used, the VARIANCE component is also processed. If the median estimator is used, then the output NDF will have no VARIANCE component, even if there is a VARIANCE component in the input NDF.
- Processing of bad pixels and automatic quality masking are supported. The bad-pixel flag is also written for the data and variance arrays.
- All non-complex numeric data types can be handled. Arithmetic is performed using single-precision floating point, or double precision if appropriate.
- Huge NDFs are supported.

CADD
Adds a scalar to an NDF data structure

Description:

The routine adds a scalar (*i.e.* constant) value to each pixel of an NDF's data array to produce a new NDF data structure.

Usage:

```
cadd in scalar out
```

Parameters:

IN = NDF (Read)
Input NDF data structure, to which the value is to be added.

OUT = NDF (Write)
Output NDF data structure.

SCALAR = _DOUBLE (Read)
The value to be added to the NDF's data array.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
cadd a 10 b
```

This adds ten to the NDF called a, to make the NDF called b. NDF b inherits its title from a.

```
cadd title="HD123456" out=b in=a scalar=17.3
```

This adds 17.3 to the NDF called a, to make the NDF called b. NDF b has the title `"HD123456"`.

Related Applications :

KAPPA: ADD, CDIV, CMULT, CSUB, DIV, MATHS, MULT, SUB.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Huge NDFs are supported.

CALC
Evaluates a mathematical expression

Description:

This task evaluates an arithmetic expression and reports the result. It main rôle is to perform floating-point arithmetic in scripts. A value "Bad" is reported if there was an error during the calculation, such as a divide by zero.

Usage:

```
calc exp [prec] fa-fz=?  pa-pz=?
```

Parameters:

EXP = LITERAL (Read)

The mathematical expression to be evaluated, *e.g.* "-2.5∗LOG10(PA)". In this expression constants may either be given literally or represented by the variables PA, PB, ... PZ. The expression may contain sub-expressions represented by the variables FA, FB, ... FZ. Values for those sub-expressions and constants which appear in the expression will be requested via the application's parameter of the same name.

FORTRAN 77 syntax is used for specifying the expression, which may contain the usual intrinsic functions, plus a few extra ones. An appendix in SUN/61 gives a full description of the syntax used and an up-to-date list of the functions available. The arithmetic operators (+,-,/,∗,∗∗) follow the normal order of precedence. Using matching (nested) parentheses will explicitly define the order of expression evaluation. The expression may be up to 132 characters long.

FA-FZ = LITERAL (Read)

These parameters supply the values of 'sub-expressions' used in the expression EXP. Any of the 26 may appear; there is no restriction on order. These parameters should be used when repeated expressions are present in complex expressions, or to shorten the value of EXP to fit within the 132-character limit. Sub-expressions may contain references to other sub-expressions and constants (PA-PZ). An example of using sub-expressions is:

EXP > PA∗ASIND(FA/PA)∗X/FA

FA > SQRT(X∗x + y∗Y)

PA > 10.1

where the parameter name is to the left of > and its value is to the right of the >.

PA-PZ = _DOUBLE (Read)

These parameters supply the values of constants used in the expression EXP and sub-expressions FA-FZ. Any of the 26 may appear; there is no restriction on order. Using parameters allows the substitution of repeated constants using one reference. This is especially convenient for constants with many significant digits. It also allows easy modification of parameterised expressions provided the application has not been used with a different EXP in the interim. The parameter PI has a default value of 3.14159265359D0. An example of using parameters is:

EXP > SQRT(PX∗PX+PY∗PY)∗EXP(PX-PY)

PX > 2.345

PY > -0.987

where the parameter name is to the left of > and its value is to the right of the >.

PREC = LITERAL (Read)

The arithmetic precision with which the transformation functions will be evaluated when used. This may be either "_REAL" for single precision, "_DOUBLE" for double precision, or "_INTEGER" for integer precision. Elastic precisions are used, such that a higher precision will be used if the input data warrant it. So for example if PREC="_REAL", but double-precision data were to be transformed, double-precision arithmetic would actually be used. The result is reported using the chosen precision. ["_REAL"]

Results Parameters:

RESULT = LITERAL (Write)

The result of the evaluation.

Examples:

Shell usage:

The syntax in the following examples apply to the shell.

```
calc "27.3*1.26"
```

The reports the value of the expression 27.3∗1.26, *i.e.* 34.398.

```
calc exp="(pa+pb+pc+pd)/4.0" pa=$med1 pb=$med2 pc=$med3 pd=$med4
```

This reports the average of four values defined by script variables med1, med2, med3, and med4.

```
calc "42.6*pi/180"
```

This reports the value in radians of 42.6 degrees.

```
calc "(mod(PO,3)+1)/2" prec=_integer po=$count
```

This reports the value of the expression "(mod($count,3)+1)/2)" where $count is the value of the shell variable count. The calculation is performed in integer arithmetic, thus if count equals 2, the result is 1 not 1.5.

```
calc "sind(pa/fa)*fa" fa="log(abs(pb+pc))" pa=2.0e-4 pb=-1 pc=$x
```

This evaluates sind(0.0002/log(abs($x−1)))∗log(abs($x−1)) where $x is the value of the shell variable x.

ICL usage:

For ICL usage only those expressions containing parentheses need to be in quotes, though ICLitself provides the arithmetic. So the above examples would be

```
calc 27.3*1.26
```

The reports the value of the expression 27.3∗1.26, *i.e.* 34.398.

```
calc exp="(pa+pb+pc+pd)/4.0" pa=(med1) pb=(med2) pc=(med3) pd=(med4)
```

This reports the average of four values defined by ICLvariables med1, med2, med3, and med4.

```
calc 42.6*pi/180
```

This reports the value in radians of 42.6 degrees.

```
calc "(mod(PO,3)+1)/2" prec=_integer po=(count)
```

This reports the value of the expression `"(mod((count),3)+1)/2)"` where `(count)` is the value of the ICLvariable count. The calculation is performed in integer arithmetic, thus if count equals 2, the result is 1 not 1.5.

```
calc "sind(pa/fa)*fa" fa="log(abs(pb+pc))" pa=2.0e-4 pb=-1 pc=(x)
```

This evaluates sind(0.0002/log(abs((x)−1)))∗log(abs((x)−1)) where `(x)` is the value of the ICL variable x.

Implementation Status:

On OSF/1 systems an error during the calculation results in a core dump. On Solaris, undefined values are set to one. These are due to problems with the TRANSFORM infrastructure.

Related Applications :

KAPPA: MATHS.

CALPOL
Calculates polarisation parameters

Description:

This routine calculates various parameters describing the polarisation described by four intensity arrays analysed at 0°, 45°, 90°, and 135° to a reference direction. Variance values are stored in the output NDFs if all the input NDFs have variances and you give a `TRUE` value for Parameter VARIANCE.

By default, three output NDFs are created holding percentage polarisation, polarisation angle and total intensity. However, NDFs holding other quantities, such as the Stokes parameters, can also be produced by overriding the default null values associated with the corresponding parameters. The creation of any output NDF can be suppressed by supplying a null value for the corresponding parameter.

There is an option to correct the calculated values of percentage polarisation and polarised intensity to take account of the statistical bias introduced by the asymmetric distribution of percentage polarisation (see Parameter DEBIAS). This correction subtracts the variance of the percentage polarisation from the squared percentage polarisation, and uses the square root of this as the corrected percentage polarisation. The corresponding polarised intensity is then found by multiplying the corrected percentage polarisation by the total intensity. Returned variance values take no account of this correction.

Usage:

```
calpol in1 in2 in3 in4 p theta i
```

Parameters:

DEBIAS = _LOGICAL (Read)

`TRUE` if a correction for statistical bias is to be made to percentage polarisation and polarised intensity. This correction cannot be used if any of the input NDFs do not contain variance values, or if you supply a `FALSE` value for Parameter VARIANCE. `[FALSE]`

I = NDF (Write)

An output NDF holding the total intensity derived from all four input NDFs.

IN1 = NDF (Read)

An NDF holding the measured intensity analysed at an angle of 0° to the reference direction. The primary input NDF.

IN2 = NDF (Read)

An NDF holding the measured intensity analysed at an angle of 45° to the reference direction. The suggested default is the current value.

IN3 = NDF (Read)

An NDF holding the measured intensity analysed at an angle of 90° to the reference direction. The suggested default is the current value.

IN4 = NDF (Read)

An NDF holding the measured intensity analysed at an angle of 135° to the reference direction. The suggested default is the current value.

IA = NDF (Write)
An output NDF holding the total intensity derived from input NDFs IN1 and IN3. [!]

IB = NDF (Write)
An output NDF holding the total intensity derived from input NDFs IN2 and IN4. [!]

IP = NDF (Write)
An output NDF holding the polarised intensity. [!]

P = NDF (Write)
An output NDF holding percentage polarisation.

Q = NDF (Write)
An output NDF holding the normalised Stokes parameter, *Q*. [!]

U = NDF (Write)
An output NDF holding the normalised Stokes parameter, *U*. [!]

THETA = NDF (Write)
An output NDF holding the polarisation angle in degrees.

VARIANCE = _LOGICAL (Read)
TRUE if output variances are to be calculated. This parameter is only accessed if all input NDFs contain variances, otherwise no variances are generated. [TRUE]

Examples:

```
calpol m51_0 m51_45 m51_90 m51_135 m51_p m51_t m51_i ip=m51_ip
```

This example produces NDFs holding percentage polarisation, polarisation angle, total intensity and polarised intensity, based on the four NDFs M51_0, m51_45, m51_90 and m51_135.

```
calpol m51_0 m51_45 m51_90 m51_135 m51_p m51_t m51_i ip=m51_ip novariance
```

As above except that variance arrays are not computed.

```
calpol m51_0 m51_45 m51_90 m51_135 m51_p m51_t m51_i ip=m51_ip
```

As the first example except that there is a correction for statistical bias in the percentage polarisation and polarised intensity, assuming that all the input NDFs have a VARIANCE array.

```
calpol m51_0 m51_45 m51_90 m51_135 q=m51_q p=m51_p
```

This example produces NDFs holding the Stokes *Q* and *U* parameters, again based on the four NDFs M51_0, m51_45, m51_90 and m51_135.

Notes:

- A bad value will appear in the output data and variance arrays when any of the four input data values is bad, or if the total intensity in the pixel is not positive. The output variance values are also undefined when any of the four input variances is bad or negative, or any computed variance is not positive, or the percentage polarisation is not positive.
- If the four input NDFs have different pixel-index bounds, then they will be trimmed to match before being added. An error will result if they have no pixels in common.
- The output NDFs are deleted if there is an error during the formation of the polarisation parameters.
- The output NDFs obtain their QUALITY, AXIS information, and TITLE from the IN1 NDF. The following labels and units are also assigned:

I	`"Total Intensity"`	UNITS of IN1
IA	`"Total Intensity"`	UNITS of IN1
IB	`"Total Intensity"`	UNITS of IN1
IP	`"Polarised Intensity"`	UNITS of IN1
P	`"Percentage Polarisation"`	`"%"`
Q	`"Stokes Q"`	—
U	`"Stokes U"`	—
THETA	`"Polarisation Angle"`	`"Degrees"`

Related Applications :

KAPPA: VECPLOT; POLPACK: POLCAL, POLVEC; TSP.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single-precision floating point.

CARPET
Creates a cube representing a carpet plot of an image

Description:

This application creates a new three-dimensional NDF from an existing two-dimensional NDF. The resulting NDF can, for instance, be viewed with the three-dimensional iso-surface facilities of the GAIA image viewer, in order to create a display similar to a *carpet plot* of the image (the iso-surface at value zero represents the input image data values).

The first two pixel axes (x and y) in the output cube correspond to the pixel axes in the input image. The third pixel axis (z) in the output cube is proportional to data value in the input image. The value of a pixel in the output cube measures the difference between the data value implied by its z-axis position, and the data value of the corresponding pixel in the input image. Two schemes are available (see Parameter MODE): the output pixel values can be either simply the difference between these two data values, or the difference divided by the standard deviation at the corresponding pixel in the input image (as determined either from the VARIANCE component in the input NDF or by Parameter SIGMA).

Usage:

```
carpet in out [ndatapix] [range] [mode] [sigma]
```

Parameters:

IN = NDF (Read)
The input two-dimensional NDF.

MODE = LITERAL (Read)
Determines how the pixel values in the output cube are calculated.

- `"Data"` — the value of each output pixel is equal to the difference between the data value implied by its position along the data value axis, and the value of the corresponding pixel in the input image.
- `"Sigma"` — this is the same as `"Data"` except that the output pixel values are divided by the standard deviation implied either by the VARIANCE component of the input image, or by the SIGMA parameter.

`["Data"]`

NDATAPIX = _INTEGER (Read)
The number of pixels to use for the data value axis in the output cube. The pixel origin of this axis will be 1. The dynamic default is the square root of the number of pixels in the input image. This gives a fairly 'cubic' output cube. `[]`

OUT = NDF (Write)
The output three-dimensional NDF.

RANGE = LITERAL (Read)
RANGE specifies the range covered by the data value axis (*i.e.* the third pixel axis) in the output cube. The supplied string should consist of up to three sub-strings,

separated by commas. For all but the option where you give explicit numerical limits, the first sub-string must specify the method to use. If supplied, the other two sub-strings should be numerical values as described below (default values will be used if these sub-strings are not provided). The following options are available.

- lower,upper — You can supply explicit lower and upper limiting values. For example, `"10,200"` would set the lower limit on the output data axis to 10 and its upper limit to 200. No method name prefixes the two values. If only one value is supplied, the `"Range"` method is adopted. The limits must be within the dynamic range for the data type of the input NDF array component.
- `"Percentiles"` — The default values for the output data axis range are set to the specified percentiles of the input data. For instance, if the value `"Per,10,99"` is supplied, then the lowest 10% and highest 1% of the data values are beyond the bounds of the output data value axis. If only one value, p1, is supplied, the second value, p2, defaults to (100 - p1). If no values are supplied, the values default to `"5,95"`. Values must be in the range 0 to 100.
  `"Range"` — The minimum and maximum array values are used. No other sub-strings are needed by this option. Null (`!`) is a synonym for the `"Range"` method.
- `"Sigmas"` — The limits on the output data value axis are set to the specified numbers of standard deviations below and above the mean of the input data. For instance, if the supplied value is `"sig,1.5,3.0"`, then the data value axis extends from the mean of the input data minus 1.5 standard deviations to the mean plus 3 standard deviations. If only one value is supplied, the second value defaults to the supplied value. If no values are supplied, both default to `"3.0"`.

The limits adopted for the data value axis are reported unless Parameter RANGE is specified on the command line. In this case values are only calculated where necessary for the chosen method.

The method name can be abbreviated to a single character, and is case insensitive. The initial default value is `"Range"`. The suggested defaults are the current values, or `!` if these do not exist. `[current value]`

SIGMA = _REAL (Read)

The standard deviation to use if Parameter MODE is set to `"Sigma"`. If a null (`!`) value is supplied, the standard deviations implied by the VARIANCE component in the input image are used (an error will be reported if the input image does not have a VARIANCE component). If a SIGMA value is supplied, the same value is used to scale all output pixels. `[!]`

Examples:

```
carpet m31 m31-cube mode=sigma
```

Asssuming the two-dimensional NDF in file `m31.sdf` contains a VARIANCE component, this will create a three-dimensional NDF called m31-cube in which the third pixel axis corresponds to data value in NDF m31, and each output pixel value is the number of standard deviations of the pixel away from the corresponding input data value. If you then use GAIA to view the cube, an iso-surface at value zero will be a carpet plot of the data values in m31, an iso-surface at value `-1.0` will be a carpet plot showing data values one standard deviation below the m31 data values, and an iso-surface at value `+1.0` will

be a carpet plot showing data values one sigma above the m31 data values. This can help to visualise the errors in an image.

Implementation Status:

- Any VARIANCE and QUALITY components in the input image are not propagated to the output cube.

CDIV
Divides an NDF by a scalar

Description:

This application divides each pixel of an NDF by a scalar (constant) value to produce a new NDF.

Usage:

```
cdiv in scalar out
```

Parameters:

IN = NDF (Read)
Input NDF structure whose pixels are to be divided by a scalar.

OUT = NDF (Write)
Output NDF structure.

SCALAR = _DOUBLE (Read)
The value by which the NDF's pixels are to be divided.

TITLE = LITERAL (Read)
A title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

Examples:

```
cdiv a 100.0 b
```

Divides all the pixels in the NDF called a by the constant value 100.0 to produce a new NDF called b.

```
cdiv in=data1 out=data2 scalar=-38
```

Divides all the pixels in the NDF called data1 by -38 to give data2.

Related Applications :

KAPPA: ADD, CADD, CMULT, CSUB, DIV, MATHS, MULT, SUB.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is carried out using the appropriate floating-point type, but the numeric type of the input pixels is preserved in the output NDF.
- Huge NDFs are supported.

CENTROID
Finds the centroids of star-like features in an NDF

Description:

This routine takes an NDF and returns the co-ordinates of the centroids of features in its data array given approximate initial co-ordinates. A feature is a set of connected pixels which are above or below the surrounding background region. For example, a feature could be a star or galaxy on the sky, although the applications is not restricted to two-dimensional NDFs.

Four methods are available for obtaining the initial positions, selected using Parameter MODE:

- from the parameter system (see Parameter INIT);
- Using a graphics cursor to indicate the feature in a previously displayed data array (see Parameter DEVICE);
- from a specified positions list (see Parameter INCAT); or
- from a simple text file containing a list of co-ordinates (see Parameter COIN).

In the first two modes the application loops, asking for new feature co-ordinates until it is told to quit or encounters an error.

The results may optionally be written to an output positions list which can be used to pass the positions on to another application (see Parameter OUTCAT), or to a log file geared more towards human readers, including details of the input parameters (see Parameter LOGFILE).

The uncertainty in the centroid positions may be estimated if variance values are available within the supplied NDF (see Parameter CERROR).

Usage:

```
                        { init
centroid ndf [mode]     { coin=?   [search] [maxiter] [maxshift] [toler]
                        { incat=?
                      mode
```

Parameters:

CATFRAME = LITERAL (Read)

A string determining the co-ordinate Frame in which positions are to be stored in the output catalogue associated with Parameter OUTCAT. The string supplied for CATFRAME can be one of the following options.

- A domain name such as SKY, AXIS, PIXEL.
- An integer value giving the index of the required Frame.

- An IRAS90 *Sky Co-ordinate System* (SCS) values such as `"EQUAT(J2000)"` (see SUN/163).

If a null (`!`) value is supplied, the positions will be stored in the current Frame. `[!]`

CATEPOCH = _DOUBLE (Read)
The epoch at which the sky positions stored in the output catalogue were determined. It will only be accessed if an epoch value is needed to qualify the co-ordinate Frame specified by COLFRAME. If required, 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.

CERROR = _LOGICAL (Read)
If `TRUE`, errors in the centroided position will be calculated. The input NDF must contain a VARIANCE component in order to compute errors. `[FALSE]`

COIN = FILENAME (Read)
Name of a text file containing the initial guesses at the co-ordinates of features to be centroided. Only accessed if Parameter MODE is given the value `"File"`. Each line should contain the formatted axis values for a single position, in the current Frame of the NDF. Axis values can be separated by spaces, tabs or commas. The file may contain comment lines with the first character `#` or `!`.

DESCRIBE = _LOGICAL (Read)
If `TRUE`, a detailed description of the co-ordinate Frame in which the centroided positions will be reported is displayed before the positions themselves. [current value]

DEVICE = DEVICE (Read)
The graphics device which is to be used to give the initial guesses at the centroid positions. Only accessed if Parameter MODE is given the value `"Cursor"`. [Current graphics device]

GUESS = _LOGICAL (Read)
If `TRUE`, then the supplied guesses for the centroid positions will be included in the screen and log file output, together with the accurate positions. [current value]

INCAT = FILENAME (Read)
A catalogue containing a positions list giving the initial guesses at the centroid positions, such as produced by applications CURSOR, LISTMAKE. Only accessed if Parameter MODE is given the value `"Catalogue"`.

INIT = LITERAL (Read)
An initial guess at the co-ordinates of the next feature to be centroided, in the current co-ordinate Frame of the NDF (supplying a colon `":"` will display details of the current co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas. INIT is only accessed if parameter MODE is given the value `"Interface"`. If the initial co-ordinates are supplied on the command line only one centroid will be found; otherwise the application will ask for further guesses, which may be terminated by supplying the null value (`!`).

LOGFILE = FILENAME (Read)
Name of the text file to log the results. If null, there will be no logging. Note this is intended for the human reader and is not intended for passing to other applications. `[!]`

MARK = LITERAL (Read)
Only accessed if Parameter MODE is given the value "Cursor". It indicates which positions are to be marked on the screen using the marker type given by Parameter MARKER. It can take any of the following values.

- "Initial": The position of the cursor when the mouse button is pressed is marked.
- "Centroid": The corresponding centroid position is marked.
- "None": No positions are marked.

[current value]

MARKER = _INTEGER (Read)
This parameter is only accessed if Parameter MARK is set TRUE. It specifies the type of marker with which each cursor position should be marked, and should be given as an integer PGPLOT marker type. For instance, 0 gives a box, 1 gives a dot, 2 gives a cross, 3 gives an asterisk, 7 gives a triangle. The value must be larger than or equal to −31. [current value]

MAXITER = _INTEGER (Read)
Maximum number of iterations to be used in the search. It must be in the range 1–9. The dynamic default is 3. [9]

MAXSHIFT() = _REAL (Read)
Maximum shift in each dimension allowed between the guess and output positions in pixels. Each must lie in the range 0.0–26.0. If only a single value is given, then it will be duplicated to all dimensions. The dynamic default is half of SEARCH + 1. [9.0]

MODE = LITERAL (Read)
The mode in which the initial co-ordinates are to be obtained. The supplied string can be one of the following values.

- "Interface" — positions are obtained using Parameter INIT.
- "Cursor" — positions are obtained using the graphics cursor of the device specified by Parameter DEVICE.
- "Catalogue" — positions are obtained from a positions list using Parameter INCAT.
- "File" — positions are obtained from a text file using Parameter COIN.

[current value]

NDF = NDF (Read)
The NDF structure containing the data array to be analysed. In cursor mode (see Parameter MODE), the run-time default is the displayed data, as recorded in the graphics database. In other modes, there is no run-time default and the user must supply a value. []

NSIM = _INTEGER (Read)
The number of simulations or realisations using the variance information in order to estimate the error in the centroid position. The uncertainty in the centroid error decreases as one over the square root of NSIM. The range of acceptable values is 3–10000. [100]

OUTCAT = FILENAME (Write)
The output catalogue in which to store the centroided positions. If a null value (!) is supplied, no output catalogue is produced. See also Parameter CATFRAME. [!]

PLOTSTYLE = GROUP (Read)
A group of attribute settings describing the style to use when drawing the graphics markers specified by Parameter MARK.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported). [current value]

POSITIVE = _LOGICAL (Read)
TRUE, if array features are positive above the background. [TRUE]

SEARCH() = _INTEGER (Read)
Size in pixels of the search box to be used. If only a single value is given, then it will be duplicated to all dimensions so that a square, cube or hypercube region is searched. Each value must be odd and lie in the range 3–51. [9]

TITLE = LITERAL (Read)
A title to store with the output catalogue specified by Parameter OUTCAT, and to display before the centroid positions are listed. If a null (!) value is supplied, the title is taken from any input catalogue specified by Parameter INCAT, or is a fixed string including the name of the NDF. [!]

TOLER = _REAL (Read)
Accuracy in pixels required in centroiding. Iterations will stop when the shift between successive centroid positions is less than the accuracy. The accuracy must lie in the range 0.0–2.0. [0.05]

Results Parameters:

CENTRE = LITERAL (Write)
The formatted co-ordinates of the last centroid position, in the current Frame of the NDF.

ERROR = LITERAL (Write)
The errors associated with the position written to Parameter CENTRE.

XCEN = LITERAL (Write)
The formatted x co-ordinate of the last centroid position, in the current co-ordinate Frame of the NDF.

XERR = LITERAL (Write)
The error associated with the value written to Parameter XCEN.

YCEN = LITERAL (Write)
The formatted y co-ordinate of the last centroid position, in the current co-ordinate Frame of the NDF.

YERR = LITERAL (Write)
The error associated with the value written to Parameter YCEN.

Examples:

```
centroid cluster cu
```

This finds the centroids in the NDF called cluster via the graphics cursor on the current graphics device.

```
centroid cluster cu search=21 mark=ce plotstyle='colour=red'
```

This finds the centroids in the NDF called cluster via the graphics cursor on the current graphics device. The search box for the centroid is 21 pixels in each dimension. The centroid positions are marked using a red symbol.

```
centroid cluster i "21.7,5007.1"
```

This finds the centroid of the object in the two-dimensional NDF called cluster around the current Frame co-ordinate (21.7,5007.1).

```
centroid arp244(6,,) i "40,30" toler=0.01
```

This finds the two-dimensional centroid of the feature near pixel (6,40,30) in the three-dimensional NDF called arp244 (assuming the current co-ordinate Frame of the NDF is PIXEL). The centroid must be found to 0.01 pixels.

```
centroid cluster cu xcen=(xp) ycen=(yp)
```

This finds the centroid of an object in the two-dimensional NDF called cluster using a graphics cursor, and writes the centroid co-ordinates to ICL variables XP and YP for use in other applications.

```
centroid cluster mode=file coin=objects.dat logfile=centroids.log
```

This finds the centroids in the NDF called cluster. The initial positions are given in the text file objects.dat in the current co-ordinate Frame. A log of the input parameter values, initial and centroid positions is written to the text file `centroids.log`.

```
centroid cluster mode=cat incat=a outcat=b catframe=ecl
```

This example reads the initial guess positions from the positions list in file a.FIT, and writes the accurate centroid positions to positions list file b.FIT, storing the output positions in ecliptic co-ordinates. The input file may, for instance, have been created using the application CURSOR.

Notes:

- All positions are supplied and reported in the current co-ordinate Frame of the NDF. A description of the co-ordinate Frame being used is given if Parameter DESCRIBE is set to a `TRUE` value. Application WCSFRAME can be used to change the current co-ordinate Frame of the NDF before running this application if required.
- In Cursor or Interface mode, only the first 200 supplied positions will be stored in the output catalogue. Any further positions will be displayed on the screen but not stored in the output catalogue.
- The centroid positions are not displayed on the screen when the message filter environment variable MSG_FILTER is set to `QUIET`. The creation of output parameters and files is unaffected by MSG_FILTER.

Estimation of Centroid Positions :

Each centroid position is obtained by projecting the data values within a search box centred on the supplied position, on to each axis in turn. This forms a set of profiles for the feature, one for each axis. An estimate of the background at each point in these profiles is made and subtracted from the profile. This flattens the profile backgrounds, removing any slope in the data. Once the profiles have been flattened in this way, and estimate of the background noise in each is made. The centroid of the feature is then found using only the data above the noise level.

Successive estimates of the centroid position are made by using the previous estimate of the centroid as the initial position for another estimation. This loop is repeated up to a maximum number of iterations, though it normally terminates when a desired accuracy has been achieved.

The achieved accuracy is affected by noise, and the presence of non-Gaussian or overlapping features, but typically an accuracy better than 0.1 pixel is readily attainable for stars. The error in the centroid position may be estimated by a Monte-Carlo method using the data variance to generate realisations of the data about the feature (see Parameter CERROR). Each realisation is processed identically to the actual data, and statistics are formed to derive the standard deviations.

Related Applications :

KAPPA: PSF, CURSOR, LISTSHOW, LISTMAKE.

Implementation Status:

- The processing of bad pixels and all non-complex numeric data types is supported.

CHAIN
Concatenates a series of vectorized NDFs

Description:

This application concatenates a series of NDFs in the order supplied and treated as vectors, to form a one-dimensional output NDF. The dimensions of the NDFs may be different, and indeed so may their dimensionalities.

Usage:

```
chain in c1 [c2] [c3] ...  [c25] out=?
```

Parameters:

IN = NDF (Read)
The base NDF after which the other input NDFs will be concatenated.

OUT = NDF (Write)
The one-dimensional NDF resulting from concatenating the input NDFs.

C1-C25 = NDF (Read)
The NDFs to be concatenated to the base NDF. The NDFs are joined in the order C1, C2, ... C25. There can be no missing NDFs, *e.g.* in order for C3 to be processed there must be a C2 given as well. A null value (`!`) indicates that there is no NDF. NDFs C2 to C25 are defaulted to `!`. At least one NDF must be pasted, therefore C1 may not be null.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (`!`) propagates the title from the base NDF to the output NDF. `[!]`

Examples:

```
chain obs1 obs2 out=stream
```

This concatenates the NDF called obs2 on to the arrays in the NDF called obs1 to produce the one-dimensional NDF stream.

```
chain c1=obs2 c2=obs1 in=obs3 out=stream
```

This concatenates the NDF called obs2 on to the arrays in the NDF called obs3, and then concatenates the arrays from obs1 to them to produce the one-dimensional NDF stream.

Related Applications :

KAPPA: PASTE, RESHAPE.

Implementation Status:

- This routine correctly processes the DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, and HISTORY, components of an NDF data structure and propagates all extensions. Propagation is from the base NDF. WCS, and AXIS information is lost.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

CHANMAP
Creates a channel map from a cube NDF by compressing slices along a nominated axis

Description:

This application creates a two-dimensional channel-map image from a three-dimensional NDF. It collapses along a nominated pixel axis in a series of slices. The collapsed slices are tiled with no margins to form the output image. This grid of channel maps is filled from left to right, and bottom to top. A specified range of axis values can be used instead of the whole axis (see Parameters LOW and HIGH). The number of channels and their arrangement into an image is controlled through Parameters NCHAN and SHAPE.

For each output pixel, all corresponding input pixel values between the channel bounds of the nominated axis to be collapsed are combined together using one of a selection of estimators, including a mean, mode, or median, to produce the output pixel value.

Usage:

```
chanmap in out axis nchan shape [low] [high] [estimator] [wlim]
```

Parameters:

AXIS = LITERAL (Read)

The axis along which to collapse the NDF. This can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If the axes of the current Frame are not parallel to the NDF pixel axes, then the pixel axis which is most nearly parallel to the specified current Frame axis will be used.

CLIP = _REAL (Read)

The number of standard deviations about the mean at which to clip outliers for the `"Mode"`, `"Cmean"` and `"Csigma"` statistics (see Parameter ESTIMATOR). The application first computes statistics using all the available pixels. It then rejects all those pixels whose values lie beyond CLIP standard deviations from the mean and will then re-evaluate the statistics. For `"Cmean"` and `"Csigma"` there is currently only one iteration, but up to seven for `"Mode"`.

The value must be positive. `[3.0]`

ESTIMATOR = LITERAL (Read)

The method to use for estimating the output pixel values. It can be one of the following options.

- `"Mean"` — Mean value
- `"WMean"` — Weighted mean in which each data value is weighted by the reciprocal of the associated variance. (2)
- `"Mode"` — Modal value (4)
- `"Median"` — Median value. Note that this is extremely memory and CPU intensive for large datasets; use with care! If strange things happen, use `"Mean"`. (3)
- `"Absdev"` — Mean absolute deviation from the unweighted mean. (2)
- `"Cmean"` — Sigma-clipped mean. (4)
- `"Csigma"` — Sigma-clipped standard deviation. (4)
- `"Comax"` — Co-ordinate of the maximum value.
- `"Comin"` — Co-ordinate of the minimum value.
- `"FBad"` — Fraction of bad pixel values.
- `"FGood"` — Fraction of good pixel values.
- `"Integ"` — Integrated value, being the sum of the products of the value and pixel width in world co-ordinates. Note that for sky co-ordinates the width is measured in radians.
- `"Iwc"` — Intensity-weighted co-ordinate, being the sum of each value times its co-ordinate, all divided by the integrated value (see the `"Integ"` option).
- `"Iwd"` — Intensity-weighted dispersion of the co-ordinate, normalised like `"Iwc"` by the integrated value. (4)
- `"Max"` — Maximum value.
- `"Min"` — Minimum value.
- `"NBad"` — Count of bad pixel values.
- `"NGood"` — Count of good pixel values.
- `"Rms"` — Root-mean-square value. (4)
- `"Sigma"` — Standard deviation about the unweighted mean. (4)
- `"Sum"` — The total value.

The selection is restricted if each channel contains three or fewer pixels. For instance, measures of dispersion like `"Sigma"` and `"Iwd"` are meaningless for single-pixel channels. The minimum number of pixels per channel for each estimator is given in parentheses in the list above. Where there is no number, there is no restriction. If you supply an unavailable option, you will be informed, and presented with the available options. `["Mean"]`

HIGH = LITERAL (Read)

Together with Parameter LOW, this parameter defines the range of values for the axis specified by Parameter AXIS to be divided into channels. For example, if AXIS is 3 and the current Frame of the input NDF has axes RA/DEC/Wavelength, then a wavelength value should be supplied. If, on the other hand, the current Frame in the NDF was the PIXEL Frame, then a pixel co-ordinate value would be required for the third axis (note, the pixel with index I covers a range of pixel co-ordinates from $(I - 1)$ to I).

Note, HIGH and LOW should not be equal. If a null value (`!`) is supplied for either HIGH or LOW, the entire range of the axis fragmented into channels. `[!]`

IN = NDF (Read)
The input NDF. This must have three dimensions.

LOW = LITERAL (Read)
Together with Parameter HIGH this parameter defines the range of values for the axis specified by Parameter AXIS to be divided into channels. For example, if AXIS is 3 and the current Frame of the input NDF has axes RA/DEC/Frequency, then a frequency value should be supplied. If, on the other hand, the current Frame in the NDF was the PIXEL Frame, then a pixel co-ordinate value would be required for the third axis (note, the pixel with index I covers a range of pixel co-ordinates from $(I-1)$ to I).

Note, HIGH and LOW should not be equal. If a null value (`!`) is supplied for either HIGH or LOW, the entire range of the axis fragmented into channels. `[!]`

NCHAN = _INTEGER (Read)
The number of channels to appear in the channel map. It must be a positive integer up to the lesser of 100 or the number of pixels along the collapsed axis.

OUT = NDF (Write)
The output NDF.

SHAPE = _INTEGER (Read)
The number of channels along the x axis of the output NDF. The number along the y axis will be (NCHAN-1)/SHAPE. A null value (`!`) asks the application to select a shape. It will generate one that gives the most square output NDF possible. The value must be positive and no more than the value of Parameter NCHAN.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (`!`) propagates the title from the input NDF to the output NDF. `[!]`

USEAXIS = GROUP (Read)
USEAXIS is only accessed if the current co-ordinate Frame of the input NDF has more than three axes. A group of three strings should be supplied specifying the three axes which are to be retained in a collapsed slab.

Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (`!`) value is supplied, the axes with the same indices as the three used pixel axes within the NDF are used. `[!]`

WLIM = _REAL (Read)
If the input NDF contains bad pixels, then this parameter may be used to determine the number of good pixels which must be present within the range of collapsed input pixels before a valid output pixel is generated. It can be used, for example, to prevent

output pixels from being generated in regions where there are relatively few good pixels to contribute to the collapsed result.

WLIM specifies the minimum fraction of good pixels which must be present in order to generate a good output pixel. If this specified minimum fraction of good input pixels is not present, then a bad output pixel will result, otherwise a good output value will be calculated. The value of this parameter should lie between 0.0 and 1.0 (the actual number used will be rounded up if necessary to correspond to at least one pixel). [0.3]

Examples:

```
chanmap cube chan4 lambda 4 2 4500 4550
```

The current Frame in the input three-dimensional NDF called cube has axes with labels "RA", "DEC" and "Lambda", with the lambda axis being parallel to the third pixel axis. The above command extracts four slabs of the input cube between wavelengths 4500 and 4550 Ångstroms, and collapses each slab, into a single two-dimensional array with RA and DEC axes forming a channel image. Each channel image is pasted into a 2×2 grid within the output NDF called chan4. Each pixel in the output NDF is the mean of the corresponding input pixels with wavelengths in 12.5-Ångstrom bins.

```
chanmap in=cube out=chan4 axis=3 low=4500 high=4550 nchan=4 shape=2
```

The same as above except the axis to collapse along is specified by index (3) rather than label (lambda), and it uses keywords rather than positional parameters.

```
chanmap cube chan4 3 4 2 9.0 45.0
```

This is the same as the above examples, except that the current Frame in the input NDF has been set to the PIXEL Frame (using WCSFRAME), and so the high and low axis values are specified in pixel co-ordinates instead of Ångstroms, and each channel covers nine pixels. Note the difference between floating-point pixel co-ordinates, and integer pixel indices (for instance the pixel with index 10 extends from pixel co-ordinate 9.0 to pixel co-ordinate 10.0).

```
chanmap in=zcube out=vel7 axis=1 low=-30 high=40 nchan=7 shape=!
estimator=max
```

This command assumes that the zcube NDF has a current co-ordinate system where the first axis is radial velocity (perhaps selected using WCSFRAME and WCSATTRIB), and the second and third axes are "RA", and "DEC". It extracts seven velocity slabs of the input cube between -30 and $+40$ km/s, and collapses each slab, into a single two-dimensional array with RA and DEC axes forming a channel image. Each channel image is pasted into a default grid (likely 4×2) within the output NDF called vel7. Each pixel in the output NDF is the maximum of the corresponding input pixels with velocities in 10-km/s bins.

Notes:

- The collapse is always performed along one of the pixel axes, even if the current Frame in the input NDF is not the PIXEL Frame. Special care should be taken if the current-Frame axes are not parallel to the pixel axes. The algorithm used to choose the pixel axis and the range of values to collapse along this pixel axis proceeds as follows.

  The current-Frame co-ordinates of the central pixel in the input NDF are determined (or some other point if the co-ordinates of the central pixel are undefined). Two current-Frame positions are then generated by substituting in turn into this central

position each of the HIGH and LOW values for the current-Frame axis specified by Parameter AXIS. These two current-Frame positions are transformed into pixel co-ordinates, and the projections of the vector joining these two pixel positions on to the pixel axes are found. The pixel axis with the largest projection is selected as the collapse axis, and the two end points of the projection define the range of axis values to collapse.

- The WCS of the output NDF retains the three-dimensional co-ordinate system of the input cube for every tile, except that each tile has a single representative mean co-ordinate for the collapsed axis.
- The slices may have slightly different pixel depths depending where the boundaries of the channels lie in pixel co-ordinates. Excise care interpreting estimators like `"Sum"` or ensure equal numbers of pixels in each channel.

Related Applications :

KAPPA: COLLAPSE, CLINPLOT.

Implementation Status:

- This routine correctly processes the DATA, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF; and propagates all extensions. AXIS and QUALITY are not propagated.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- The origin of the output NDF is at (1,1).
- Huge NDFs are supported.

CHPIX
Replaces the values of selected pixels in an NDF

Description:

This application replaces selected elements of an NDF array component with specified values.

Two methods are available for obtaining the regions and replacement values, selected through Parameter MODE:

- from the parameter system (see Parameters SECTION and NEWVAL). The task loops until there are no more elements to change, indicated by a null value in response to a prompt. For non-interactive processing, supply the value of Parameter NEWVAL on the command line.
- The second approach uses a text file, that is especially beneficial where there are too many section and value pairs to enter manually. The file should contain two space-separated columns; the first column is the NDF section to replace, and the second supplies the value to insert into the section (see Parameter FILE).

Usage:

```
chpix in out section newval [comp]
```

Parameters:

COMP = LITERAL (Read)

The name of the NDF array component to be modified. The options are: `"Data"`, `"Error"`, `"Quality"` or `"Variance"`. `"Error"` is the alternative to `"Variance"` and causes the square of the supplied replacement value to be stored in the output VARIANCE array. `["Data"]`

FILE = FILENAME (Read)

Name of a text file containing the sections and replacement values. It is only accessed if Parameter MODE is given the value `"File"`. Each line should contain an NDF section of a region (see Parameter SECTION), then one or more spaces, followed by the replacement value. The value is either numeric or `"Bad"`, the latter requests that the bad value be inserted into the section. The file may contain comment lines with the first character `#` or `!`.

IN = NDF (Read)

NDF structure containing the array component to be modified.

MODE = LITERAL (Read)

The mode in which the sections and replacement values are to be obtained. The supplied string can be one of the following values.

- `"Interface"` — sections and values are obtained via Parameters SECTION and NEWVAL respectively.
- `"File"` — sections and values are obtained from a text file specified by Parameter FILE.

`["Interface"]`

NEWVAL = LITERAL (Read)
Value to substitute in the output array element or elements. The range of allowed values depends on the data type of the array being modified. NEWVAL="Bad" instructs that the bad value appropriate for the array data type be substituted. Placing NEWVAL on the command line permits only one section to be replaced. If there are multiple replacements, a null value (!) terminates the loop. If the section being modified contains only a single pixel, then the original value of that pixel is used as the suggested default value.

OUT = NDF (Write)
Output NDF structure containing the modified version of the array component.

SECTION = LITERAL (Read)
The elements to change. This is defined as an NDF section, so that ranges can be defined along any axis, and be given as pixel indices or axis (data) co-ordinates. So for example "3,4,5" would select the pixel at (3,4,5); "3:5," would replace all elements in Columns 3 to 5; ",4" replaces Line 4. See Section 9 for details. A null value (!) terminates the loop during multiple replacements.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (!) propagates the title from the input NDF to the output NDF. [!]

Results Parameters:

OLDVAL = LITERAL (Write)
If the section being modified contains only a single pixel, then the original value of that pixel is written out to this output parameter.

Examples:

```
chpix rawspec spectrum 55 100
```

Assigns the value 100 to the pixel at index 55 within the one-dimensional NDF called rawspec, creating the output NDF called spectrum.

```
chpix rawspec spectrum 10:19 0 error
```

Assigns the value 0 to the error values at indices 10 to 19 within the one-dimensional NDF called rawspec, creating the output NDF called spectrum. The rawspec dataset must have a variance compoenent.

```
chpix in=rawimage out=galaxy section="~20,100:109" newval=bad
```

Assigns the bad value to the pixels in the section ~20,100:109 within the two-dimensional NDF called rawimage, creating the output NDF called galaxy. This section is the central 20 pixels along the first axis, and pixels 110 to 199 along the second.

```
chpix in=zzcha out=zzcha_c section="45,21," newval=-1
```

Assigns value −1 to the pixels at index (45, 21) within all planes of the three-dimensional NDF called zzcha, creating the output NDF called zzcha_c.

```
chpix harpcube harpmasked mode=file file=badbaseline.txt
```

T his reads the text file called `badbaseline.txt` to obtain the editing commands to be applied to the NDF called harpcube to form the NDF harpmasked.

Related Applications :

KAPPA: ARDMASK, FILLBAD, GLITCH, NOMAGIC, REGIONMASK, SEGMENT, SETMAGIC, SUBSTITUTE, ZAPLIN; FIGARO: CSET, ICSET, NCSET, TIPPEX.

Implementation Status:

- The routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF; and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

CLINPLOT
Draws a spatial grid of line plots for an axis of a cube NDF

Description:

This application displays a three-dimensional NDF as a series of line plots of array value against position, arranged on a uniform spatial grid and plotted on the current graphics device. The vertical axis of each line plot represents array value, and the horizontal axis represents position along a chosen axis (see Parameter USEAXIS). All the line plots have the same axis limits.

This application will typically be used to display a grid of spectra taken from a cube in which the current WCS Frame includes one spectral axis (*e.g.* frequency) and two spatial axes (*e.g.* RA and Dec). For this reason the following documentation refers to the 'spectral axis' and the 'spatial axes'. However, cubes containing other types of axes can also be displayed, and references to 'spectral' and 'spatial' axes should be interpreted appropriately.

A rectangular grid of *NX* by *NY* points (see Parameters NX and NY) is defined over the spatial extent of the cube, and a spectrum is drawn at each such point. If *NX* and *NY* equal the spatial dimensions of the cube (which is the default for spatial axes of fewer than 31 pixels), then one spectrum is drawn for every spatial pixel in the cube. For speed, the spectrum will be binned up so that the number of elements in the spectrum does not exceed the horizontal number of device pixels available for the line plot.

Annotated axes for the spatial co-ordinates may be drawn around the grid of line plots (see Parameter AXES). The appearance of these and the space they occupy may be controlled in detail (see Parameters STYLE and MARGIN).

The plot may take several different forms such as a `"join-the-dots"` plot, a `"staircase"` plot, a `"chain"` plot (see Parameter MODE). The plotting style (colour, founts, text size, *etc.*) may be specified in detail using Parameter SPECSTYLE.

The data value at the top and bottom of each line plot can be specified using Parameters YBOT and YTOP. The defaults can be selected in several ways including percentiles (see Parameter LMODE).

The current picture is usually cleared before plotting the new picture, but Parameter CLEAR can be used to prevent this, allowing the plot (say) to be drawn over the top of a previously displayed grey-scale image.

The range and nature of the vertical and horizontal axes in each line plot can be displayed in a key to the right of the main plot (see Parameter KEY). Also, an option exists to add numerical labels to the first (*i.e.* bottom-left) line plot, see Parameter REFLABEL. However, due to the nature of the plot, the text used may often be too small to read.

Usage:

```
clinplot ndf [useaxis] [device] [nx] [ny]
```

Parameters:

ALIGN = _LOGICAL (Read)
Controls whether or not the spectra should be aligned spatially with an existing data plot. If ALIGN is TRUE, each spectrum will be drawn in a rectangular cell that is centred on the corresponding point on the sky. This may potentially cause the spectra to overlap, depending on their spatial separation. If ALIGN is FALSE, then the spectra are drawn in a regular grid of equal-sized cells that cover the entire picture. This may cause them to be drawn at spatial positions that do not correspond to their actual spatial positions within the supplied cube. The dynamic default is TRUE if Parameter CLEAR is TRUE and there is an existing DATA picture on the graphics device. []

AXES = _LOGICAL (Read)
TRUE if labelled and annotated axes describing the spatial are to be drawn around the outer edges of the plot. The appearance of the axes can be controlled using the STYLE parameter. The dynamic default is to draw axes only if the CLEAR parameter indicates that the graphics device is not being cleared. []

BLANKEDGE = _LOGICAL (Read)
If TRUE then no tick marks or labels are placed on the edges of line plots that touch the outer spatial axes (other edges that do not touch the outer axes will still be annotated). This can avoid existing tick marks being over-written when drawing a grid of spectra over the top of a picture that includes annotated axes. The dynamic default is TRUE if and only if the graphics device is not being cleared (*i.e.* Parameter CLEAR is FALSE) and no spatial axes are being drawn (*i.e.* Parameter AXES is FALSE). []

CLEAR = _LOGICAL (Read)
If TRUE the current picture is cleared before the plot is drawn. If FALSE, then the display is left uncleared and an attempt is made to align the spatial axes of the new plot with any spatial axes of the existing plot. Thus, for instance, a while light image may be displayed using DISPLAY, and then spectra drawn over the top of the image using this application. [TRUE]

COMP = LITERAL (Read)
The NDF array component to be displayed. It may be "Data", "Quality", "Variance", or "Error" (where "Error" is an alternative to "Variance" and causes the square root of the variance values to be displayed). If "Quality" is specified, then the quality values are treated as numerical values (in the range 0 to 255). ["Data"]

DEVICE = DEVICE (Read)
The name of the graphics device used to display the cube. [Current graphics device]

FILL = _LOGICAL (Read)
If FILL is set to TRUE, then the display will be 'stretched' to fill the current picture in both directions. This can be useful to elongate the spectra to reveal more detail by using more of the display surface at the cost of different spatial scales, and when the spatial axes have markedly different dimensions. The dynamic default is TRUE if either of the spatial dimensions is one. and FALSE otherwise. []

KEY = _LOGICAL (Read)
If TRUE, then a 'key' will be drawn to the right of the plot. The key will include information about the vertical and horizontal axes of the line plots, including the maximum and minimum value covered by the axis and the quantity represented by the axis. The appearance of this key can be controlled using Parameter KEYSTYLE, and its position can be controlled using Parameter KEYPOS. [TRUE]

KEYPOS() = _REAL (Read)

Two values giving the position of the key. The first value gives the gap between the right-hand edge of the grid plot and the left-hand edge of the key (0.0 for no gap, 1.0 for the largest gap). The second value gives the vertical position of the top of the key (1.0 for the highest position, 0.0 for the lowest). If the second value is not given, the top of the key is placed level with the top of the grid plot. Both values should be in the range 0.0 to 1.0. If a key is produced, then the right-hand margin specified by Parameter MARGIN is ignored. [current value]

KEYSTYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the key (see Parameter KEY).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the text in the key can be changed by setting new values for the attributes Colour(Strings), Font(Strings) *etc.* [current value]

LMODE = LITERAL (Read)

LMODE specifies how the defaults for Parameters YBOT and YTOP (the lower and upper limit of the vertical axis of each line plot) should be found. The supplied string should consist of up to three sub-strings, separated by commas. The first sub-string must specify the method to use. If supplied, the other two sub-strings should be numerical values as described below (default values will be used if these sub-strings are not provided). The following methods are available.

- `"Range"` — The minimum and maximum data values in the supplied cube are used as the defaults for YBOT and YTOP. No other sub-strings are needed by this option.
- `"Extended"` — The minimum and maximum data values in the cube are extended by percentages of the data range, specified by the second and third sub-strings. For instance, if the value `"Ex,10,5"` is supplied, then the default for YBOT is set to the minimum data value minus 10% of the data range, and the default for YTOP is set to the maximum data value plus 5% of the data range. If only one

value is supplied, the second value defaults to the supplied value. If no values are supplied, both values default to "2.5".

- "Percentile" — The default values for YBOT and YTOP are set to the specified percentiles of the data in the supplied cube. For instance, if the value "Per,10,99" is supplied, then the default for YBOT is set so that the lowest 10% of the plotted points are off the bottom of the plot, and the default for YTOP is set so that the highest 1% of the points are off the top of the plot. If only one value, $p1$, is supplied, the second value, $p2$, defaults to $(100 - p1)$. If no values are supplied, the values default to "5,95".
- "Sigma" — The default values for YBOT and YTOP are set to the specified numbers of standard deviations below and above the mean of the data. For instance, if the value "sig,1.5,3.0" is supplied, then the default for YBOT is set to the mean of the data minus 1.5 standard deviations, and the default for YTOP is set to the mean plus 3 standard deviations. If only one value is supplied, the second value defaults to the supplied value. If no values are provided both default to "3.0".

The method name can be abbreviated to a single character, and is case insensitive. The initial value is "Range". [current value]

MARGIN(4) = _REAL (Read)

The widths of the margins to leave around the outer spatial axes for axis annotations, given as fractions of the corresponding dimension of the current picture. The actual margins used may be increased to preserve the aspect ratio of the data. 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. If a null (!) value is supplied, the value used is (for all edges); 0.15 if annotated axes are being produced; and 0.0 otherwise. The initial default is null. [current value]

MARKER = _INTEGER (Read)

This parameter is only accessed if Parameter MODE is set to "Chain" or "Mark". It specifies the symbol with which each position should be marked, and should be given as an integer PGPLOT marker type. For instance, 0 gives a box, 1 gives a dot, 2 gives a cross, 3 gives an asterisk, 7 gives a triangle. The value must be larger than or equal to −31. [current value]

MODE = LITERAL (Read)

Specifies the way in which data values are represented. MODE can take the following values.

- "Histogram" — An histogram of the points is plotted in the style of a 'staircase' (with vertical lines only joining the y-axis values and not extending to the base of the plot). The vertical lines are placed midway between adjacent x-axis positions. Bad values are flanked by vertical lines to the lower edge of the plot.
- "GapHistogram" — The same as the "Histogram" option except bad values are not flanked by vertical lines to the lower edge of the plot, leaving a gap.
- "Line" — The points are joined by straight lines.
- "Point" — A dot is plotted at each point.
- "Mark" — Each point is marker with a symbol specified by Parameter MARKER.

- "Chain" — A combination of "Line" and "Mark".

The initial default is "Line". [current value]

NDF = NDF (Read)
The input NDF structure containing the data to be displayed. It should have three significant axes, *i.e.* whose dimensions are greater than 1.

NX = _INTEGER (Read)
The number of spectra to draw in each row. The spectra will be equally spaced over the bounds of the *x* pixel axis. The dynamic default is the number of pixels along the *x* axis of the NDF, so long as this value is no more than 30. If the *x* axis spans more than 30 pixels, then the dynamic default is 30 (meaning that some spatial pixels will be ignored). []

NY = _INTEGER (Read)
The number of spectra to draw in each column. The spectra will be equally spaced over the bounds of the *y* pixel axis. The dynamic default is the number of pixels along the *y* axis of the NDF, so long as this value is no more than 30. If the *y* axis spans more than 30 pixels, then the dynamic default is 30 (meaning that some spatial pixels will be ignored). []

REFLABEL = _LOGICAL (Read)
If TRUE then the first line plot (*i.e.* the lower-left spectrum) will be annotated with numerical and textual labels describing the two axes. Note, due to the small size of the line plot, such text may be too small to read on some graphics devices. [current value]

SPECAXES = _LOGICAL (Read)
TRUE if axes are to be drawn around each spectrum. The appearance of the axes can be controlled using the SPECSTYLE parameter. [TRUE]

SPECSTYLE = GROUP (Read)
A group of attribute settings describing the plotting style to use when drawing the axes and data values in the spectrum line plots.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

By default the axes have interior tick marks, and are without labels and a title to avoid overprinting on adjacent plots.

The appearance of the data values is controlled by the attributes Colour(Curves), Width(Curves), *etc.* (the synonym Lines may be used in place of Curves). [current value]

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the annotated outer spatial axes (see Parameter AXES).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported). [current value]

USEAXIS = LITERAL (Read)

The WCS axis that will appear along the horizontal axis of each line plot (the other two axes will be used as the spatial axes). The axis can be specified using one of the following options.

- Its integer index within the current Frame of the NDF (in the range 1 to 3 in the current frame).
- Its Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. The dynamic default is the index of any spectral axis found in the current Frame of the NDF. []

YBOT = _REAL (Read)

The data value for the bottom edge of each line plot. The dynamic default is chosen in a manner determined by Parameter LMODE. []

YTOP = _REAL (Read)

The data value for the top edge of each line plot. The dynamic default is chosen in a manner determined by Parameter LMODE. []

Examples:

```
clinplot cube useaxis=3
```

Plots a set of line plots of data values versus position along the third axis for the three-dimensional NDF called cube on the current graphics device. Axes are drawn around the grid of plots indicating the spatial positions in the current co-ordinate Frame. The third axis may not be spectral and the other two axes need not be spatial.

```
clinplot cube margin=0.1
```

As above, but if a search locates a spectral axis in the world co-ordinate system, this is plotted along the horizontal of the line plots, and the other axes are deemed to be spatial. Also the margin for the spatial axes is reduced to 0.1 to allow more room for the grid of line plots.

```
clinplot map(~5,~5,) useaxis=3 noaxes
```

Plots data values versus position for the central 5-by-5 pixel region of the three-dimensional NDF called map on the current graphics device. No spatial axes are drawn.

```
clinplot map(~5,~5,) useaxis=3 noaxes device=ps_l mode=hist
```

As the previous example but now the output goes to a text file (`pgplot.ps`) which can be printed on a PostScript printer and the data are plotted in histogram form.

```
clinplot nearc v style="title=Ne Arc variance" useaxis=1 reflabel=f
```

Plots variance values versus position along Axis 1, for each spatial position in dimensions two and three, for the three dimensional NDF called nearc on the current graphics device. The plot has a title of `"Ne Arc variance"`. No labels are drawn around the lower-left line plot.

```
clinplot ndf=speccube noclear specstyle="colour(curves)=blue"
```

Plots data values versus pixel co-ordinate at each spatial position for the three-dimensional NDF called speccube on the current graphics device. The plot is drawn over any existing plot and inherits the spatial bounds of the previous plot. The data are drawn in blue, probably to distinguish it from the previous plot drawn in a different colour.

Notes:

- If no Title is specified via the STYLE parameter, then the TITLE component in the NDF is used as the default title for the annotated axes. Should the NDF not have a TITLE component, then the default title is instead taken from current co-ordinate

Frame in the NDF, unless this attribute has not been set explicitly, whereupon the name of the NDF is used as the default title.

- If all the data values at a spatial position are bad, no line plot is drawn at that location.
- The application stores a number of pictures in the graphics database in the following order: a FRAME picture containing the annotated axes, data plots, and optional key; a KEY picture to store the key if present; and a DATA picture containing just the data plots. The world co-ordinates in the DATA picture will correspond to the offset along a spectrum on the horizontal axis, data value on the vertical axis, and the two spatial co-ordinates for that spectrum. On exit the current database picture for the chosen device reverts to the input picture.

Related Applications :

KAPPA: DISPLAY, LINPLOT, MLINPLOT; FIGARO: SPECGRID; SPLAT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, WCS, and UNITS components of the input NDF.
- Processing of bad pixels and automatic quality masking are supported.

CMULT
Multiplies an NDF by a scalar

Description:

This application multiplies each pixel of an NDF by a scalar (constant) value to produce a new NDF.

Usage:

```
cmult in scalar out
```

Parameters:

IN = NDF (Read)
Input NDF structure whose pixels are to be multiplied by a scalar.

OUT = NDF (Write)
Output NDF structure.

SCALAR = _DOUBLE (Read)
The value by which the NDF's pixels are to be multiplied.

TITLE = LITERAL (Read)
A title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
cmult a 12.5 b
```

Multiplies all the pixels in the NDF called a by the constant value 12.5 to produce a new NDF called b.

```
cmult in=rawdata out=newdata scalar=-19
```

Multiplies all the pixels in the NDF called rawdata by -19 to give newdata.

Related Applications :

KAPPA: ADD, CADD, CDIV, CSUB, DIV, MATHS, MULT, SUB.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is carried out using the appropriate floating-point type, but the numeric type of the input pixels is preserved in the output NDF.
- Huge NDFs are supported.

COLCOMP
Produces a colour composite of up to three two-dimensional NDFs

Description:

This application combines up to three two-dimensional NDFs, using a different primary colour (red, green or blue) to represent each NDF. The resulting colour composite image is available in two forms; as an NDF with an associated colour table (see Parameters OUT and LUT), and as an ASCII PPM image file (see Parameter PPM). The full pixel resolution of the input NDFs is retained. Note, this application does not actually display the image, it just creates various output files which must be displayed using other tools (see below).

The data values in each of the input NDFs which are to be mapped on to zero intensity and full intensity can be given manually using Parameters RLOW, RHIGH, GLOW, GHIGH, BLOW and BHIGH, but by default they are evaluated automatically. This is done by finding specified percentile points within the data histograms of each of the input images (see Parameter PERCENTILES).

The NDF outputs are intended to be displayed with KAPPA application DISPLAY, using the command:

```
display <out> scale=no lut=<lut>
```

where "<out>" and "<lut>" are the names of the NDF image and colour table created by this application using Parameters OUT and LUT. The main advantage of this NDF form of output over the PPM form is that any WCS or AXIS information in the input NDFs is still available, and can be used to create axis annotations by the DISPLAY command. The graphics device which will be used to display the image must be specified when running this application (see Parameter DEVICE).

The PPM form of output can be displayed using tools such as **xv**, or converted into other forms (GIF or JPEG, for instance) using tools such as **ppmtogif** and **cjpeg** from the NetPbm or PbmPlus packages. These tools provide more sophisticated colour quantisation methods than are used by this application when creating the NDF outputs, and so may give better visual results.

Usage:

```
colcomp inr ing inb out lut [device]
```

Parameters:

BADCOL = LITERAL (Read)

The colour with which to mark any bad (*i.e.* missing) pixels in the display. There are a number of options described below.

- "MAX" — The maximum colour index used for the display of the image.
- "MIN" — The minimum colour index used for the display of the image.
- An integer — The actual colour index. It is constrained between 0 and the maximum colour index available on the device.

- 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 colour code such as `#ff002d`.

If the colour is to remain unaltered as the lookup table is manipulated choose an integer between 0 and 15, or a named colour. Note, if only the PPM output is to be created (see Parameter PPM), then a named colour must be given for BADCOL. [current value]

BHIGH = _DOUBLE (Read)
The data value corresponding to full blue intensity. If a null (`!`) value is supplied, the value actually used will be determined by forming a histogram of the data values in the NDF specified by Parameter INB, and finding the data value at the second histogram percentile specified by Parameter PERCENTILES. `[!]`

BLOW = _DOUBLE (Read)
The data value corresponding to zero blue intensity. If a null (`!`) value is supplied, the value actually used will be determined by forming a histogram of the data values in the NDF specified by Parameter INB, and finding the data value at the first histogram percentile specified by Parameter PERCENTILES. `[!]`

DEVICE = DEVICE (Read)
The name of the graphics device which will be used to display the NDF output (see Parameter OUT). This is needed only to determine the number of available colours. No graphics output is created by this application. This parameter is not accessed if a null (`!`) value is supplied for Parameter OUT. The device must have at least 24 colour indices or grey-scale intensities. [current graphics device]

GHIGH = _DOUBLE (Read)
The data value corresponding to full green intensity. If a null (`!`) value is supplied, the value actually used will be determined by forming a histogram of the data values in the NDF specified by Parameter ING, and finding the data value at the second histogram percentile specified by Parameter PERCENTILES. `[!]`

GLOW = _DOUBLE (Read)
The data value corresponding to zero green intensity. If a null (`!`) value is supplied, the value actually used will be determined by forming a histogram of the data values in the NDF specified by Parameter ING, and finding the data value at the first histogram percentile specified by Parameter PERCENTILES. `[!]`

INB = NDF (Read)
The input NDF containing the data to be displayed in blue. A null (`!`) value may be supplied in which case the blue intensity in the output will be zero at every pixel.

ING = NDF (Read)
The input NDF containing the data to be displayed in green. A null (`!`) value may be supplied in which case the green intensity in the output will be zero at every pixel.

INR = NDF (Read)
The input NDF containing the data to be displayed in red. A null (`!`) value may be supplied in which case the red intensity in the output will be zero at every pixel.

LUT = NDF (Write)
Name of the output NDF to contain the colour lookup table which should be used when displaying the NDF created using Parameter OUT. This colour table can be

loaded using LUTREAD, or specified when the image is displayed. This parameter is not accessed if a null (!) value is given for Parameter OUT.

RHIGH = _DOUBLE (Read)
The data value corresponding to full red intensity. If a null (!) value is supplied, the value actually used will be determined by forming a histogram of the data values in the NDF specified by Parameter INR, and finding the data value at the second histogram percentile specified by Parameter PERCENTILES. [!]

RLOW = _DOUBLE (Read)
The data value corresponding to zero red intensity. If a null (!) value is supplied, the value actually used will be determined by forming a histogram of the data values in the NDF specified by Parameter INR, and finding the data value at the first histogram percentile specified by Parameter PERCENTILES. [!]

OUT = NDF (Write)
The output colour composite image in NDF format. Values in this output image are integer colour indices into the colour table created using Parameter LUT. The values are shifted to account for the indices reserved for the palette (*i.e.* the first entry in the colour table is addressed as entry 16, not entry 1). The NDF is intended to be used as the input data in conjunction with `display scale=false`. If a null value (!) is supplied, no output NDF will be created.

PERCENTILES(2) = _REAL (Read)
The percentiles that define the default scaling limits. For example, [25,75] would scale between the quartile values. [5,95]

PPM = FILE (Write)
The name of the output text file to contain the PPM form of the colour composite image. The colours specified in this file represent the input data values directly. They are not quantised or dithered in any way. Also note that because this is a text file, containing formatted data values, it is portable, but can be very large, and slow to read and write. If a null (!) value is supplied, no PPM output is created. [!]

Examples:

```
colcomp m31_r m31_g m31_b m31_col m31_lut
```

Combines the 3 NDFs m31_r, m31_g, and m31_b to create a colour composite image stored in NDF m31_col. A colour look up table is also created and stored in NDF m31_lut. It is assumed that the output image will be displayed on the current graphics device. The created colour composite image should be displayed using the command:

```
display m31_col scale=no lut=m31_lut
```

```
colcomp m31_r m31_g m31_b out=!  ppm=m31.ppm
```

As above, but no NDF outputs are created. Instead, a file called `m31.ppm` is created which (for instance) can be displayed using the command:

```
xv m31.ppm
```

It can be converted to a GIF (for instance, for inclusion in WWW pages) using the command:

```
ppmquant 256 m31.ppm | ppmtogif > m31.gif
```

These commands assume you have **xv**, **ppmquant** and **ppmtogif** installed at your site. None of them are part of KAPPA.

Notes:

- The output image (PPM or NDF) covers the area of overlap between the input NDFs at full resolution. If the input NDFs are very large is is a good idea to compress them first (for instance, using COMPAVE) to reduce the size of the output images. Note, compressing the output NDF will normally produce spurious colours in the compressed image.
- The output image is based on the values in the DATA components of the input NDFs. Any VARIANCE and QUALITY arrays in the input NDFs are ignored.

Related Applications :

KAPPA: DISPLAY, LUTREAD; XV; PBMPLUS; NETPBM.

Implementation Status:

- The HISTORY, WCS, and AXIS components, together with any extensions are propagated to the output NDF, from the first supplied input NDF.
- Processing of bad pixels and automatic quality masking are supported.
- Only data of type _REAL can be processed directly. Data of other types will be converted to _REAL before being processed.

COLLAPSE
Reduces the number of axes in an *n*-dimensional NDF by compressing it along a nominated axis

Description:

This application collapses a nominated axis of an *n*-dimensional NDF , producing an output NDF with one fewer axes than the input NDF. A specified range of axis values can be used instead of the whole axis (see Parameters LOW and HIGH).

For each output pixel, all corresponding input pixel values between the specified bounds of the nominated axis to be collapsed are combined together using one of a selection of estimators, including a mean, mode, or median, to produce the output pixel value.

Possible uses include such things as collapsing a range of wavelength planes in a three-dimensional RA/DEC/Wavelength cube to produce a single two-dimensional RA/DEC image, or collapsing a range of slit positions in a two-dimensional slit position/wavelength image to produce a one-dimensional wavelength array.

Usage:

```
collapse in out axis [low] [high] [estimator] [wlim]
```

Parameters:

AXIS = LITERAL (Read)

The axis along which to collapse the NDF. This can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If the axes of the current Frame are not parallel to the NDF pixel axes, then the pixel axis which is most nearly parallel to the specified current Frame axis will be used.

CLIP = _REAL (Read)

The number of standard deviations about the mean at which to clip outliers for the `"Mode"`, `"Cmean"` and `"Csigma"` statistics (see Parameter ESTIMATOR). The application first computes statistics using all the available pixels. It then rejects all those pixels whose values lie beyond CLIP standard deviations from the mean and will then re-evaluate the statistics. For `"Cmean"` and `"Csigma"` there is currently only one iteration, but up to seven for `"Mode"`.

The value must be positive. `[3.0]`

COMP = LITERAL (Read)
The name of the NDF array component for which statistics are required: `"Data"`, `"Error"`, `"Quality"` or `"Variance"` (where `"Error"` is the alternative to `"Variance"` and causes the square root of the variance values to be taken before computing the statistics). If `"Quality"` is specified, then the quality values are treated as numerical values (in the range 0 to 255). `["Data"]`

ESTIMATOR = LITERAL (Read)
The method to use for estimating the output pixel values. It can be one of the following options. The first five are more for general collapsing, and the remainder are for cube analysis.

- `"Mean"` — Mean value
- `"WMean"` — Weighted mean in which each data value is weighted by the reciprocal of the associated variance (not available for COMP=`"Variance"` or `"Error"`).
- `"Mode"` — Modal value
- `"Median"` — Median value. Note that this is extremely memory and CPU intensive for large datasets; use with care! If strange things happen, use `"Mean"`.
- `"FastMed"` — Faster median using Wirth's algorithm for selecting the *k*th value, rather than a full sort. Weighting is not supported, thus this option is unavailable if both Parameter VARIANCE is `TRUE` and the input NDF contains a VARIANCE component.

- `"Absdev"` — Mean absolute deviation from the unweighted mean.
- `"Cmean"` — Sigma-clipped mean.
- `"Csigma"` — Sigma-clipped standard deviation.
- `"Comax"` — Co-ordinate of the maximum value.
- `"Comin"` — Co-ordinate of the minimum value.
- `"FBad"` — Fraction of bad pixel values.
- `"FGood"` — Fraction of good pixel values.
- `"Integ"` — Integrated value, being the sum of the products of the value and pixel width in world co-ordinates. Note that for sky co-ordinates the width is measured in radians. co-ordinates.
- `"Iwc"` — Intensity-weighted co-ordinate, being the sum of each value times its co-ordinate, all divided by the integrated value (see the `"Integ"` option).
- `"Iwd"` — Intensity-weighted dispersion of the co-ordinate, normalised like `"Iwc"` by the integrated value.
- `"Max"` — Maximum value.
- `"Min"` — Minimum value.
- `"NBad"` — Count of bad pixel values.
- `"NGood"` — Count of good pixel values.
- `"Rms"` — Root-mean-square value.
- `"Sigma"` — Standard deviation about the unweighted mean.
- `"Sum"` — The total value.

`["Mean"]`

HIGH = LITERAL (Read)
A formatted value for the axis specified by Parameter AXIS. For example, if AXIS is 3 and the current Frame of the input NDF has axes RA/DEC/Wavelength, then a wavelength value should be supplied. If, on the other hand, the current Frame in the NDF was the PIXEL Frame, then a pixel co-ordinate value would be required for the third axis (note, the pixel with index I covers a range of pixel co-ordinates from $(I-1)$ to I). Together with Parameter LOW, this parameter gives the range of axis values to be compressed. Note, HIGH and LOW should not be equal since. If a null value (`!`) is supplied for either HIGH or LOW, the entire range of the axis is collapsed. `[!]`

IN = NDF (Read)
The input NDF.

LOW = LITERAL (Read)
A formatted value for the axis specified by Parameter AXIS. For example, if AXIS is 3 and the current Frame of the input NDF has axes RA/DEC/Wavelength, then a wavelength value should be supplied. If, on the other hand, the current Frame in the NDF was the PIXEL Frame, then a pixel co-ordinate value would be required for the third axis (note, the pixel with index I covers a range of pixel co-ordinates from $(I-1)$ to I). Together with Parameter HIGH, this parameter gives the range of axis values to be compressed. Note, LOW and HIGH should not be equal since. If a null value (`!`) is supplied for either LOW or HIGH, the entire range of the axis is collapsed. `[!]`

OUT = NDF (Write)
The output NDF.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (`!`) propagates the title from the input NDF to the output NDF. `[!]`

TRIM = _LOGICAL (Read)
This parameter controls whether the collapsed axis should be removed from the co-ordinate systems describing the output NDF. If a `TRUE` value is supplied, the collapsed WCS axis will be removed from the WCS FrameSet of the output NDF, and the collapsed pixel axis will also be removed from the NDF, resulting in the output NDF having one fewer pixel axes than the input NDF. If a `FALSE` value is supplied, the collapsed WCS and pixel axes are retained in the output NDF, resulting in the input and output NDFs having the same number of pixel axes. In this case, the pixel-index bounds of the collapse axis will be set to (1:1) in the output NDF (that is, the output NDF will span only a single pixel on the collapse axis). Thus, setting TRIM to `FALSE` allows information to be retained about the range of values over which the collapse occurred. `[TRUE]`

VARIANCE = _LOGICAL (Read)
A flag indicating whether a variance array present in the NDF is used to weight data values while forming the estimator's statistic, and to derive output variance. If VARIANCE is `TRUE` and the NDF contains a variance array, this array will be used to define the weights, otherwise all the weights will be set equal. By definition this parameter is set to `FALSE` when COMP is `"Variance"` or `"Error"`.

The VARIANCE parameter is ignored and set to `FALSE` when there are more than 300 pixels along the collapse axis and ESTIMATOR is `"Median"`, `"Mode"`, `"Cmean"`,

or "Csigma". This prevents the covariance matrix from being huge. For "Median" estimates of variance come from mean variance instead. The other affected estimators switch to use equal weighting. [TRUE]

WCSATTS = GROUP (Read)

A group of attribute settings which will be used to make temporary changes to the properties of the current co-ordinate Frame in the WCS FrameSet before it is used. Supplying a list of attribute values for this parameter is equivalent to invoking WCSATTRIB on the input NDF prior to running this command, except that no permanent change is made to the NDF (however the changes are propagated through to the output NDF).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Any unspecified attributes will retain the value they have in the supplied NDF. No attribute values will be changed if a null value (!) is supplied. Any unrecognised attributes are ignored (no error is reported). [!]

WLIM = _REAL (Read)

If the input NDF contains bad pixels, then this parameter may be used to determine the number of good pixels which must be present within the range of collapsed input pixels before a valid output pixel is generated. It can be used, for example, to prevent output pixels from being generated in regions where there are relatively few good pixels to contribute to the collapsed result.

WLIM specifies the minimum fraction of good pixels which must be present in order to generate a good output pixel. If this specified minimum fraction of good input pixels is not present, then a bad output pixel will result, otherwise a good output value will be calculated. The value of this parameter should lie between 0.0 and 1.0 (the actual number used will be rounded up if necessary to correspond to at least one pixel). [0.3]

Examples:

```
collapse m31 profile axis=RA low="0:36:01" high="0:48:00"
```

Collapses the two-dimensional NDF called m31 along the right-ascension axis, from "0:36:01" to "0:48:00", and puts the result in an output NDF called profile.

```
collapse cube slab lambda 4500 4550
```

The current Frame in the input three-dimensional NDF called cube has axes with labels "RA", "DEC" and "Lambda", with the lambda axis being parallel to the third pixel axis. The above command extracts a slab of the input cube between wavelengths 4500 and 4550 Ångstroms, and collapses this slab into a single two-dimensional output NDF called slab

with RA and DEC axes. Each pixel in the output NDF is the mean of the corresponding input pixels with wavelengths between 4500 and 4550 Ångstroms.

```
collapse cube slab 3 4500 4550
```

The same as the previous example except the axis to collapse along is specified by index (3) rather than label (lambda).

```
collapse cube slab 3 101.0 134.0
```

This is the same as the second example, except that the current Frame in the input NDF has been set to the PIXEL Frame (using WCSFRAME), and so the high and low axis values are specified in pixel co-ordinates instead of Ångstroms. Note the difference between floating-point pixel co-ordinates, and integer pixel indices (for instance the pixel with index 10 extends from pixel co-ordinate 9.0 to pixel co-ordinate 10.0).

```
collapse cube slab 3 low=99.0 high=100.0
```

This is the same as the second example, except that a single pixel plane in the cube (pixel 100) is used to create the output NDF. Following the usual definition of pixel co-ordinates, pixel 100 extends from pixel co-ordinate 99.0 to pixel co-ordinate 100.0. So the given HIGH and LOW values encompass the single pixel plane at pixel 100.

Notes:

- The collapse is always performed along one of the pixel axes, even if the current Frame in the input NDF is not the PIXEL Frame. Special care should be taken if the current-Frame axes are not parallel to the pixel axes. The algorithm used to choose the pixel axis and the range of values to collapse along this pixel axis proceeds as follows.

  The current-Frame co-ordinates of the central pixel in the input NDF are determined (or some other point if the co-ordinates of the central pixel are undefined). Two current-Frame positions are then generated by substituting in turn into this central position each of the HIGH and LOW values for the current-Frame axis specified by Parameter AXIS. These two current-Frame positions are transformed into pixel co-ordinates, and the projections of the vector joining these two pixel positions on to the pixel axes are found. The pixel axis with the largest projection is selected as the collapse axis, and the two end points of the projection define the range of axis values to collapse.
- A warning is issued (at the normal reporting level) whenever any output values are set bad because there are too few contributing data values. This reports the fraction of flagged output data generated by the WLIM parameter's threshold.

  No warning is given when Parameter WLIM=0. Input data containing only bad values are not counted in the flagged fraction, since no potential good output value has been lost.

Related Applications :

KAPPA: WCSFRAME, COMPAVE, COMPICK, COMPADD, MANIC.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF; and propagates all extensions. QUALITY is not propagated.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.
- Huge NDFs are supported.

COMPADD
Reduces the size of an NDF by adding values in rectangular boxes

Description:

This application takes an NDF data structure and reduces it in size by integer factors along each dimension. The compression is achieved by adding the values of the input NDF within non-overlapping 'rectangular' boxes whose dimensions are the compression factors. The additions may be normalised to correct for any bad values present in the input NDF. The exact placement of the boxes can be controlled using Parameter ALIGN.

Usage:

```
compadd in out compress [wlim]
```

Parameters:

ALIGN = LITERAL (Read)

This parameter controls the placement of the compression boxes within the input NDF (also see Parameter TRIM). It can take any of the following values:

- "ORIGIN" — The compression boxes are placed so that the origin of the pixel co-ordinate Frame (*i.e.* pixel co-ordinates (0,0)) in the input NDF corresponds to a corner of a compression box. This results in the pixel origin being retain in the output NDF. For instance, if a pair of two-dimensional images which have previously been aligned in pixel co-ordinates are compressed, then using this option ensures that the compressed images will also be aligned in pixel co-ordinates.
- "FIRST" — The compression boxes are placed so that the first pixel in the input NDF (for instance, the bottom-left pixel in a two-dimensional image) corresponds to the first pixel in a compression box. This can result in the pixel origin being shifted by up to one compression box in the output image. Thus, images which were previously aligned in pixel co-ordinates may not be aligned after compression. You may want to use this option if you are using a very large box to reduce the number of dimensions in the data (for instance summing across the entire width of an image to produce a one-dimensional array).
- "LAST" — The compression boxes are placed so that the last pixel in the input NDF (for instance, the top-right pixel in a two-dimensional image) corresponds to the last pixel in a compression box. See the "FIRST" option above for further comments.

["ORIGIN"]

AXWEIGHT = _LOGICAL (Read)

When there is an AXIS variance array present in the NDF and AXWEIGHT=TRUE the application forms weighted averages of the axis centres using the variance. For all other conditions the non-bad axis centres are given equal weight during the averaging to form the output axis centres. [FALSE]

COMPRESS( ) = _INTEGER (Read)
Linear compression factors to be used to create the output NDF. There should be one for each dimension of the NDF. If fewer are supplied the last value in the list of compression factors is given to the remaining dimensions. Thus if a uniform compression is required in all dimensions, just one value need be entered. All values are constrained to be in the range one to the size of its corresponding dimension. The suggested default is the current value.

IN = NDF (Read)
The NDF structure to be reduced in size.

NORMAL = _LOGICAL (Read)
When there are bad pixels present in the summation box these are ignored. Therefore a simple addition of the input-array component's values will yield a result discordant with neighbouring output pixels that were formed from summation of all the pixels in the box. When NORMAL=TRUE the output values are normalised: the addition is multiplied by the ratio of the number of pixels in the box to the number of good pixels therein to arrive at the output value. When NORMAL=FALSE the output values are always just the sum of the good pixels. [TRUE]

OUT = NDF (Write)
NDF structure to contain compressed version of the input NDF.

PRESERVE = _LOGICAL (Read)
If the input data type is to be preserved on output then this parameter should be set TRUE. However, this may result in overflows for integer types and hence additional bad values written to the output NDF. If this parameter is set FALSE then the output data type will be one of _REAL or _DOUBLE, depending on the input type. [FALSE]

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (!) propagates the title from the input NDF to the output NDF. [!]

TRIM = _LOGICAL (Read)
If Parameter TRIM is set TRUE, the output NDF only contains data for compression boxes which are entirely contained within the input NDF. Any pixels around the edge of the input NDF which are not contained within a compression box are ignored. If TRIM is set FALSE, the output NDF contains data for all compression boxes which have any overlap with the input NDF. All pixels outside the bounds of the NDF are assumed to be bad. That is, any boxes which extend beyond the bounds of the input NDF are padded with bad pixels. See also Parameter ALIGN. [current value]

WLIM = _REAL (Read)
If the input NDF contains bad pixels, then this parameter may be used to determine the number of good pixels which must be present within the addition box before a valid output pixel is generated. It can be used, for example, to prevent output pixels from being generated in regions where there are relatively few good pixels to contribute to the smoothed result.

WLIM specifies the minimum fraction of good pixels which must be present in the summation box in order to generate a good output pixel. If this specified minimum fraction of good input pixels is not present, then a bad output pixel will result, otherwise the output value will be the sum of the good values. The value of this parameter should lie between 0.0 and 1.0 (the actual number used will be rounded up if necessary to correspond to at least one pixel). [0.3]

Examples:

```
compadd cosmos galaxy 4
```

This compresses the NDF called cosmos summing four times in each dimension, and stores the reduced data in the NDF called galaxy. Thus if cosmos is two-dimensional, this command would result in a sixteen-fold reduction in the array components.

```
compadd cosmos profile [10000,1] wlim=0 align=first trim=no
```

This compresses the two-dimensional NDF called cosmos to produce a one-dimensional NDF called profile. This is done using a compression box which is 1 pixel high, but which is wider than the whole input image. Each pixel in the output NDF thus corresponds to the sum of the corresponding row in the input image. WLIM is set to zero to ensure that bad pixels are ignored. ALIGN is set to `"FIRST"` so that each compression box is flush with the left edge of the input image. TRIM is set to `NO` so that compression boxes which extend outside the bounds of the input image (which will be all of them if the input image is narrower than 10000 pixels) are retained in the output NDF.

```
compadd cosmos galaxy 4 wlim=1.0
```

This compresses the NDF called cosmos adding four times in each dimension, and stores the reduced data in the NDF called galaxy. Thus if cosmos is two-dimensional, this command would result in a sixteen-fold reduction in the array components. If a summation box contains any bad pixels, the output pixel is set to bad.

```
compadd cosmos galaxy 4 0.0 preserve
```

As above except that a summation box need only contains a single non-bad pixels for the output pixel to be good, and galaxy's array components will have the same as those in cosmos.

```
compadd cosmos galaxy [4,3] nonormal title="COSMOS compressed"
```

This compresses the NDF called cosmos adding four times in the first dimension and three times in higher dimensions, and stores the reduced data in the NDF called galaxy. Thus if cosmos is two-dimensional, this command would result in a twelve-fold reduction in the array components. Also, if there are bad pixels there will be no normalisation correction for the missing values. The title of the output NDF is `"COSMOS compressed"`.

```
compadd in=arp244 compress=[1,1,3] out=arp244cs
```

Suppose arp244 is a huge NDF storing a spectral-line data cube, with the third dimension being the spectral axis. This command compresses arp244 in the spectral dimension, adding every three pixels to form the NDF called arp244cs.

Notes:

- The axis centres and variances are averaged, whilst the widths are summed and always normalised for bad values.

Related Applications :

KAPPA: BLOCK, COMPAVE, COMPICK, PIXDUPE, SQORST, REGRID; FIGARO: ISTRETCH, YSTRACT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions. QUALITY is not processed since it is a series of flags, not numerical values.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

COMPAVE
Reduces the size of an NDF by averaging values in rectangular boxes

Description:

This application takes an NDF data structure and reduces it in size by integer factors along each dimension. The compression is achieved by averaging the input NDF within non-overlapping 'rectangular' boxes whose dimensions are the compression factors. The averages may be weighted when there is a variance array present. The exact placement of the boxes can be controlled using Parameter ALIGN.

Usage:

```
compave in out compress [wlim]
```

Parameters:

ALIGN = LITERAL (Read)

This parameter controls the placement of the compression boxes within the input NDF (also see Parameter TRIM). It can take any of the following values:

- `"ORIGIN"` — The compression boxes are placed so that the origin of the pixel co-ordinate Frame (*i.e.* pixel co-ordinates (0,0)) in the input NDF corresponds to a corner of a compression box. This results in the pixel origin being retain in the output NDF. For instance, if a pair of two-dimensional images which have previously been aligned in pixel co-ordinates are compressed, then using this option ensures that the compressed images will also be aligned in pixel co-ordinates.
- `"FIRST"` — The compression boxes are placed so that the first pixel in the input NDF (for instance, the bottom-left pixel in a two-dimensional image) corresponds to the first pixel in a compression box. This can result in the pixel origin being shifted by up to one compression box in the output image. Thus, images which were previously aligned in pixel co-ordinates may not be aligned after compression. You may want to use this option if you are using a very large box to reduce the number of dimensions in the data (for instance averaging across the entire width of an image to produce a one-dimensional array).
- `"LAST"` — The compression boxes are placed so that the last pixel in the input NDF (for instance, the top-right pixel in a two-dimensional image) corresponds to the last pixel in a compression box. See the `"FIRST"` option above for further comments. `["ORIGIN"]`

AXWEIGHT = _LOGICAL (Read)

When there is an AXIS variance array present in the NDF and AXWEIGHT=`TRUE` the application forms weighted averages of the axis centres using the variance. For all other conditions the non-bad axis centres are given equal weight during the averaging to form the output axis centres. `[FALSE]`

COMPRESS( ) = _INTEGER (Read)

Linear compression factors to be used to create the output NDF. There should be

one for each dimension of the NDF. If fewer are supplied the last value in the list of compression factors is given to the remaining dimensions. Thus if a uniform compression is required in all dimensions, just one value need be entered. The suggested default is the current value.

IN = NDF (Read)
The NDF structure to be reduced in size.

OUT = NDF (Write)
NDF structure to contain compressed version of the input NDF.

PRESERVE = _LOGICAL (Read)
If the input data type is to be preserved on output then this parameter should be set TRUE. However, this will probably result in a loss of precision. If this parameter is set FALSE then the output data type will be one of _REAL or _DOUBLE, depending on the input type. [FALSE]

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (!) propagates the title from the input NDF to the output NDF. [!]

TRIM = _LOGICAL (Read)
If Parameter TRIM is set TRUE, the output NDF only contains data for compression boxes which are entirely contained within the input NDF. Any pixels around the edge of the input NDF which are not contained within a compression box are ignored. If TRIM is set FALSE, the output NDF contains data for all compression boxes which have any overlap with the input NDF. All pixels outside the bounds of the NDF are assumed to be bad. That is, any boxes which extend beyond the bounds of the input NDF are padded with bad pixels. See also Parameter ALIGN. [current value]

WEIGHT = _LOGICAL (Read)
When there is a variance array present in the NDF and WEIGHT=TRUE the application forms weighted averages of the data array using the variance. For all other conditions the non-bad pixels are given equal weight during averaging. [FALSE]

WLIM = _REAL (Read)
If the input NDF contains bad pixels, then this parameter may be used to determine the number of good pixels which must be present within the averaging box before a valid output pixel is generated. It can be used, for example, to prevent output pixels from being generated in regions where there are relatively few good pixels to contribute to the smoothed result.

WLIM specifies the minimum fraction of good pixels which must be present in the averaging box in order to generate a good output pixel. If this specified minimum fraction of good input pixels is not present, then a bad output pixel will result, otherwise an averaged output value will be calculated. The value of this parameter should lie between 0.0 and 1.0 (the actual number used will be rounded up if necessary to correspond to at least one pixel). [0.3]

Examples:

```
compave cosmos galaxy 4
```

This compresses the NDF called cosmos averaging four times in each dimension, and stores the reduced data in the NDF called galaxy. Thus if cosmos is two-dimensional, this command would result in a sixteen-fold reduction in the array components.

```
compave cosmos profile [10000,1] wlim=0 align=first trim=no
```

This compresses the two-dimensional NDF called cosmos to produce a one-dimensional NDF called profile. This is done using a compression box which is 1 pixel high, but which is wider than the whole input image. Each pixel in the output NDF thus corresponds to the average of the corresponding row in the input image. WLIM is set to zero to ensure that bad pixels are ignored. ALIGN is set to "FIRST" so that each compression box is flush with the left edge of the input image. TRIM is set to NO so that compression boxes which extend outside the bounds of the input image (which will be all of them if the input image is narrower than 10000 pixels) are retained in the output NDF.

```
compave cosmos galaxy 4 wlim=1.0
```

This compresses the NDF called cosmos averaging four times in each dimension, and stores the reduced data in the NDF called galaxy. Thus if cosmos is two-dimensional, this command would result in a sixteen-fold reduction in the array components. If an averaging box contains any bad pixels, the output pixel is set to bad.

```
compave cosmos galaxy 4 0.0 preserve
```

As above except that an averaging box need only contains a single non-bad pixels for the output pixel to be good, and galaxy's array components will have the same as those in cosmos.

```
compave cosmos galaxy [4,3] weight title="COSMOS compressed"
```

This compresses the NDF called cosmos averaging four times in the first dimension and three times in higher dimensions, and stores the reduced data in the NDF called galaxy. Thus if cosmos is two-dimensional, this command would result in a twelve-fold reduction in the array components. Also, if there is a variance array present it is used to form weighted means of the data array. The title of the output NDF is "COSMOS compressed".

```
compave in=arp244 compress=[1,1,3] out=arp244cs
```

Suppose arp244 is a huge NDF storing a spectral-line data cube, with the third dimension being the spectral axis. This command compresses arp244 in the spectral dimension, averaging every three pixels to form the NDF called arp244cs.

Notes:

- The axis centres and variances are averaged, whilst the widths are summed and always normalised for bad values.

Related Applications :

KAPPA: BLOCK, COMPADD, COMPICK, PIXDUPE, SQORST, REGRID; FIGARO: ISTRETCH.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions. QUALITY is not processed since it is a series of flags, not numerical values.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

COMPICK
Reduces the size of an NDF by picking equally spaced pixels

Description:

This application takes an NDF data structure and reduces it in size by integer factors along each dimension. The input NDF is sampled at these constant compression factors or intervals along each dimension, starting from a defined origin, to form an output NDF structure. The compression factors may be different in each dimension.

Usage:

```
compick in out compress [origin]
```

Parameters:

COMPRESS( ) = _INTEGER (Read)

Linear compression factors to be used to create the output NDF. There should be one for each dimension of the NDF. If fewer are supplied the last value in the list of compression factors is given to the remaining dimensions. Thus if a uniform compression is required in all dimensions, just one value need be entered. All values are constrained to be in the range one to the size of its corresponding dimension. The suggested default is the current value.

IN = NDF (Read)

The NDF structure to be reduced in size.

ORIGIN( ) = _INTEGER (Read)

The pixel indices of the first pixel to be selected. Thereafter the selected pixels will be spaced equally by COMPRESS() pixels. The origin must lie within the first selection intervals, therefore the ith origin must be in the range LBND(i) to LBND(i)+COMPRESS(i)-1, where LBND(i) is the lower bound of the ith dimension. If a null (`!`) value is supplied, the first array element is used. `[!]`

OUT = NDF (Write)

NDF structure to contain compressed version of the input NDF.

TITLE = LITERAL (Read)

Title for the output NDF structure. A null value (`!`) propagates the title from the input NDF to the output NDF. `[!]`

Examples:

```
compick cosmos galaxy 4
```

This compresses the NDF called cosmos selecting every fourth array element along each dimension, starting from the first element in the NDF, and stores the reduced data in the NDF called galaxy.

```
compick cosmos galaxy 4 [3,2]
```

This compresses the two-dimensional NDF called cosmos selecting every fourth

array element along each dimension, starting from the pixel index (3,2), and stores the reduced data in the NDF called galaxy.

```
compick in=arp244 compress=[1,1,3] out=arp244cs
```

Suppose arp244 is a huge NDF storing a spectral-line data cube, with the third dimension being the spectral axis. This command compresses arp244 in the spectral dimension, sampling every third pixel, starting from the first wavelength at each image position, to form the NDF called arp244cs.

Notes:

- The compression is centred on the origin of the pixel co-ordinate Frame. That is, if a position has a value p(*i*) on the *i*'th pixel co-ordinate axis of the input NDF, then it will have position p(*i*)/COMPRESS(*i*) on the corresponding axis of the output NDF. The pixel index bounds of the output NDF are chosen accordingly.

Related Applications :

KAPPA: BLOCK, COMPADD, COMPAVE, PIXDUPE, SQORST, REGRID; FIGARO: ISTRETCH.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

COMPLEX
Converts between representations of complex data

Description:

This application converts between various representations of complex data, including complex NDFs . The conversion may simply unpack or pack real and imaginary parts of a complex NDF, or it may convert between polar and Cartesian representation.

Usage:

```
complex in1 in2 out1 out2 [intype] [outtype]
```

Parameters:

IN1 = NDF (Read)
The first input NDF. See Parameter INTYPE for its description.

IN2 = NDF (Read)
The second input NDF. See Parameter INTYPE for its description. IN2 will not be accessed when Parameter INTYPE is set to "COMPLEX". When Parameter INTYPE is set to "MOD_ARG", supply a null (!) value.

INTYPE = LITERAL (Read)
The nature of the input NDF(s). The allowed options are listed below.

- "COMPLEX" – IN1 is a complex NDF containing real and imaginary parts. (IN2 will not be accessed.)
- "REAL_IMAG" – IN1 contains the real part and IN2 contains the imaginary parts.
- "MOD_ARG" – IN1 contains the modulus and IN2 the argument in radians.

The default is"COMPLEX" if IN1 is a complex NDF, otherwise it is "REAL_IMAG". []

OUT1 = NDF (Write)
The first output NDF. Its contents are governed by Parameter OUTTYPE.

OUT2 = NDF (Write)
The second output NDF. Its contents are governed by Parameter OUTTYPE. OUT2 will not be accessed when Parameter OUTTYPE is set to "COMPLEX". When Parameter OUTTYPE is set to "MOD_ARG", supply a null (!) value.

OUTTYPE = LITERAL (Read)
The nature of the output NDF(s). The same options are available as for INTYPE, but relate to NDFs OUT1 and OUT instead of IN1 and IN2.

The default is "REAL_IMAG" if IN1 is a complex NDF, otherwise it is "COMPLEX". []

TITLE1 = LITERAL (Read)
The title for the first output NDF.

TITLE2 = LITERAL (Read)
The title for the second output NDF.

Results Parameters:

CALCMODE = LITERAL (Write)
Set to indicate the type of calculation that was performed: `"To polar"`, `"From polar"` or `"None"`.

Examples:

```
complex realmap imagmap cmplxmap
```

This example combines real and imaginary parts into a complex NDF.

```
complex cplxmap !  modulusmap !  OUTTYPE=MOD_ARG
```

This example would compute the modulus from a complex NDF.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, WCS, and HISTORY components of the first input NDF and propagates all of that NDF's extensions. UNITS is set to `"radians"` for OUTTYPE = `"MOD_ARG"`.
- The DATA component is processed in double precision. The output NDFs have type _DOUBLE, except when OUTTYPE ="COMPLEX" where it is COMPLEX_DOUBLE.

CONFIGECHO
Displays one or more configuration parameters

Description:

This application displays the name and value of one or all configuration parameters, specified using Parameters CONFIG or NDF. If a single parameter is displayed, its value is also written to an output parameter. If the parameter value is not specified by the CONFIG, NDF or DEFAULTS parameter, then the value supplied for DEFVAL is displayed.

If an input NDF is supplied then configuration parameters are read from its history (see Parameters NDF and APPLICATION).

If values are supplied for both CONFIG and NDF, then the differences between the two sets of configuration parameters are displayed (see Parameter NDF).

Usage:

```
configecho name config [defaults] [select] [defval]
```

Parameters:

APPLICATION = LITERAL (Read)

When reading configuration parameters from the history of an NDF, this parameter specifies the name of the application to find in the history. There must be a history component corresponding to the value of this parameter, and it must include a CONFIG group. [!]

CONFIG = GROUP (Read)

Specifies values for the configuration parameters. If the string "def" (case-insensit
or a null (!) value is supplied, the configuration parameters are obtained using Parameter NDF. If a null value is also supplied for NDF, a set of default configuration parameter values will be used, as specified by Parameter DEFAULTS.

The supplied value should be either a comma-separated list of strings or the name of a text file preceded by an up-arrow character "^", containing one or more comma-separated lists of strings. Each string is either a "keyword=value' setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner (any blank lines or lines beginning with "#" are ignored). Within a text file, newlines can be used as delimiters, as well as commas. Settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same keyword.

Each individual setting should be of the form "<keyword>=<value>". If a non-null value is supplied for Parameter DEFAULTS, an error will be reported if CONFIG includes values for any parameters that are not included in DEFAULTS.

DEFAULTS = LITERAL (Read)

The path to a file containing the default value for every allowed configuration parameter. If null (!) is supplied, no defaults will be supplied for parameters

that are not specified by CONFIG, and no tests will be performed on the validity of paramter names supplied by CONFIG. [!]

DEFVAL = LITERAL (Read)
The value to return if no value can be obtained for the named parameter, or if the value is "<undef>". [<***>]

LOGFILE = LITERAL (Read)
The name of a text file in which to store the displayed configuration parameters. [!]

NAME = LITERAL (Read)
The name of the configuration parameter to display. If it is set to null (!), then all parameters defined in the configuration are displayed.

NDF = NDF (Read)
An NDF file containing history entries which include configuration parameters.

SELECT = GROUP (Read)
A group that specifies any alternative prefixes that can be included at the start of any parameter name. For instance, if this group contains the two entries "450=1" and "850=0", then either CONFIG or DEFAULTS can specify two values for any single parameter--one for the parameter prefixed by "450." and another for the parameter prefixed by "850.". Thus, for instance, if DEFAULTS defines a parameter called "filter", it could include "450.filter=300" and "850.filter=600". The CONFIG parameter could then either set the filter parameter for a specific prefix (as in "450.filter=234"); or it could leave the prefix unspecified, in which case the prefix used is the first one with a non-zero value in SELECT (450 in the case of this example--850 has a value zero in SELECT). Thus the names of the items in SELECT define the set of allowed alternative prefixes, and the values indicate which one of these alternatives is to be used (the first one with non-zero value). [!]

SORT = _LOGICAL (Read)
If TRUE then sort the listed parameters in to alphabetical order. Otherwise, retain the order they have in the supplied configuration. Only used if a null (!) value is supplied for Parameter NAME. [FALSE]

Results Parameters:

VALUE = LITERAL (Write)
The value of the configuration parameter, or "<***>" if the parameter has no value in CONFIG and DEFAULTS.

Examples:

```
configecho m81 ^myconf
```

Report the value of configuration parameter "m81" defined within the file myconf. If the file does not contain a value for "m81", then <***> is displayed.

```
configecho type ^myconf select="m57=0,m31=1,m103=0"
```

Report the value of configuration parameter "type" defined within the file myconf. If the

file does not contain a value for "type", then the value of "m31.type" will be reported instead. If neither is present, then <***> is displayed.

```
configecho flt.filt_edge_largescale \
config=^/star/share/smurf/dimmconfig.lis \
defaults=/star/bin/smurf/smurf_makemap.def select="450=1,850=0"
```

Report the value of configuration parameter flt.filt_edge_largescale defined within the file /star/share/smurf/dimmconfig.lis, using defaults from the file /star/bin/smurf/smurf_makemap.def. If dimmconfig.lis does not contain a value for flt.filt_edge_largescale then it is searched for 450.flt.filt_edge_largescale instead. An error is reported if dimmconfig.lis contains values for any items that are not defined in smurf_makemap.def.

```
configecho ndf=omc1 config=^/star/share/smurf/dimmconfig.lis \
defaults=/star/bin/smurf/smurf_makemap.def \
application=makemap name=!  sort select="450=0,850=1"
```

Show how the configuration used to generate the 850μm map of OMC1 differs from the basic dimmconfig.lis file.

CONTOUR
Contours a two-dimensional NDF

Description:

This application produces a contour map of a two-dimensional NDF on the current graphics device, with single-pixel resolution. Contour levels can be chosen automatically in various ways, or specified explicitly (see Parameter MODE). In addition, this application can also draw an outline around either the whole data array, or around the good pixels in the data array (set MODE to `"Bounds"` or `"Good"`).

The plot is produced within the current graphics database picture, and may be aligned with an existing DATA picture if the existing picture contains suitable co-ordinate Frame information (see Parameter CLEAR).

The appearance of each contour can be controlled in several ways. The pens used can be rotated automatically (see Parameter PENROT). Contours below a given threshold value can be drawn dashed (see Parameter DASHED). Alternatively, the appearance of each contour can be set explicitly (see Parameter PENS).

Annotated axes can be produced (see Parameter AXES), and the appearance of the axes can be controlled in detail (see Parameter STYLE). The axes show co-ordinates in the current co-ordinate Frame of the supplied NDF.

A list of the contour levels can be displayed to the right of the contour map (see Parameter KEY). The appearance and position of this key may be controlled using Parameters KEYSTYLE and KEYPOS.

Usage:

```
contour ndf [comp] mode ncont [key] [device]   { low =? high =?
                                               { percentiles =?
                                               { sigmas =?
                                               mode
```

Parameters:

AXES = _LOGICAL (Read)

`TRUE` if labelled and annotated axes are to be drawn around the contour map, showing the current co-ordinate Frame of the supplied NDF. The appearance of the axes can be controlled using the STYLE parameter. If a null (`!`) value is supplied, then axes will be drawn unless the CLEAR parameter indicates that the graphics device is not being cleared. `[!]`

CLEAR = _LOGICAL (Read)

`TRUE` if the graphics device is to be cleared before displaying the contour map. If you want the contour map to be drawn over the top of an existing DATA picture, then set CLEAR to `FALSE`. The contour map will then be drawn in alignment with the displayed data. If possible, alignment occurs within the current co-ordinate Frame of the NDF. If this is not possible, (for instance if suitable WCS information was not stored with the existing DATA picture), then alignment is attempted in PIXEL

co-ordinates. If this is not possible, then alignment is attempted in GRID co-ordinates. If this is not possible, then alignment is attempted in the first suitable Frame found in the NDF irrespective of its domain. A message is displayed indicating the domain in which alignment occurred. If there are no suitable Frames in the NDF then an error is reported. [TRUE]

COMP = LITERAL (Read)
The NDF component to be contoured. It may be "Data", "Quality", "Variance", or "Error" (where "Error" is an alternative to "Variance" and causes the square root of the variance values to be displayed). If "Quality" is specified, then the quality values are treated as numerical values (in the range 0 to 255). ["Data"]

DASHED = _REAL (Read)
The height below which the contours will be drawn with dashed lines (if possible). A null value (!) results in contours being drawn with the styles specified by Parameters PENS, PENROT, and STYLE. [!]

DEVICE = DEVICE (Read)
The plotting device. [current graphics device]

FAST = _LOGICAL (Read)
If TRUE, then a faster, but in certain cases less-accurate, method is used to draw the contours. In fast mode, contours may be incorrectly placed on the display if the mapping between graphics co-ordinates and the current co-ordinate Frame of the supplied NDF has any discontinuities, or is strongly non-linear. This may be the case, for instance, when displaying all-sky maps on top of each other. [TRUE]

FILL = _LOGICAL (Read)
The contour plot normally has square pixels, in other words a specified length along each axis corresponds to the same number of pixels. However, for images with markedly different dimensions this default behaviour may not be suitable or give the clearest plot. When FILL is TRUE, the square-pixel constraint is relaxed and the contour plot is the largest possible within the current picture. When FILL is FALSE, the pixels are square. [FALSE]

FIRSTCNT = _REAL (Read)
Height of the first contour (Linear and Magnitude modes).

HEIGHTS() = _REAL (Read)
The required contour levels (Free mode).

KEY = _LOGICAL (Read)
TRUE if a key of the contour level versus pixel value is to be produced. The appearance of this key can be controlled using Parameter KEYSTYLE, and its position can be controlled using Parameter KEYPOS. [TRUE]

KEYPOS() = _REAL (Read)
Two values giving the position of the key. The first value gives the gap between the right-hand edge of the contour map and the left-hand edge of the key (0.0 for no gap, 1.0 for the largest gap). The second value gives the vertical position of the top of the key (1.0 for the highest position, 0.0 for the lowest). If the second value is not given, the top of the key is placed level with the top of the contour map. Both values should be in the range 0.0 to 1.0. If a key is produced, then the right-hand margin specified by Parameter MARGIN is ignored. [current value]

KEYSTYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the key (see Parameter KEY).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The heading in the key can be changed by setting a value for the Title attribute (the supplied heading is split into lines of no more than 17 characters). The appearance of the heading is controlled by attributes Colour(Title), Font(Title), *etc*. The appearance of the contour indices is controlled by attributes Colour(TextLab), Font(TextLab), *etc.* (the synonym Index can be used in place of TextLab). The appearance of the contour values is controlled by attributes Colour(NumLab), Font(NumLab), *etc.* (the synonym Value can be used in place of NumLab). Contour indices are formatted using attributes Format(1), Digits(1), *etc.* (the synonym Index can be used in place of value 1). Contour values are formatted using attributes Format(2), *etc.* (the synonym Value can be used in place of the value 2). [current value]

LABPOS = _REAL() (Read)

Only used if Parameter MODE is set to `"Good"` or `"Bounds"`. It specifies the position at which to place a label identifying the input NDF within the plot. The label is drawn parallel to the first pixel axis. Two values should be supplied for LABPOS. The first value specifies the distance in millimetres along the first pixel axis from the centre of the bottom-left pixel to the left edge of the label. The second value specifies the distance in millimetres along the second pixel axis from the centre of the bottom-left pixel to the baseline of the label. If a null (`!`) value is given, no label is produced. The appearance of the label can be set by using the STYLE parameter (for instance `"Size(strings)=2"`). [current value]

LASTCNT = _REAL (Read)

Height of the last contour (Linear and Magnitude modes).

MARGIN(4) = _REAL (Read)

The widths of the margins to leave around the contour map for axis annotation. The widths should be given as fractions of the corresponding dimension of the current picture. The actual margins used may be increased to preserve the aspect ratio of the DATA 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. If a null (!) value is supplied, the value used is 0.15 (for all edges) if annotated axes are being produced, and zero otherwise. See also Parameter KEYPOS. [current value]

MODE = LITERAL (Read)

The method used to select the contour levels. The options are:

- "Area" — The contours enclose areas of the array for which the equivalent radius increases by equal increments. You specify the number of levels.
- "Automatic" — The contour levels are equally spaced between the maximum and minimum pixel values in the array. You supply the number of contour levels.
- "Bounds" — A single 'contour' is drawn representing the bounds of the input array. A label may also be added (see Parameter LABPOS).
- "Equalised" — You define the number of equally spaced percentiles.
- "Free" — You specify a series of contour values explicitly.
- "Good" — A single 'contour' is drawn outlining the good pixel values. A label may also be added (see Parameter LABPOS).
- "Linear" — You define the number of contours, the start contour level and linear step between contours.
- "Magnitude" — You define the number of contours, the start contour level and step between contours. The step size is in magnitudes so the *n*th contour is dex(-0.4∗(n-1)∗step) times the start contour level.
- "Percentiles" — You specify a series of percentiles.
- "Scale" — The contour levels are equally spaced between two pixel values that you specify. You also supply the number of contour levels, which must be at least two.

If the contour map is aligned with an existing DATA picture (see Parameter CLEAR), then only part of the supplied NDF may be displayed. In this case, the choice of contour levels is based on the data within a rectangular section of the input NDF enclosing the existing DATA picture. Data values outside this section are ignored.

NCONT = _INTEGER (Read)

The number of contours to draw (only required in certain modes). It must be between 1 and 50. If the number is large, the plot may be cluttered and take longer to produce. The initial suggested default of 6 gives reasonable results.

NDF = NDF (Read)

NDF structure containing the two-dimensional image to be contoured.

PENROT = _LOGICAL (Read)

If TRUE, the plotting pens are cycled through the contours to aid identification of the contour heights. Only accessed if pen definitions are not supplied using Parameter PENS. [FALSE]

PENS = GROUP (Read)

A group of strings, separated by semicolons, each of which specifies the appearance of a pen to be used to draw a contour. The first string in the group describes the pen to use for the first contour, the second string describes the pen for the second contour, *etc*. If there are fewer strings than contours, then the supplied pens are cycled

through again, starting at the beginning. Each string should be a comma-separated list of plotting attributes to be used when drawing the contour. For instance, the string `"width=10.0,colour=red,style=2"` produces a thick, red, dashed contour. Attributes that are unspecified in a string default to the values implied by Parameter STYLE. If a null value (`!`) is given for PENS, then the pens implied by Parameters PENROT, DASHED and STYLE are used. `[!]`

PERCENTILES() = _REAL (Read)
Contour levels given as percentiles. The values must lie between 0.0 and 100.0. (Percentiles mode).

STATS = _LOGICAL (Read)
If `TRUE`, the LENGTH and NUMBER statistics are computed. `[FALSE]`.

STEPCNT = _REAL (Read)
Separation between contour levels, linear for Linear mode and in magnitudes for Magnitude mode.

STYLE = GROUP (Read)
A group of attribute settings describing the plotting style to use for the contours and annotated axes.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the contours is controlled by the attributes Colour(Curves), Width(Curves), *etc.* (the synonym Contours may be used in place of Curves). The contour appearance established in this way may be modified using Parameters PENS, PENROT and DASHED. [current value]

USEAXIS = GROUP (Read)
USEAXIS is only accessed if the current co-ordinate Frame of the NDF has more than two axes. A group of two strings should be supplied specifying the two axes which are to be used when annotating and aligning the contour map. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).

- Its Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (!) value is supplied, the axes with the same indices as the two significant NDF pixel axes are used. [!]

Results Parameters:

LENGTH() = _REAL (Write)
On exit this holds the total length in pixels of the contours at each selected height. These values are only computed when Parameter STATS is TRUE.

NUMBER() = _INTEGER (Write)
On exit this holds the number of closed contours at each selected height. Contours are not closed if they intersect a bad pixel or the edge of the image. These values are only computed when Parameter STATS is TRUE.

Examples:

```
contour myfile
```

Contours the data array in the NDF called myfile on the current graphics device. All other settings are defaulted, so for example the current mode for determining heights is used, and a key is plotted.

```
contour taurus1(100:199,150:269,4)
```

Contours a two-dimensional section of the three-dimensional NDF called taurus1 on the current graphics device. The section extends from pixel (100,150,4) to pixel (199,269,4).

```
contour ngc6872 mode=au ncont=5 device=ps_l pens="style=1;style=2"
```

Contours the data array in the NDF called ngc6872 on the ps_l graphics device. Five equally spaced contours between the maximum and minimum data values are drawn, alternating between line styles 1 and 2 (solid and dashed).

```
contour ndf=ngc6872 mode=au ncont=5 penrot style="^mysty,grid=1"
```

As above except that the current graphics device is used, pens are cycled automatically, and the appearance of the axes is read from text file mysty. The plotting attribute Grid is set explicitly to 1 to ensure that a co-ordinate grid is drawn over the plot. The text file mysty could, for instance, contain the two lines "Title=NGC6872 at 25 microns" and "grid=0". The Title setting gives the title to display at the top of the axes. The Grid setting would normally prevent a co-ordinate grid being drawn, but is overridden in this example by the explicit setting for Grid which follows the file name.

```
contour m51 mode=li firstcnt=10 stepcnt=2 ncont=4 keystyle=^keysty
```

Contours the data array in the NDF called m51 on the current graphics device. Four contours at heights 10, 12, 14, and 16 are drawn. A key is plotted using the style specified in the text file `keysty`. This file could, for instance, contain the two lines `"font=3"` and `"digits(2)=4"` to cause all text in the key to be drawn using PGPLOT font 3 (an italic fount), and 4 digits to be used when formatting the contour values.

```
contour ss443 mode=pe percentiles=[80,90,95] stats keypos=0.02
```

Contours the data array in the NDF called ss443 on the current graphics device. Contours at heights corresponding to the 80, 90 and 95 percentiles are drawn. The key is placed closer to the contour map than usual. Contour statistics are computed.

```
contour skyflux mode=eq ncont=5 dashed=0 pens='colour=red' noclear
```

Contours the data array in the NDF called skyflux on the current graphics device. The contour map is automatically aligned with any existing DATA picture, if possible. Contours at heights corresponding to the 10, 30, 50, 70 and 90 percentiles (of the data within the picture) are drawn in red. Those contours whose values are negative will appear as dashed lines.

```
contour comp=d nokey penrot style="grid=1,title=My data" \
```

Contours the data array in the current NDF on the current graphics device using the current method for height selection. No key is drawn. The appearance of the contours cycles every third contour. A co-ordinate grid is drawn over the plot, and a title of `"My data"` is displayed at the top.

```
contour comp=v mode=fr heights=[10,20,40,80] \
```

Contours the variance array in the current NDF on the current graphics device. Contours at 10, 20, 40 and 80 are drawn.

Notes:

- If no Title is specified via the STYLE parameter, then the TITLE component in the NDF is used as the default title for the annotated axes. Should the NDF not have a TITLE component, then the default title is instead taken from current co-ordinate Frame in the NDF, unless this attribute has not been set explicitly, whereupon the name of the NDF is used as the default title.
- The application stores a number of pictures in the graphics database in the following order: a FRAME picture containing the annotated axes, contours, and key; a KEY picture to store the key if present; and a DATA picture containing just the contours. Note, the FRAME picture is only created if annotated axes or a key has been drawn,

or if non-zero margins were specified using Parameter MARGIN. The world co-ordinates in the DATA picture will be pixel co-ordinates. A reference to the supplied NDF, together with a copy of the WCS information in the NDF are stored in the DATA picture. On exit the current database picture for the chosen device reverts to the input picture.

Related Applications :

KAPPA: WCSFRAME, PICDEF; FIGARO: ICONT, SPECCONT.

Implementation Status:

- Only real data can be processed directly. Other non-complex numeric data types will undergo a type conversion before the contour plot is drawn.
- Bad pixels and quality masking are supported.

CONVOLVE
Convolves a pair of one- or two-dimensional NDFs together

Description:

This application smooths a one- or two-dimensional NDF using a Point-Spread Function given by a second NDF. The output NDF is normalised to the same mean data value as the input NDF (if Parameter NORM is set to TRUE), and is the same size as the input NDF.

Usage:

```
convolve in psf out xcentre ycentre
```

Parameters:

AXES(2) = _INTEGER (Read)

This parameter is only accessed if the NDF has exactly three significant pixel axes. It should be set to the indices of the NDF pixel axes which span the plane in which smoothing is to be applied. All pixel planes parallel to the specified plane will be smoothed independently of each other. The dynamic default is the indices of the first two significant axes in the NDF. []

IN = NDF (Read)

The input NDF containing the image to be smoothed.

NORM = _LOGICAL (Read)

Determines how the output NDF is normalised to take account of the total data sum in the PSF, and of the presence of bad pixels in the input NDF. If TRUE, bad pixels are excluded from the data sum for each output pixel, and the associated weight for the output pixel is reduced appropriately. The supplied PSF is normalised to a total data sum of unity so that the output NDF has the same normalisation as the input NDF. If NORM is FALSE, bad pixels are replaced by the mean value and then included in the convolution as normal. The normalisation of the supplied PSF is left unchanged, and so determines the normalisation of the output NDF. [TRUE]

OUT = NDF (Write)

The output NDF which is to contain the smoothed image.

PSF = NDF (Read)

An NDF holding the Point-Spread Function (PSF) with which the input image is to be smoothed. An error is reported if the PSF contains any bad pixels. The PSF can be centred anywhere within the image (see Parameters XCENTRE and YCENTRE). A constant background is removed from the PSF before use. This background level is equal to the minimum of the absolute value in the four corner pixel values. The PSF is assumed to be zero beyond the bounds of the supplied NDF. It should have the same number of dimensions as the NDF being smoothed, unless the input NDF has three significant dimensions, whereupon the PSF must be two-dimensional. It will be normalised to a total data sum of unity if Parameter NORM is TRUE.

TITLE = LITERAL (Read)

A title for the output NDF. A null (!) value means using the title of the input NDF. [!]

WLIM = _REAL (Read)

If the input array contains bad pixels, and NORM is TRUE, then this parameter may be used to determine the number of good pixels that must be present within the smoothing box before a valid output pixel is generated. It can be used, for example, to prevent output pixels from being generated in regions where there are relatively few good pixels to contribute to the smoothed result.

By default, a null (!) value is used for WLIM, which causes the pattern of bad pixels to be propagated from the input image to the output image unchanged. In this case, smoothed output values are only calculated for those pixels which are not bad in the input image.

If a numerical value is given for WLIM, then it specifies the minimum total weight associated with the good pixels in the smoothing box required to generate a good output pixel (weights for each pixel are defined by the normalised PSF). If this specified minimum weight is not present, then a bad output pixel will result, otherwise a smoothed output value will be calculated. The value of this parameter should lie between 0.0 and 1.0. A value of 0.0 will result in a good output pixel being created even if only one good input pixel contributes to it. A value of 1.0 will result in a good output pixel being created only if all the input pixels which contribute to it are good. See also Parameter NORM. [!]

XCENTRE = _INTEGER (Read)

The *x* pixel index (column number) of the centre of the PSF within the supplied PSF array. The suggested default is the centre of the PSF array. (This is how the PSF command would generate the array.)

YCENTRE = _INTEGER (Read)

The *y* pixel index (line number) of the centre of the PSF within the supplied PSF array. The suggested default is the centre of the PSF array. (This is how the PSF command would generate the array.)

Examples:

```
convolve ccdframe iraspsf ccdlores 50 50
```

The image in the NDF called ccdframe is convolved using the PSF in NDF iraspsf to create the smoothed image ccdlores. The centre of the PSF image in iraspsf is at pixel indices (50, 50). Any bad pixels in the input image are propagated to the output.

```
convolve ccdframe iraspsf ccdlores 50 50 wlim=1.0
```

As above, but good output values are only created for pixels which have no contributions from bad input pixels.

```
convolve ccdframe iraspsf ccdlores \
```

As in the first example except the centre of the PSF is located at the centre of the PSF array.

Notes:

- The algorithm used is based on the multiplication of the Fourier transforms of the input image and PSF image.
- A PSF can be created using the PSF command or MATHS if the PSF is an analytic function.

Related Applications :

KAPPA: BLOCK, FFCLEAN, GAUSMOOTH, MATHS, MEDIAN, PSF; FIGARO: ICONV3, ISMOOTH, IXSMOOTH, MEDFILT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using double-precision floating point.

COPYBAD
Copies bad pixels from one NDF file to another

Description:

This application copies bad pixels from one NDF file to another. It takes in two NDFs (Parameters IN and REF), and creates a third (Parameter OUT) which is a copy of IN, except that any pixel which is set bad in the DATA array of REF, is also set bad in the DATA and VARIANCE (if available) arrays in OUT.

By setting the INVERT Parameter TRUE, the opposite effect can be produced (*i.e.* any pixel that is not set bad in the DATA array of REF, is set bad in OUT and the others are left unchanged).

Usage:

```
copybad in ref out [title]
```

Parameters:

IN = NDF (Read)
NDF containing the data to be copied to OUT.

INVERT = _LOGICAL (Read)
If TRUE, then the bad and good pixels within the reference NDF specified by Parameter REF are inverted before being used (that is, good pixels are treated as bad and bad pixels are treated as good). [FALSE]

OUT = NDF (Write)
The output NDF.

REF = NDF (Read)
NDF containing the bad pixels to be copied to OUT.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

Results Parameters:

NBAD = _INTEGER (Write)
The number of bad pixels copied to the output NDF.

NGOOD = _INTEGER (Write)
The number of pixels not made bad in the output NDF.

Examples:

```
copybad in=a ref=b out=c title="New image"
```

This creates a NDF called c, which is a copy of the NDF called a. Any bad pixels present in the NDF called b are copied into the corresponding positions in c (non-bad pixels in b are ignored). The title of c is "New image".

Notes:

- If the two input NDFs have different pixel-index bounds, then they will be trimmed to match before being processed. An error will result if they have no pixels in common.

Related Applications :

KAPPA: SUBSTITUTE, NOMAGIC, FILLBAD, PASTE, GLITCH.

Implementation Status:

- This routine correctly processes the WCS, AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, and VARIANCE components of an NDF data structure and propagates all extensions.
- The BAD_PIXEL flag is set appropriately.
- All non-complex numeric data types can be handled.

CREFRAME
Generates a test two-dimensional NDF with a selection of several forms

Description:

This application creates a two-dimensional output NDF containing artificial data of various forms (see Parameter MODE). The output NDF can, optionally, have a VARIANCE component describing the noise in the data array (see Parameter VARIANCE), and additionally a randomly generated pattern of bad pixels (see Parameter BADPIX). Bad columns or rows of pixels can also be generated.

Usage:

```
creframe out mode [lbound] [ubound]
      { mean=?
      { background=?  distrib=?  max=?  min=?  ngauss=?  seeing=?
      { mean=?  sigma=?
      { high=?  low=?
   mode
```

Parameters:

BACKGROUND = _REAL (Read)

Background intensity to be used in the generated data array. Must not be negative. (GS mode).

BADCOL = _INTEGER (Read)

The number of bad columns to include. Only accessed if Parameter BADPIX is TRUE. The bad columns are distributed at random using a uniform distribution. [0]

BADPIX = _LOGICAL (Read)

Whether or not bad pixels are to be included. See also Parameters FRACTION, BADCOL and BADROW. [FALSE]

BADROW = _INTEGER (Read)

The number of bad rows to include. Only accessed if Parameter BADPIX is TRUE. The bad rows are distributed at random using a uniform distribution. [0]

DIRN = _INTEGER (Read)

Direction of the ramp. 1 means left to right, 2 is right to left, 3 is bottom to top, and 4 is top to bottom. (RA mode)

DISTRIB = LITERAL (Read)

Radial distribution of the Gaussians to be used (GS mode). Alternatives weightings are:

- "FIX" — fixed distance, and
- "RSQ" — one over radius squared.

`["FIX"]`

FRACTION = _REAL (Read)
Fraction of bad pixels to be included. Only accessed if BADPIX is `TRUE`. `[0.01]`

HIGH = _REAL (Read)
High value used in the generated data array (RA and RL modes).

LBOUND(2) = _INTEGER (Read)
Lower pixel bounds of the output NDF. Only accessed if Parameter LIKE is set to null (`!`).

LIKE = NDF (Read)
An optional template NDF which, if specified, will be used to define the bounds for the output NDF. If a null value (`!`) is given the bounds are obtained via Parameters LBOUND and UBOUND. `[!]`

LOGFILE = LITERAL (Read)
Name of a log file in which to store details of the Gaussians added to the output NDF (GS mode). If a null value is supplied no log file is created. `[!]`

LOW = _REAL (Read)
Low value used in the generated data array (RA and RL modes).

MAX = _REAL (Read)
Peak Gaussian intensity to be used in the generated data array (GS mode).

MEAN = _REAL (Read)
Mean value used in the generated data array (FL, RP and GN modes).

MIN = _REAL (Read)
Lowest Gaussian intensity to be used in the generated data array (GS mode).

MODE = LITERAL (Read)
The form of the data to be generated. The options are as follows.

- `"RR"` — Uniform noise between 0 and 1.
- `"RL"` — Uniform noise between specified limits.
- `"BL"` — A constant value of zero.
- `"FL"` — A specified constant value.
- `"RP"` — Poisson noise about a specified mean.
- `"GN"` — Gaussian noise about a specified mean.
- `"RA"` — Ramped between specified minimum and maximum values and a choice of four directions.
- `"GS"` — A random distribution of two-dimensional Gaussians of defined FWHM and range of maximum peak values on a specified background, with Poissonian noise. There is a choice of spatial distributions for the Gaussians: fixed, or inverse square radially from the array centre. (In essence it is equivalent to a simulated star field.) The *x*-*y* position and peak value of each Gaussian may be stored in a log file, a positions list catalogue, or reported on the screen. Bad pixels may be included randomly, and/or in a column or line of the array.

NGAUSS = _INTEGER (Read)
Number of Gaussian star-like images to be generated (GS mode).

OUT = NDF (Write)
The output NDF.

OUTCAT = FILENAME (Write)
An output catalogue in which to store the pixel co-ordinates of the Gausians in the output NDF (GS mode). If a null value is supplied, no output positions list is produced. [!]

SEEING = _REAL (Read)
Seeing (FWHM) in pixels (not the same as the standard deviation) (GS mode).

SIGMA = _REAL (Read)
Standard deviation of noise to be used in the generated data array (GN mode).

TITLE = LITERAL (Read)
Title for the output NDF. ["KAPPA - Creframe"]

UBOUND(2) = _INTEGER (Read)
Upper pixel bounds of the output NDF. Only accessed if Parameter LIKE is set to null (!).

VARIANCE = _LOGICAL (Read)
If TRUE, a VARIANCE component is added to the output NDF representing the noise added to the field. If a null (!) value is supplied, a default is used which is TRUE for modes which include noise, and FALSE for modes which do not include any noise. [!]

Examples:

```
creframe out=file ubound=[128,128] mode=gs ngauss=5 badpix badcol=2 max=200
min=20 background=20 seeing=1.5
```

Produces a 128×128 pixel data array with 5 gaussians with peak values of 200 counts and a background of 20 counts. There will be two bad columns added to the resulting data.

Notes:

- The Gaussian parameters (GS mode) are not displayed when the message filter environment variable MSG_FILTER is set to QUIET.

Implementation Status:

- The DATA and VARIANCE components of the output NDF have a numerical type of "_REAL" (single-precision floating point).
- This routine does not assign values to any of the following components in the output NDF: LABEL, UNITS, QUALITY, AXIS, WCS.

CSUB
Subtracts a scalar from an NDF data structure

Description:

The routine subtracts a scalar (*i.e.* constant) value from each pixel of an NDF's data array to produce a new NDF data structure.

Usage:

```
csub in scalar out
```

Parameters:

IN = NDF (Read)
Input NDF data structure, from which the value is to be subtracted.

OUT = NDF (Write)
Output NDF data structure.

SCALAR = _DOUBLE (Read)
The value to be subtracted from the NDF's data array.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
csub a 10 b
```

This subtracts ten from the NDF called a, to make the NDF called b. NDF b inherits its title from a.

```
csub title="HD123456" out=b in=a scalar=21.9
```

This subtracts 21.9 from the NDF called a, to make the NDF called b. NDF b has the title `"HD123456"`.

Related Applications :

KAPPA: ADD, CADD, CDIV, CMULT, DIV, MATHS, MULT, SUB.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Huge NDFs are supported.

CUMULVEC
Sums the values cumulatively in a one-dimensional NDF

Description:

This application forms the cumulative sum of the values of a one-dimensional NDF starting from the first to the last element. thus the first output pixel will be unchanged but the second will be the sum of the first two input pixels, third output pixel is the sum of the first three input pixels and so on. Anomalous values may be excluded from the summation by setting a threshold.

Usage:

```
cumulvec in out [thresh]
```

Parameters:

IN = NDF (Read)
The one-dimensional NDF containing the vector to be summed.

OUT = NDF (Write)
The NDF to contain the summed image.

THRESH = _DOUBLE (Read)
The maximum difference between adjacent elements for the summation to ocur. For increments outside the allowed range, the increment becomes zero. If null, `!`, is given, then there is no limit. `[!]`

TITLE = LITERAL (Read)
The title of the output NDF. A null (`!`) value means using the title of the input NDF. `[!]`

Examples:

```
cumulvec gradient profile
```

The one-dimensional NDF called gradient is summed cumulatively to form NDF profile.

```
cumulvec in=gradient out=profile thresh=20
```

As above but only adjacent values separated by less than 20 are included in the summation.

Related Applications :

KAPPA: HISTOGRAM.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.

- Processing of bad pixels and automatic quality masking are supported. Bad pixels are propagated and excluded from the summation.
- All non-complex numeric data types can be handled. Arithmetic is performed using single- or double-precision floating point as appropriate.

CURSOR
Reports the co-ordinates of positions selected using the cursor

Description:

This application reads co-ordinates from the chosen graphics device using a cursor and displays them on your terminal. The selected positions may be marked in various ways on the device (see Parameter PLOT), and can be written to an output positions list so that subsequent applications can make use of them (see Parameter OUTCAT). The format of the displayed positions may be controlled using Parameter STYLE. The pixel data value in any associated NDF can also be displayed (see Parameter SHOWDATA).

Positions may be reported in several different co-ordinate Frames (see Parameter FRAME). Optionally, the corresponding pixel co-ordinates at each position may also be reported (see Parameter SHOWPIXEL).

The picture or pictures within which positions are required can be selected in several ways (see Parameters MODE and NAME).

Restrictions can be made on the number of positions to be given (see Parameters MAXPOS and MINPOS), and screen output can be suppressed (see the "Notes").

Usage:

```
cursor [mode] [name] [outcat] [device]
```

Parameters:

CATFRAME = LITERAL (Read)

A string determining the co-ordinate Frame in which positions are to be stored in the output catalogue associated with Parameter OUTCAT. The string supplied for CATFRAME can be one of the following:

- A domain name such as SKY, AXIS, PIXEL.
- An integer value giving the index of the required Frame.
- An IRAS90 *Sky Co-ordinate System* (SCS) values such as `"EQUAT(J2000)"` (see SUN/163).

If a null (`!`) value is supplied, the positions will be stored in the current Frame. `[!]`

CATEPOCH = _DOUBLE (Read)

The epoch at which the sky positions stored in the output catalogue were determined. It will only be accessed if an epoch value is needed to qualify the co-ordinate Frame specified by COLFRAME. If required, 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.

CLOSE = _LOGICAL (Read)

This parameter is only accessed if Parameter PLOT is set to `"Chain"` or `"Poly"`. If `TRUE`, polygons will be closed by joining the first position to the last position. [current value]

COMP = LITERAL (Read)
The NDF component to be displayed. It may be "Data", "Quality", "Variance", or "Error" (where "Error" is an alternative to "Variance" and causes the square root of the variance values to be displayed). If "Quality" is specified, then the quality values are treated as numerical values (in the range 0 to 255). ["Data"]

DESCRIBE = _LOGICAL (Read)
If TRUE, a detailed description of the co-ordinate Frame in which subsequent positions will be reported is produced each time a position is reported within a new picture. [current value]

DEVICE = DEVICE (Read)
The graphics workstation. This device must support cursor interaction. [current graphics device]

EPOCH = _DOUBLE (Read)
If a 'Sky Co-ordinate System' specification is supplied (using Parameter FRAME) for a celestial co-ordinate system, 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.

FRAME = LITERAL (Read)
A string determining the co-ordinate Frame in which positions are to be reported. When a data array is displayed by an application such as DISPLAY, CONTOUR the WCS information describing the co-ordinate systems known to the data array are stored with the DATA picture in the graphics database. This application can report positions in any of the co-ordinate Frames stored with each picture. The string supplied for FRAME can be one of the following:

- A domain name such as SKY, AXIS, PIXEL. The special domains AGI_WORLD and AGI_DATA are used to refer to the world and data co-ordinate system stored in the AGI graphics database. They can be useful if no WCS information was store with the picture when it was created.
- An integer value giving the index of the required Frame.
- An IRAS90 *Sky Co-ordinate System* (SCS) values such as "EQUAT(J2000)" (see SUN/163).

If a null value (!) is supplied, positions are reported in the co-ordinate Frame which was current when the picture was created. [!]

GEODESIC = _LOGICAL (Read)
This parameter is only accessed if Parameter PLOT is set to "Chain" or "Poly". It specifies whether the curves drawn between positions should be straight lines, or should be geodesic curves. In many co-ordinate Frames geodesic curves will be simple straight lines. However, in others (such as the majority of celestial co-ordinates Frames) geodesic curves will be more complex curves tracing the shortest path between two positions in a non-linear projection. [FALSE]

INFO = _LOGICAL (Read)
If TRUE, then messages are displayed describing the use of the mouse prior to ob-

taining the first position. Note, these informational messages are not suppressed by setting MSG_FILTER environment variable to QUIET. [TRUE]

JUST = LITERAL (Read)
A string specifying the justification to be used when displaying text strings at the supplied cursor positions. This parameter is only accessed if Parameter PLOT is set to "Text". The supplied string should contain 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 supplied position is at the specified point within the displayed text string. ["CC"]

LOGFILE = FILENAME (Write)
The name of the text file in which the formatted co-ordinates of positions selected with the cursor may be stored. This is intended primarily for recording the screen output, and not for communicating positions to subsequent applications (use Parameter OUTCAT for this purpose). A null string (!) means that no file is created. [!]

MARKER = _INTEGER (Read)
This parameter is only accessed if Parameter PLOT is set to "Chain" or "Mark". It specifies the symbol with which each position should be marked, and should be given as an integer PGPLOT marker type. For instance, 0 gives a box, 1 gives a dot, 2 gives a cross, 3 gives an asterisk, 7 gives a triangle. The value must be larger than or equal to −31. [current value]

MAXPOS = _INTEGER (Read)
The maximum number of positions which may be supplied before the application terminates. The number must be in the range 1 to 200. [200]

MINPOS = _INTEGER (Read)
The minimum number of positions which may be supplied. The user is asked to supply more if necessary. The number must be in the range 0 to the value of Parameter MAXPOS. [0]

MODE = LITERAL (Read)
The method used to select the pictures in which cursor positions are to be reported. There are three options.

- "Current" — reports positions within the current picture in the AGI database. If a position does not lie within the current picture, an extrapolated position is reported, if possible.
- "Dynamic" — reports positions within the top-most picture under the cursor in the AGI database. Thus the second and subsequent cursor hits may result in the selection of a new picture.
- "Anchor" — lets the first cursor hit select the picture in which all positions are to be reported. If a subsequent cursor hit falls outside this picture, an extrapolated position is reported if possible.

["Dynamic"]

NAME = LITERAL (Read)
Only pictures of this name are to be selected. For instance, if you want positions in a DATA picture which is covered by a transparent FRAME picture, then you could specify NAME="DATA". A null (!) or blank string means that pictures of all names may be selected. NAME is ignored when MODE="Current". [!]

OUTCAT = FILENAME (Write)
An output catalogue in which to store the valid selected positions. The catalogue has the form of a positions list such as created by application LISTMAKE. Only positions in the first selected picture are recorded. This application uses the conventions of the CURSA package for determining the format of the catalogue. If a file type of `.fit` is given, then the catalogue is stored as a FITS binary table. If a file type of `.txt` is given, then the catalogue is stored in a text file in `"Small Text List"` (STL) format. If no file type is given, then `.fit` is assumed. If a null value is supplied, no output positions list is produced. See also Parameter CATFRAME. `[!]`

PLOT = LITERAL (Read)
The type of graphics to be used to mark the selected positions which have valid co-ordinates. The appearance of these graphics (colour, size, *etc.*) is controlled by the STYLE parameter. PLOT can take any of the following values:

- `"None"` — No graphics are produced.
- `"Mark"` — Each position is marked by the symbol specified by Parameter MARKER.
- `"Poly"` — Causes each position to be joined by a line to the previous position. These lines may be simple straight lines or geodesic curves (see Parameter GEODESIC). The polygons may optionally be closed by joining the last position to the first (see Parameter CLOSE).
- `"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. Parameters MARKER, GEODESIC and CLOSE are used to specify the symbols and lines to use.
- `"Box"` — A rectangular box with edges parallel to the edges of the graphics device is drawn with the specified position at one corner, and the previously specified position at the diagonally opposite corner.
- `"Vline"` — A vertial line is drawn through each specified position, extending the entire height of the selected picture.
- `"Hline"` — A horizontal line is drawn through each specified position, extending the entire width of the selected picture.
- `"Cross"` — A combination of `"Vline"` and `"Hline"`.
- `"Text"` — A text string is used to mark each position. The string is drawn horizontally with the justification specified by Parameter JUST. The strings to use for each position are specified using Parameter STRINGS.

[current value]

SHOWDATA = _LOGICAL (Read)
If `TRUE`, the pixel value within the displayed NDF is reported for each selected position. This is only possible if the picture within which position are being selected contains a reference to an existing NDF. The NDF array component to be displayed is selected via Parameter COMP. `[FALSE]`

SHOWPIXEL = _LOGICAL (Read)
If `TRUE`, the pixel co-ordinates of each selected position are shown on a separate line, following the co-ordinates requested using Parameter FRAME. If pixel co-ordinates are being displayed anyway (see Parameter FRAME) then a value of `FALSE` is used for. SHOWPIXEL. [current value]

STRINGS = LITERAL (Read)

A group of text strings which are used to mark the supplied positions if Parameter PLOT is set to "TEXT". The first string in the group is used to mark the first position, the second string is used to mark the second position, *etc.* If more positions are given than there are strings in the group, then the extra positions will be marked with an integer value indicating the index within the list of supplied positions. If a null value (!) is given for the parameter, then all positions will be marked with integer indices, starting at 1.

A comma-separated list should be given in which each element is either a marker string, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Note, strings within text files can be separated by new lines as well as commas.

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use when drawing the graphics specified by Parameter PLOT. The format of the positions reported on the screen may also be controlled.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

In addition to the attributes which control the appearance of the graphics (Colour, Font, *etc.*), the following attributes may be set in order to control the appearance of the formatted axis values reported on the screen: Format, Digits, Symbol, Unit. These may be suffixed with an axis number (*e.g.* "Digits(2)") to refer to the values displayed for a specific axis. [current value]

Results Parameters:

LASTDIM = _INTEGER (Write)

The number of axis values written to Parameter LASTPOS.

LASTPOS() = _DOUBLE (Write)

The unformatted co-ordinates of the last valid position selected with the cursor, in the co-ordinate Frame which was used to report the position. The number of axis values is written to output Parameter LASTDIM.

NUMBER = _INTEGER (Write)
The number of positions selected with the cursor (excluding invalid positions).

Examples:

```
cursor frame=pixel
```

This obtains co-ordinates within any visible picture for the current graphics device by use of the cursor. Positions are reported in pixel co-ordinates if available, and in the current co-ordinate Frame of the picture otherwise.

```
cursor frame=pixel outcat=a catframe=gal
```

Like the previous example, except that, in addition to being displayed on the screen, the positions are transformed into galactic co-ordinates and stored in FITS binary table called `a.FIT`, together with any associated WCS information.

```
cursor frame=equat(J2010)
```

This obtains co-ordinates within any visible picture for the current graphics device by use of the cursor. Positions are reported in equatorial RA/DEC co-ordinates (referenced to the J2010 equinox) if available, and in the current co-ordinate Frame of the picture otherwise.

```
cursor describe plot=mark marker=3 style="colour=red,size=2"
```

As above except, positions are always reported in the current co-ordinate Frame of each picture. The details of these co-ordinate Frames are described as they are used. Each selected point is marked with PGPLOT marker 3 (an asterisk). The markers are red and are twice the default size.

```
cursor current maxpos=2 minpos=2 plot=poly outcat=slice
```

Exactly two positions are obtained within the current picture, and are joined with a straight line. The positions are written to a FITS binary catalogue called `slice.FIT`. The catalogue may be used to communicate the positions to later applications (LISTSHOW, PROFILE, *etc.*).

```
cursor name=data style="^mystyle,digits(1)=5,digits(2)=7"
```

This obtains co-ordinates within any visible DATA picture on the current graphics device. The style to use is read from text file mystyle, but is then modified so that five digits are used to format axis-1 values, and seven to format axis-2 values.

```
cursor plot=box style="width=3,colour=red" maxpos=2 minpos=2
```

Exactly two positions must be given using the cursor, and a red box is drawn joining the two positions. The lines making up the box are three times the default width.

```
cursor plot=text style="size=2,textbackcolour=clear"
```

Positions are marked using integer values, starting at 1 for the first position. The text drawn is twice as large as normal, and the background is not cleared before drawing the text.

Notes:

- The unformatted values stored in the output Parameter LASTPOS, may not be in the same units as the formatted values shown on the screen and logged to the log file. For instance, unformatted celestial co-ordinate values are stored in radians.
- The current picture is unchanged by this application.
- In DYNAMIC and ANCHOR modes, if the cursor is situated at a position where there are no pictures of the selected name, the co-ordinates in the BASE picture are reported.
- Pixel co-ordinates are formatted with 1 decimal place unless a format has already been specified by setting the Format attributes for the axes of the PIXEL co-ordinate Frame (*e.g.* using application WCSATTRIB).
- Positions can be removed (the instructions state how), starting from the most-recent one. Such positions are excluded from the output positions list and log file (if applicable). If graphics are being used to mark the positions, then removed positions will be highlighted by drawing a marker of type 8 (a circle containing a cross) over the removed positions in a different colour.
- The positions are not displayed on the screen when the message filter environment variable MSG_FILTER is set to `QUIET`. The creation of output parameters and files is unaffected by MSG_FILTER. The display of informational messages describing the use of the cursor is controlled by the Parameter INFO.

Related Applications :

KAPPA: LISTSHOW, LISTMAKE, PICCUR; FIGARO: ICUR, IGCUR.

DISPLAY
Displays a one- or two-dimensional NDF

Description:

This application displays a one- or two-dimensional NDF as an image on the current graphics device. The minimum and maximum data values to be displayed can be selected in several ways (see Parameter MODE). Data values outside these limits are displayed with the colour of the nearest limit. A key showing the relationship between colour and data value can be displayed (see Parameter KEY).

Annotated axes or a simple border can be drawn around the image (see Parameters AXES and BORDER). The appearance of these may be controlled in detail (see Parameters STYLE and BORSTYLE).

A specified colour lookup table may optionally be loaded prior to displaying the image (see Parameter LUT). For devices which reset the colour table when opened (such as PostScript files), this may be the only way of controlling the colour table.

The image is produced within the current graphics database picture. The co-ordinates at the centre of the image, and the scale of the image can be controlled using Parameters CENTRE, XMAGN and YMAGN. Only the parts of the image that lie within the current picture are visible; the rest is clipped. The image is padded with bad pixels if necessary.

Usage:

```
display in [comp] clear [device] mode [centre] [xmagn] [ymagn] [out]
     { low=?  high=?
     { percentiles=?
     { sigmas=?
   mode
```

Parameters:

AXES = _LOGICAL (Read)

`TRUE` if labelled and annotated axes are to be drawn around the image. These display co-ordinates in the current co-ordinate Frame of the supplied NDF, and may be changed using application WCSFRAME (see also Parameter USEAXIS). The width of the margins left for the annotation may be controlled using Parameter MARGIN. The appearance of the axes (colours, founts, *etc.*) can be controlled using the STYLE Parameter. [current value]

BADCOL = LITERAL (Read)

The colour with which to mark any bad (*i.e.* missing) pixels in the display. There are a number of options described below.

- `"MAX"` — The maximum colour index used for the display of the image.
- `"MIN"` — The minimum colour index used for the display of the image.

- An integer — The actual colour index. It is constrained between 0 and the maximum colour index available on the device.
- 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 colour code such as `#ff002d`.

If the colour is to remain unaltered as the lookup table is manipulated choose an integer between 0 and 15, or a named colour. The suggested default is the current value. [current value]

BORDER = _LOGICAL (Read)

`TRUE` if a border is to be drawn around the regions of the displayed image containing valid co-ordinates in the current co-ordinate Frame of the NDF. For instance, if the NDF contains an Aitoff all-sky map, then an elliptical border will be drawn if the current co-ordinate Frame is galactic longitude and latitude. This is because pixels outside this ellipse have undefined positions in galactic co-ordinates. If, instead, the current co-ordinate Frame had been pixel co-ordinates, then a simple box would have been drawn containing the whole image. This is because every pixel has a defined position in pixel co-ordinates. The appearance of the border (colour, width, *etc.*) can be controlled using Parameter BORSTYLE. [current value]

BORSTYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the border (see Parameter BORDER).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported). [current value]

CENTRE = LITERAL (Read)

The co-ordinates of the data pixel to be placed at the centre of the image, in the current co-ordinate Frame of the NDF (supplying a colon `":"` will display details of the current co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas. See also Parameter USEAXIS. A null (`!`) value causes the centre of the image to be used. `[!]`

CLEAR = _LOGICAL (Read)

`TRUE` if the current picture is to be cleared before the image is displayed. [current value]

COMP = LITERAL (Read)
The NDF array component to be displayed. It may be "Data", "Quality", "Variance", or "Error" (where "Error" is an alternative to "Variance" and causes the square root of the variance values to be displayed). If "Quality" is specified, then the quality values are treated as numerical values (in the range 0 to 255). ["Data"]

DEVICE = DEVICE (Read)
The name of the graphics device used to display the image. The device must have at least 24 colour indices or grey-scale intensities. [current graphics device]

FILL = _LOGICAL (Read)
If FILL is set to TRUE, then the image will be 'stretched' to fill the current picture in both directions. This can be useful when displaying images with markedly different dimensions, such as two-dimensional spectra. The dynamic default is TRUE if the array being displayed is one-dimensional, and FALSE otherwise. []

HIGH = _DOUBLE (Read)
The data value corresponding to the highest pen in the colour table. All larger data values are set to the highest colour index when HIGH is greater than LOW, otherwise all data values greater than HIGH are set to the lowest colour index. The dynamic default is the maximum data value. There is an efficiency gain when both LOW and HIGH are given on the command line, because the extreme values need not be computed. (Scale mode)

IN = NDF (Read)
The input NDF structure containing the data to be displayed.

KEY = _LOGICAL (Read)
TRUE if a key to the colour table is to be produced to the right of the display. This can take the form of a colour ramp, a coloured histogram of pen indices, or graphs of RGB intensities, all annotated with data value. The form and appearance of this key can be controlled using Parameter KEYSTYLE, and its horizontal position can be controlled using Parameter KEYPOS. If the key is required in a different location, set KEY=NO and use application LUTVIEW after displaying the image. [TRUE]

KEYPOS(2) = _REAL (Read)
The first element gives the gap between the right-hand edge of the display and the left-hand edge of the key, as a fraction of the width of the current picture. If a key is produced, then the right-hand margin specified by Parameter MARGIN is ignored, and the value supplied for KEYPOS is used instead.

The second element gives the vertical position of the key as a fractional value in the range zero to one: zero puts the key as low as possible, one puts it as high as possible. A negative value (no lower than -1) causes the key to match the height of the display image. This may mean any text, like a label, for the horizontal axis may not appear, though if AXES is TRUE there is usually room. [current value]

KEYSTYLE = GROUP (Read)
A group of attribute settings describing the plotting style to use for the key (see Parameter KEY).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which

they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

Axis 1 is always the *data value* axis. So for instance, to set the label for the data-value axis, assign a value to "Label(1)" in the supplied style.

To get a ramp key (the default), specify "form=ramp". To get a histogram key (a coloured histogram of pen indices), specify "form=histogram". To get a graph key (three curves of RGB intensities), specify "form=graph". If a histogram key is produced, the population axis can be either logarithmic or linear. To get a logarithmic population axis, specify "logpop=1". To get a linear population axis, specify "logpop=0" (the default). To annotate the long axis with pen numbers instead of pixel value, specify "pennums=1" (the default, "pennums=0", shows pixel values). [current value]

LOW = _DOUBLE (Read)

The data value corresponding to the lowest pen in the colour table. All smaller data values are set to the lowest colour index when LOW is less than HIGH, otherwise all data values smaller than LOW are set to the highest colour index. The dynamic default is the minimum data value. There is an efficiency gain when both LOW and HIGH are given on the command line, because the extreme values need not be computed. (Scale mode)

LUT = NDF (Read)

Name of the NDF containing a colour lookup table in its Data array; the lookup table is written to the graphics device's colour table. The purpose of this parameter is to provide a means of controlling the appearance of the image on certain devices, such as colour printers, that do not have a dynamic colour table (*i.e.* the colour table is reset when the device is opened). If used with dynamic devices (such as X-windows), the new colour table remains after this application has completed. A null value (!) causes the existing colour table to be used.

The LUT must be two-dimensional, the dimension of the first axis being 3, and the second being arbitrary. The method used to compress or expand the colour table if the second dimension is different from the number of unreserved colour indices is controlled by Parameter NN. Also the LUT's values must lie in the range 0.0–1.0. [!]

MARGIN(4) = _REAL (Read)

The widths of the margins to leave around the image for axis annotations, given as fractions of the corresponding dimension of the current picture. The actual margins used may be increased to preserve the aspect ratio of the data. 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. If a null (`!`) value is supplied, the value used is (for all edges); `0.15` if annotated axes are being produced; 0.04, if a simple border is being produced; and 0.0 if neither border nor axes are being produced. [current value]

MODE = LITERAL (Read)

The method by which the maximum and minimum data values to be displayed are chosen. The options are as follows.

- `"Current"` — The image is scaled between the upper and lower limits that were used by the previous invocation of DISPLAY. If the previous scaling limits cannot be determined, the MODE value reverts to `"Scale"`.
- `"Faint"` — The image is scaled between the mean data value minus one standard deviation and the mean data value plus seven standard deviations. The scaling values are reported so that the faster Scale mode may be utilised later.
- `"Flash"` — The image is flashed on to the screen without any scaling at all. This is the fastest option.
- `"Percentiles"` — The image is scaled between the data values corresponding to two percentiles. The scaling values are reported so that the faster Scale mode may be used later.
- `"Range"` — The image is scaled between the minimum and maximum data values.
- `"Scale"` — You define the upper and lower limits between which the image is to be scaled. The application reports the maximum and the minimum data values for reference and makes these the suggested defaults.
- `"Sigmas"` — The image is scaled between two standard-deviation limits. The scaling values used are reported so that the faster Scale mode may be utilised later.

NN = _LOGICAL (Read)

If `TRUE` the input lookup table is mapped to the colour table by using the nearest-neighbour method. This preserves sharp edges and is better for lookup tables with blocks of colour. If NN is `FALSE`, linear interpolation is used, and this is suitable for smoothly varying colour tables. NN is ignored unless LUT is not null. [`FALSE`]

NUMBIN = _INTEGER (Read)

The number of histogram bins used to compute percentiles for scaling. (Percentiles mode) [`2048`]

OUT = NDF (Write)

A scaled copy of the displayed section of the image. Values in this output image are integer colour indices shifted to exclude the indices reserved for the palette (*i.e.* the value zero refers to the first colour index following the palette). The output NDF is intended to be used as the input data in conjunction with SCALE=`FALSE`. If a null value (`!`) is supplied, no output NDF will be created. This parameter is not accessed when SCALE=`FALSE`. [`!`]

PENRANGE(2) = _REAL (Read)

The range of colour indices ("pens") to use. The supplied values are fractional values where zero corresponds to the lowest available colour index and 1.0 corresponds to

the highest available colour index. The default value of [0.0,1.0] thus causes the full range of colour indices to be used. Note, if Parameter LUT is null (!) or Parameter SCALE is FALSE then this parameter is ignored and the fill range of pens is used. [0.0,1.0]

PERCENTILES(2) = _REAL (Read)
The percentiles that define the scaling limits. For example, [25,75] would scale between the quartile values. (Percentile mode)

SCALE = _LOGICAL (Read)
If TRUE the input data are to be scaled according to the value of Parameter MODE. If it is FALSE, MODE is ignored, and the input data are displayed as is (*i.e.* the data values are simply converted to integer type and used as indices into the colour table). A value of zero refers to the first pen following the palette. A FALSE value is intended to be used with data previously scaled by this or similar applications which have already performed the required scaling (see Parameter OUT). It provides the quickest method of image display within this application. [TRUE]

SIGMAS(2) = _REAL (Read)
The standard-deviation bounds that define the scaling limits. To obtain values either side of the mean both a negative and a positive value are required. Thus [-2,3] would scale between the mean minus two and the mean plus three standard deviations. [3,-2] would give the negative of that.

SQRPIX = _LOGICAL (Read)
If TRUE, then the default value for YMAGN equals the value supplied for XMAGN, resulting in all pixels being displayed as squares on the display surface. If a FALSE value is supplied for SQRPIX, then the default value for YMAGN is chosen to retain the pixels original aspect ratio at the centre of the image. [current value]

STYLE = GROUP (Read)
A group of attribute settings describing the plotting style to use for the annotated axes (see Parameter AXES).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported). [current value]

USEAXIS = GROUP (Read)
USEAXIS is only accessed if the current co-ordinate Frame of the NDF has more than two axes. A group of two strings should be supplied specifying the two axes which are to be used when annotating the image, and when supplying a value for Parameter CENTRE. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (`!`) value is supplied, the axes with the same indices as the two used pixel axes within the NDF are used. `[!]`

XMAGN = _REAL (Read)
The horizontal magnification for the image. The default value of `1.0` corresponds to 'normal' magnification in which the the image fills the available space in at least one dimension. A value larger than 1.0 makes each data pixel wider. If this results in the image being wider than the available space then the image will be clipped to display fewer pixels. See also Parameters YMAGN, CENTRE, SQRPIX, and FILL. `[1.0]`

YMAGN = _REAL (Read)
The vertical magnification for the image. A value of `1.0` corresponds to 'normal' magnification in which the image fills the available space in at least one dimension. A value larger than 1.0 makes each data pixel taller. If this results in the image being taller than the available space then the image will be clipped to display fewer pixels. See also Parameters XMAGN, CENTRE, and FILL. If a null ((`!`) value is supplied, the default value used depends on Parameter SQRPIX. If SQRPIX is `TRUE`, the default YMAGN value used is the value supplied for XMAGN. This will result in each pixel occupying a square area on the screen. If SQRPIX is `FALSE`, then the default value for YMAGN is chosen so that each pixel occupies a rectangular area on the screen matching the pixel aspect ratio at the centre of the image, determined within the current WCS Frame. `[!]`

Results Parameters:

SCAHIGH = _DOUBLE (Write)
On exit, this holds the data value which corresponds to the maximum colour index in the displayed image. In Flash mode or when there is no scaling the highest colour index is returned.

SCALOW = _DOUBLE (Write)
The data value scaled to the minimum colour index for display. In Flash mode or when there is no scaling the lowest colour index is used. The current display linear-scaling minimum is set to this value.

Examples:
```
display ngc6872 mode=p percentiles=[10,90] noaxes
```

Displays the NDF called ngc6872 on the current graphics device device. The scaling is between the 10 and 90 per cent percentiles of the image. No annotated axes are produced.

```
display vv256 mode=flash noaxes border borstyle="colour=blue,style=2"
```

Displays the NDF called vv256 on the current graphics device device. There is no scaling of the data; instead the modulus of each pixel with respect to the number of colour-table indices is shown. No annotated axes are drawn, but a blue border is drawn around the image using PGPLOT line style number 2 (*i.e.* dashed lines).

```
display mode=fa axes style="^sty,grid=1" margin=0.2 clear out=video \
```

Displays the current NDF DATA component with annotated axes after clearing the current picture on the current graphics device device. The appearance of the axes is specified in the text file `sty`, but this is modified by setting the Grid attribute to 1 so that a co-ordinate grid is drawn across the plot. The margins around the image containing the axes are made slightly wider than normal. The scaling is between the -1 and $+7$ standard deviations of the image around its mean. The scaled data are stored in an NDF called video.

```
display kn26 axes key keypos=[0.0,-1.0] keystyle=^key.sty \
```

Displays the NDF called kn26 using the current scaling, surrounded by axes. It adds a colour-table key to the right that abuts the data picture and is aligned vertically with the image. The plot attributes set in the text file `key.sty` controls the appearance of the key.

```
display video noscale \
```

Displays the DATA component of the NDF called video (created in the previous example) without scaling within the current picture on the current graphics device.

```
display in=cgs4a comp=v mode=sc low=1 high=5.2 device=xwindows
```

Displays the VARIANCE component of NDF cgs4a on the xwindows device, scaling between 1 and 5.2.

```
display mydata centre="12:23:34 -22:12:23" xmagn=2 badcol="red" \
```

Displays the NDF called mydata centred on the position RA=12:23:34, DEC=-22:12:23. This assumes that the current co-ordinate Frame in the NDF is an equatorial (RA/DEC) Frame. The image is displayed with a magnification of 2 so that each data pixel appears twice as large (on each axis) as normal. Fewer data pixels may be displayed to ensure the image fits within the available space in the current picture. The current scaling is used, and bad pixels are shown in red.

```
display ngc6872 mode=ra device=lj250 lut=pizza
```

Displays the NDF called ngc6872 on the LJ250 device. The lookup table in the NDF called pizza is mapped on the LJ250's colour table. The scaling is between the minimum and maximum of the image.

Notes:

- For large images the resolution of the graphics device may allow only a fraction of the detail in the data to be plotted. Therefore, large images will be compressed by block averaging when this can be done without loss of resolution in the displayed image. This saves time scaling the data and transmitting them to the graphics device. Note that the default values for Parameters LOW and HIGH are the minimum and maximum values in the compressed floating-point data.
- If no Title is specified via the STYLE parameter, then the TITLE component in the NDF is used as the default title for the annotated axes. Should the NDF not have a TITLE component, then the default title is instead taken from current co-ordinate Frame in the NDF, unless this attribute has not been set explicitly, whereupon the name of the NDF is used as the default title.
- The application stores a number of pictures in the graphics database in the following order: a FRAME picture containing the annotated axes, the image area, and the border; if there is a key, a KEY picture encompassing the key and its annotations; and a DATA picture containing just the image area. Note, the FRAME picture is only created if annotated axes or a border have been drawn, or if non-zero margins were specified using Parameter MARGIN. The world co-ordinates in the DATA picture will be pixel co-ordinates. A reference to the supplied NDF, together with a copy of the WCS information in the NDF are stored in the DATA picture. On exit the current database picture for the chosen device reverts to the input picture.
- The data type of the output NDF depends on the number of colour indices: _UBYTE for no more than 256, _UWORD for 257 to 65535, and _INTEGER otherwise. The output NDF will not contain any extensions, UNITS, QUALITY, and VARIANCE; but LABEL, TITLE, WCS and AXIS information are propagated from the input NDF. The output NDF does not become the new current data array. It is a Simple NDF (because the bad-pixel flag is set to false in order to access the maximum colour index, and to handle sections), therefore only NDF-compliant applications can process it.

Related Applications :

KAPPA: WCSFRAME, PICDEF, LUTVIEW; FIGARO: IGREY, IMAGE, MOVIE.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, WCS, and UNITS components of the input NDF.
- Processing of bad pixels and automatic quality masking are supported.

- This application will handle data in all numeric types, though type conversion to integer will occur for unsigned byte and word images. However, when there is no scaling only integer data will not be type converted, but this is not expensive for the expected byte-type data.

DIV
Divides one NDF data structure by another

Description:

The routine divides one NDF data structure by another pixel-by-pixel to produce a new NDF.

Usage:

```
div in1 in2 out
```

Parameters:

IN1 = NDF (Read)
First NDF, to be divided by the second NDF.

IN2 = NDF (Read)
Second NDF, to be divided into the first NDF.

OUT = NDF (Write)
Output NDF to contain the ratio of the two input NDFs.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN1 to be used instead. `[!]`

Examples:

```
div a b c
```

This divides the NDF called a by the NDF called b, to make the NDF called c. NDF c inherits its title from a.

```
div out=c in1=a in2=b title="Normalised data"
```

This divides the NDF called a by the NDF called b, to make the NDF called c. NDF c has the title `"Normalised data"`.

Notes:

If the two input NDFs have different pixel-index bounds, then they will be trimmed to match before being divided. An error will result if they have no pixels in common.

Related Applications :

KAPPA: ADD, CADD, CDIV, CMULT, CSUB, MATHS, MULT, SUB.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.

- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Calculations will be performed using either real or double precision arithmetic, whichever is more appropriate. If the input NDF structures contain values with other data types, then conversion will be performed as necessary.
- Huge NDFs are supported.

DRAWNORTH
Draws arrows parallel to the axes

Description:

This application draws a pair of arrows on top of a previously displayed DATA picture which indicate the directions of the labelled axes in the underlying picture, at the position specified by Parameter ORIGIN. For instance, if the underlying picture has axes labelled with celestial co-ordinates, then the arrows will by default indicate the directions of north and east. The appearance of the arrows, including the labels attached to each arrow, may be controlled using the STYLE parameter. The picture area behind the arrows may optionally be cleared before drawing the arrows (see Parameter BLANK).

Usage:

```
drawnorth [device] [length] [origin]
```

Parameters:

ARROW = _REAL (Read)

The size of the arrow heads are specified by this parameter. Simple lines can be drawn by setting the arrow head size to zero. The value should be expressed as a fraction of the largest dimension of the underlying DATA picture. [current value]

BLANK = _LOGICAL (Read)

If `TRUE`, then the area behind the arrows is blanked before the arrows are drawn. This is done by drawing a rectangle filled with the current background colour of the selected graphics device. The size of the blanked area can be controlled using Parameter BLANKSIZE. `[FALSE]`

BLANKSIZE = _REAL (Read)

Specifies the size of the blanked area (see Parameter BLANK). A value of `1.0` results in the blanked area being just large enough to contain the drawn arrows and labels. Values larger than 1.0 introduce a blank margin around the drawn arrows and labels. This parameter also specifies the size of the picture stored in the graphics database. `[1.05]`

DEVICE = DEVICE (Read)

The plotting device. [Current graphics device]

EPOCH = _DOUBLE (Read)

If a 'Sky Co-ordinate System' specification is supplied (using Parameter FRAME) for a celestial co-ordinate system, 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.

FRAME = LITERAL (Read)

Specifies the co-ordinate Frame to which the drawn arrows refer. If a null (`!`) value is supplied, the arrows are drawn parallel to the two axes which were used to annotate the previously displayed picture. If the arrows are required to be parallel to the axes

of some other Frame, the required Frame should be specified using this parameter. The string supplied for FRAME can be one of the following options.

- A domain name such as SKY, AXIS, PIXEL.
- An integer value giving the index of the required Frame.
- An IRAS90 *Sky Co-ordinate System* (SCS) values such as `"EQUAT(J2000)"` (see SUN/163).

An error will be reported if a co-ordinate Frame is requested which is not available in the previously displayed picture. If the selected Frame has more than two axes, the Parameter USEAXIS will determine the two axes which are to be used. `[!]`

LENGTH(2) = _REAL (Read)

The lengths of the arrows, expressed as fractions of the largest dimension of the underlying DATA picture. If only one value is supplied, both arrows will be drawn with the given length. One of the supplied values can be set to zero if only a single arrow is required. [current value]

OFRAME = LITERAL (Read)

Specifies the co-ordinate Frame in which the position of the arrows will be supplied (see Parameter ORIGIN). The following Frames will always be available.

- `"GRAPHICS"` — gives positions in millimetres from the bottom-left corner of the plotting surface.
- `"BASEPIC"` — gives positions in a normalised system in which the bottom-left corner of the plotting surface is (0, 0) and the shortest dimension of the plotting surface has length 1.0. The scales on the two axes are equal.
- `"CURPIC"` — gives positions in a normalised system in which the bottom-left corner of the underlying DATA picture is (0, 0) and the shortest dimension of the picture has length 1.0. The scales on the two axes are equal.
- `"NDC"` — gives positions in a normalised system in which the bottom-left corner of the plotting surface is (0, 0) and the top-right corner is (1, 1).
- `"CURNDC"` — gives positions in a normalised system in which the bottom-left corner of the underlying DATA picture is (0, 0) and the top-right corner is (1, 1).

Additional Frames will be available, describing the co-ordinates systems known to the data displayed within the underlying picture. These could include PIXEL, AXIS, SKY, for instance, but the exact list will depend on the displayed data. If a null value is supplied, the ORIGIN position should be supplied in the Frame used to annotate the underlying picture (supplying a colon `":"` will display details of this co-ordinate Frame). `["CURNDC"]`

ORIGIN = LITERAL (Read)

The co-ordinates at which to place the origin of the arrows, in the Frame specified by Parameter OFRAME. If a null (`!`) value is supplied, OFRAME is ignored and the arrows are situated at a default position near one of the corners, or at the centre. The supplied position can be anywhere within the current picture. An error is reported if the arrows and labels cannot be drawn at any of these positions. `[!]`

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the vectors and annotated axes.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the arrows is controlled by the attributes Colour(Axes), Width(Axes), *etc.* (the synonym Arrows may be used in place of Axes).

The text of the label to draw against each arrow is specified by the Symbol(1) and Symbol(2) attributes. These default to the corresponding attributes of the underlying picture. The appearance of these labels can be controlled using the attributes Font(TextLab), Size(TextLab), *etc*. The gap between the end of the arrow and the corresponding label can be controlled using attribute TextLabGap. The drawing of labels can be suppressed using attribute TextLab. [current value]

USEAXIS = GROUP (Read)

USEAXIS is only accessed if the co-ordinate Frame selected using Parameter FRAME has more than two axes. A group of two strings should be supplied specifying the two axes to which the two drawn arrows should refer. Each axis can be specified using one of the following options.

- An integer index of an axis within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- An axis Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (!) value is supplied, the first two axes of the Frame are used. [!]

Examples:

drawnorth

Draws a pair of arrows indicating the directions of the axes of the previously displayed image, contour map, *etc*. The arrows are drawn at the top left of the picture. The current values for all other parameters are used.

```
drawnorth blank origin="0.5,0.5" style='TextBackColour=clear'
```

As above, but blanks out the picture area behind the arrows, and positions them in the middle of the underlying DATA picture. In addition, the text labels are drawn with a clear background so that the underlying image can seen around the text.

```
drawnorth blank blanksize=1.2 oframe=pixel origin="150,250"
```

As above, but positions the arrows at pixel co-ordinates (150,250), and blanks out a larger area around the arrows.

```
drawnorth blank oframe=!  origin="10:12:34,-12:23:37"
```

As above, but positions the arrows at RA=10:12:34 and DEC=-12:23:37 (this assumes the underlying picture was annotated with RA and DEC axes).

```
drawnorth length=[0.1,0] style='colour(arrows)=red'
```

Draws the axis-1 arrow with length equal to 0.1 of the longest dimension of the underlying picture, but does not draw the axis-2 arrow. Both arrows are drawn red.

```
drawnorth style='textlab=0'
```

Draws both arrows but does not draw any text labels.

```
drawnorth style="'Size(TextLab1)=2,Symbol(1)=A,Symbol(2)=B'"
```

Draws arrows with labels `"A"` and `"B"`, using characters of twice the default size for the label for the first axis.

Notes:

- An error is reported if there is no existing DATA picture within the current picture on the selected graphics device.
- The application stores a picture in the graphics database with name KEY which contains the two arrows. On exit the current database picture for the chosen device reverts to the input picture.

DRAWSIG
Draws ±n standard-deviation lines on a line plot

Description:

This routine draws straight lines on an existing plot stored in the graphics database, such as produced by LINPLOT or HISTOGRAM. The lines are located at arbitrary multiples of the standard deviation (NSIGMA) either side of the mean of a given dataset. The default dataset is the one used to draw the existing plot. You can plot the lines horizontally or vertically as appropriate. The lines extend the full width or height of the plot's data area. Up to five different multiples of the standard deviation may be presented in this fashion. Each line can be drawn with a different style (see Parameter STYLE).

The application also computes statistics for those array values that lie between each pair of plotted lines. In other words it finds the statistics between clipping limits defined by each 2∗NSIGMA range centred on the unclipped mean.

The task tabulates NSIGMA, the mean, the standard deviation, and the error in the mean after the application of each pair of clipping limits. For comparison purposes the first line of the table presents these values without clipping. The table is written at the normal reporting level.

Usage:

```
drawsig ndf nsigma [axis] [comp]
```

Parameters:

AXIS = LITERAL (Read)

The orientation of the lines, or put another way, the axis which represents data value. Thus the allowed values are `"Horizontal"`, `"Vertical"`, `"X"`, or `"Y"`. `"Horizontal"` is equivalent to `"Y"` and `"Vertical"` is a synonym for `"X"`. On LINPLOT output AXIS would be `"Y"`, but on a plot from HISTOGRAM it would be `"X"`. The suggested default is the current value. `["Y"]`

COMP = LITERAL (Read)

The name of the NDF array component from which to derive the mean and standard deviation used to draw the lines: `"Data"`, `"Error"`, `"Quality"` or `"Variance"` (where `"Error"` is the alternative to `"Variance"` and causes the square root of the variance values to be taken before computing the statistics). If `"Quality"` is specified, then the quality values are treated as numerical values (in the range 0 to 255). `["Data"]`

DEVICE = DEVICE (Read)

The graphics device to draw the sigma lines on. [Current graphics device]

NDF = NDF (Read)

The NDF structure containing the data array whose error limits are to be plotted. Usually this parameter is not defined thereby causing the statistics to be derived from the dataset used to draw the plot. If, however, you had plotted a section of a dataset but wanted to plot the statistics from the whole dataset, you would specify the full dataset with Parameter NDF. [The dataset used to create the existing plot]

NSIGMA() = _REAL (Read)

Number of standard deviations about the mean at which the lines should be drawn. The null value or 0.0 causes a line to be drawn at the mean value.

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the lines.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The attributes Colour(Curves), Width(Curves), *etc.*, can be used to specify the style for the lines (Lines is recognised as a synonym for Curves). These values apply to all lines unless subsequent attributes override them. Attributes for individual clipping levels can be given by replacing Curves above by a string of the form `"Nsig<i>"` where `"<i>"` is an integer index into the list of clipping levels supplied for Parameter NSIGMA. Thus, `"Colour(Nsig1)"` will set the colour for the lines associated with the first clipping level, *etc*. The attribute settings can be restricted to one of the two lines by appending either a `"+"` or a `"-"` to the `"Nsig<i>"` string. Thus, `"Width(Nsig2-)"` sets the line width for the lower of the two lines associated with the second clipping level, and `"Width(Nsig2+)"` sets the width for the upper of the two lines. [current value]

Examples:

```
drawsig nsigma=3 style='style=1'
```

This draws solid horizontal lines on the last DATA picture on the current graphics device located at plus and minus 3 standard deviations about the mean. The statistics come from the data array used to draw the DATA picture.

```
drawsig phot 2.5
```

This draws horizontal plus and minus 2.5 standard-deviation lines about the mean for the data in the NDF called phot on the default graphics device.

```
drawsig phot 2.5 style='"colour(nsig1-)=red,colour(nsig1+)=green"'
```

As above, but the lower line is drawn in red and the upper line is drawn in green.

```
drawsig cluster [2,3] X Error
```

This draws vertical lines at plus and minus 2 and 3 standard deviations about the mean for the error data in the NDF called cluster on the default graphics device.

```
drawsig device=xwindows phot(20:119) 3 style="'colour=red,style=4'"
```

This draws red dotted horizontal lines on the xwindows device at ± 3 standard deviations using the 100 pixels in NDF phot(20:119).

Notes:

There must be an existing DATA picture stored within the graphics database for the chosen device. Lines will only be plotted within this picture.

Related Applications :

KAPPA: HISTOGRAM, LINPLOT, MLINPLOT, STATS.

Implementation Status:

- This routine correctly processes the DATA, VARIANCE, and QUALITY, components of the NDF.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. The statistics are calculated using double-precision floating point.
- Any number of NDF dimensions is supported.

ELPROF
Creates a radial or azimuthal profile of a two-dimensional image

Description:

This application will bin the input image into elliptical annuli, or into a 'fan' of adjacent sectors, centred on a specified position. The typical data values in each bin are found (see Parameter ESTIMATOR), and stored in a one-dimensional NDF which can be examined using LINPLOT, LOOK, *etc*. A two-dimensional mask image can optionally be produced indicating which bin each input pixel was placed in.

The area of the input image which is to be binned is the annulus enclosed between the two concentric ellipses defined by Parameters RATIO, ANGMAJ, RMIN, and RMAX. The binned area can be restricted to an azimuthal section of this annulus using Parameter ANGLIM. Input data outside the area selected by these parameters is ignored. The selected area can be binned in two ways, specified by Parameter RADIAL.

- If radial binning is selected (the default), then each bin is an elliptical annulus concentric with the ellipses bounding the binned area. The number of bins is specified by Parameter NBIN and the radial thickness of each bin is specified by WIDTH.
- If azimuthal binning is selected, then each bin is a sector (*i.e.* a wedge-shape), with its vertex given by Parameters XC and YC, and its opening angle given by Parameter WIDTH. The number of bins is specified by NBIN.

Usage:

```
elprof in out nbin xc yc
```

Parameters:

ANGLIM(2) = _REAL (Read)
Defines the wedge-shaped sector within which binning is to be performed. The first value should be the azimuthal angle of the clockwise boundary of the sector, and the second should be the azimuthal angle of the anti-clockwise boundary. The angles are measured in degrees from the *x*-axis, and rotation from the *x*-axis to the *y*-axis is positive. If only a single value is supplied, or if both values are equal, the sector starts at the given angle and extends for 360 degrees. `[0.0]`

ANGMAJ = _REAL (Read)
The angle between the *x*-axis and the major axis of the ellipse, in degrees. Rotation from the *x*-axis to the *y*-axis is positive. `[0.0]`

ESTIMATOR = LITERAL (Read)
The method to use for estimating the output pixel values. It can be either `"Mean"` or `"Weighted Mean"`. If the weighted mean option is selected but no variances are available in the input data, the unweighted mean will be used instead. `["Mean"]`

IN = NDF (Read)
The input NDF containing the two-dimensional image from which a profile is to be generated.

MASK = NDF (Write)
An output NDF of the same shape and size as the input NDF indicating the bin into which each input pixel was placed. For radial profiles, the bins are identified by a mask value equal to the radius (in pixels) measured on the major axis, at the centre of the annular bin. For azimuthal profiles, the bins are identified by a mask value equal to the angle from the *x*-axis to the centre of the sector-shaped bin (in degrees). If a null value is supplied, then no mask NDF is produced. `[!]`

MTITLE = LITERAL (Read)
A title for the mask NDF. If a null value is given, the title is propagated from the input NDF. This is only prompted for if MASK is given a non-null value. `["Mask created by KAPPA - Elprof"]`

NBIN = _INTEGER (Read)
The number of radial or azimuthal bins required.

OUT = NDF (Write)
The output one-dimensional NDF containing the required profile. For radial profiles, it has associated axis information describing the radius, in pixels, at the centre of each annular bin (the radius is measured on the major axis). For azimuthal profiles, the axis information describes the azimuthal angle, in degrees, at the centre of each sector-shaped bin. It will contain associated variance information if the input NDF has associated variance information.

RADIAL = _LOGICAL (Read)
Specifies the sort of profile required. If RADIAL is `TRUE`, then a radial profile is produced in which each bin is an elliptical annulus. Otherwise, an azimuthal profile is produced in which each bin is a wedge-shaped sector. `[TRUE]`

RATIO = _REAL (Read)
The ratio of the length of the minor axis of the ellipse to the length of the major axis. It must be in the range 0.0 to 1.0. `[1.0]`

RMAX = _REAL (Read)
The radius in pixels, measured on the major axis, at the outer edge of the elliptical annulus to be binned. If a null value (`!`) is supplied the value used is the distance from the ellipse centre (specified by XC and YC) to the furthest corner of the image. This will cause the entire image to fall within the outer edge of the binning area. `[!]`

RMIN = _REAL (Read)
The radius in pixels, measured on the major axis, at the inner edge of the elliptical region to be binned. `[0.0]`

TITLE = LITERAL (Read)
A title for the output profile NDF. If a null value is supplied the title is propagated from the input NDF. `["KAPPA - Elprof"]`

WIDTH = _REAL (Read)
The width of each bin. If a radial profile is being created (see Parameter RADIAL) this is the width of each annulus in pixels (measured on the major axis). If an azimuthal profile is being created, it is the opening angle of each sector, in degrees. If a null (`!`) value is supplied, the value used is chosen so that there are no gaps between adjacent bins. Smaller values will result in gaps appearing between adjacent bins. The supplied value must be small enough to ensure that adjacent bins do not overlap. The supplied value must be at least 1.0. `[!]`

XC = _REAL (Read)
The *x* pixel co-ordinate of the centre of the ellipse, and the vertex of the sectors.

YC = _REAL (Read)
The *y* pixel co-ordinate of the centre of the ellipse, and the vertex of the sectors.

Examples:

```
elprof galaxy galprof 20 113 210 angmaj=49 rmin=10 rmax=210 ratio=0.5
```

This example will create a one-dimensional NDF called galprof containing a radial profile of the two-dimensional NDF called galaxy. The profile will contain 20 bins and it will be centred on the pixel co-ordinates (113,210). Each bin will be an annulus of an ellipse with axis ratio of 0.5 and inclination of 49 degrees to the *x*-axis. The image will be binned between radii of 10 pixels, and 210 pixels (measured on the major axis), and there will be no gaps between adjacent bins (*i.e.* each bin will have a width on the major axis of about 10 pixels).

```
elprof galaxy galprof 10 113 210 radial=f anglim=20 rmin=50 rmax=60
```

This example also creates a one-dimensional NDF called galprof, this time containing an azimuthal profile of the two-dimensional NDF called `"galaxy"`, containing 10 bins. Each bin will be a wedge-shaped sector with vertex at pixel co-ordinates (113,210). The clockwise edge of the first bin will be at an angle of 20 degrees to the *x*-axis, and each bin will have a width (opening angle) of 36 degrees (so that 360 degrees are covered in total). Only the section of each sector bounded by radii of 50 and 60 pixels is included in the profile. In this case the default value of `1.0` is accepted for Parameter RATIO and so the bins will form a circular annulus of width 10 pixels.

Related Applications :

ESP: ELLFOU, ELLPRO, SECTOR.

Implementation Status:

- This routine correctly processes the DATA, VARIANCE, TITLE, UNITS, WCS (in radial mode only) and HISTORY components of the input NDF to the output profile NDF. WCS information is also propagated to the output mask NDF.
- WCS information is currently lost by this application.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single-precision floating point.

ERASE
Erases an HDS object

Description:

This routine erases a specified HDS object or container file. If the object is a structure, then all the structure's components (and sub-components, *etc.*) are also erased. If a slice or cell of an array is specified, then the entire array is erased.

Usage:

```
erase object
```

Parameters:

OBJECT = UNIV (Write)
The HDS object or container file to be erased.

OK = _LOGICAL (Read)
This parameter is used to seek confirmation before an object is erased. If a `TRUE` value is given, then the HDS object will be erased. If a `FALSE` value is given, then the object will not be erased and a message will be issued to this effect.

REPORT = _LOGICAL (Read)
This parameter controls what happens if the named OBJECT does not exist. If `TRUE`, an error is reported. Otherwise no error is reported. `[TRUE]`

Examples:

```
erase horse
```

This erases the HDS container file called `horse.sdf`.

```
erase fig123.axis
```

This erases the AXIS component of the HDS file called `fig123.sdf`. If AXIS is a structure, all its components are erased too.

```
erase fig123.axis(1).label
```

This erases the LABEL component within the first element of the AXIS structure of the HDS file called `fig123.sdf`.

```
erase $AGI_USER/agi_restar.agi_3200_1
```

This erases the AGIDEV_3200_1 structure of the HDS file called `$AGI_USER/agi_restar.sdf`.

Related Applications :

FIGARO: CREOBJ, DELOBJ, RENOBJ.

ERRCLIP
Removes pixels with large errors from an NDF

Description:

This application produces a copy of the input NDF in which pixels with errors greater than a specified limit are set invalid in both DATA and VARIANCE components. The error limit may be specified as the maximum acceptable standard deviation (or variance), or the minimum acceptable signal-to-noise ratio.

Usage:

```
errclip in out limit [mode]
```

Parameters:

IN = NDF (Read)
The input NDF. An error is reported if it contains no VARIANCE component.

LIMIT = _DOUBLE (Read)
Either the maximum acceptable standard deviation or variance value, or the minimum acceptable signal-to-noise ratio (depending on the value given for MODE). It must be positive.

MODE = LITERAL (Read)
Determines how the value supplied for LIMIT is to be interpreted: `"Sigma"` for a standard deviation, `"Variance"` for variance, or `"SNR"` for minimum signal-to-noise ratio. `["Sigma"]`

OUT = NDF (Write)
The output NDF.

Examples:

```
errclip m51 m51_good 2.0
```

The NDF m51_good is created holding a copy of m51 in which all pixels with standard deviation greater than 2 are set invalid.

```
errclip m51 m51_good 2.0 snr
```

The NDF m51_good is created holding a copy of m51 in which all pixels with a signal-to-noise ratio less than 2 are set invalid.

```
errclip m51 m51_good mode=v limit=100
```

The NDF m51_good is created holding a copy of m51 in which all pixels with a variance greater than 100 are set invalid.

Notes:

- The limit and the number of rejected pixels are reported.
- A pair of output data and variance values are set bad when either of the input data or variances values is bad.
- For MODE="SNR" the comparison is with respect to the absolute data value.

Related Applications :

KAPPA: FFCLEAN, PASTE, SEGMENT, SETMAGIC, THRESH.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. The output NDF has the same numeric type as the input NDF. However, all internal calculations are performed in double precision.

EXCLUDEBAD
Excludes bad rows or columns from a two-dimensional NDF

Description:

This application produces a copy of a two-dimensional NDF , but excludes any rows that contain too many bad aata values. Rows with higher pixel indices are shuffled down to fill the gaps left by the omission of bad rows. Thus if any bad rows are found, the output NDF will have fewer rows than the input NDF, but the order of the remaining rows will be unchanged. The number of good pixels required in a row for the row to be retained is specified by Parameter WLIM.

Bad columns may be omitted instead of bad rows (see Parameter ROWS).

Usage:

```
excludebad in out [rows] [wlim]
```

Parameters:

IN = NDF (Read)
The input two-dimensional NDF.

OUT = NDF (Write)
The output NDF.

ROWS = _LOGICAL (Read)
If `TRUE`, bad rows are excluded from the output NDF. If `FALSE`, bad columns are excluded. `[TRUE]`

WLIM = _REAL (Read)
The minimum fraction of the pixels which must be good in order for a row to be retained. A value of `1.0` results in rows being excluded if they contain one or more bad values. A value of `0.0` results in rows being excluded only if they contain no good values. `[0.0]`

Examples:

```
excludebad ifuframe goodonly false
```

Columns within NDF ifuframe that contain any good data are copied to NDF goodonly.

Notes:

- The lower pixel bounds of the output will be the same as those of the input, but the upper pixel bounds will be different if any bad rows or columns are excluded.

Related Applications :

KAPPA: CHPIX, FILLBAD, GLITCH, NOMAGIC, ZAPLIN; FIGARO: BCLEAN, CLEAN, ISEDIT, REMBAD, TIPPEX.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.

EXP10
Takes the base-10 exponential of an NDF data structure

Description:

This routine takes the base-10 exponential of each pixel of a NDF to produce a new NDF data structure.

This command is a synonym for `expon base=10D0`.

Usage:

```
exp10 in out
```

Parameters:

IN = NDF (Read)
Input NDF data structure.

OUT = NDF (Write)
Output NDF data structure being the exponential of the input NDF.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
exp10 a b
```

This takes exponentials to base ten of the pixels in the NDF called a, to make the NDF called b. NDF b inherits its title from a.

```
exp10 title="Abell 4321" out=b in=a
```

This takes exponentials to base ten of the pixels in the NDF called a, to make the NDF called b. NDF b has the title `"Abell 4321"`.

Related Applications :

KAPPA: LOG10, LOGAR, LOGE, EXPE, EXPON, POW; FIGARO: IALOG, ILOG, IPOWER.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

EXPE
Takes the natural exponential of an NDF data structure

Description:

This routine takes the natural exponential of each pixel of a NDF to produce a new NDF data structure.

This command is a synonym for `expon base=natural`.

Usage:

```
expe in out
```

Parameters:

IN = NDF (Read)
Input NDF data structure.

OUT = NDF (Write)
Output NDF data structure being the exponential of the input NDF.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
loge a b
```

This takes the natural exponential of the pixels in the NDF called a, to make the NDF called b. NDF b inherits its title from a.

```
loge title="Cas A" out=b in=a
```

This takes natural exponentials of the pixels in the NDF called a, to make the NDF called b. NDF b has the title `"Cas A"`.

Related Applications :

KAPPA: LOG10, LOGAR, LOGE, EXP10, EXPON, POW; FIGARO: IALOG, ILOG, IPOWER.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

EXPON
Takes the exponential (specified base) of an NDF data structure

Description:

This routine takes the exponential to a specified base of each pixel of a NDF to produce a new NDF data structure.

Usage:

```
expon in out base
```

Parameters:

BASE = LITERAL (Read)
The base of the exponential to be applied. A special value "Natural" gives natural (base-e) exponentiation.

IN = NDF (Read)
Input NDF data structure.

OUT = NDF (Write)
Output NDF data structure being the exponential of the input NDF.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

Examples:

```
expon a b 10
```

This takes exponentials to base ten of the pixels in the NDF called a, to make the NDF called b. NDF b inherits its title from a.

```
expon base=8 title="HD123456" out=b in=a
```

This takes expoentials to base eight of the pixels in the NDF called a, to make the NDF called b. NDF b has the title "HD123456".

Related Applications :

KAPPA: LOG10, LOGAR, LOGE, EXP10, EXPE, POW; FIGARO: IALOG, ILOG, IPOWER.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

FFCLEAN
Removes defects from a substantially flat one-, or two-, or three-dimensional NDF

Description:

This application cleans a one- or two-dimensional NDF by removing defects smaller than a specified size. In addition, three-dimensional NDFs can be cleaned by processing each row or plane within it using the one- or two-dimensional algorithm (see Parameter AXES).

The defects are flagged with the bad value. The defects are found by looking for pixels that deviate from the spectrum or image's smoothed version by more than an arbitrary number of standard deviations from the local mean, and that lie within a specified range of values. Therefore, the data array must be substantially flat. The data variances provide the local noise estimate for the threshold, but if these are not available a variance for the whole of the data array is derived from the mean squared deviations of the original and smoothed versions. The smoothed version of the data array is obtained by block averaging over a rectangular box. An iterative process progressively removes the outliers from the data array.

Usage:

```
ffclean in out clip box [thresh] [wlim]
```

Parameters:

AXES(2) = _INTEGER (Read)

The indices of up to two axes that span the rows or planes that are to be cleaned. If only one value is supplied, then the NDF is processed as a set of one-dimensional spectra parallel to the specified pixel axis. If two values are supplied, then the NDF is processed as a set of two-dimensional images spanned by the given axes. Thus, a two-dimensional NDF can be processed either as a single two-dimensional image or as a set of one-dimensional spectra. Likewise, a three-dimensional NDF can be processed either as a set of two-dimensional images or a set of one-dimensional spectra. By default, a two-dimensional NDF is processed as a single two-dimensional image, and a three-dimensional NDF is processed as a set of one-dimensional spectra (the spectral axis is chosen by examining the WCS component—pixel-axis 1 is used if the current WCS frame does not contain a spectral axis). []

BOX(2) = _INTEGER (Read)

The *x* and *y* sizes (in pixels) of the rectangular box to be applied to smooth the image. If only a single value is given, then it will be duplicated so that a square filter is used except where the image is one-dimensional for which the box size along the insignificant dimension is set to 1. The values given will be rounded up to positive odd integers if necessary.

CLIP( ) = _REAL (Read)

The number of standard deviations for the rejection threshold of each iteration. Pixels that deviate from their counterpart in the smoothed image by more than CLIP times the noise are made bad. The number of values given specifies the number of iterations.

Values should lie in the range 0.5–100. Up to one hundred values may be given. [3.0, 3.0, 3.0]

GENVAR = _LOGICAL (Read)

If TRUE, the noise level implied by the deviations from the local mean over the supplied box size are stored in the output VARIANCE component. This noise level has a constant value over the whole NDF (or over each section of the NDF if the NDF is being processed in sections—see Parameter AXES). This constant noise level is also displayed on the screen if the current message-reporting level is at least NORMAL. If GENVAR is FALSE, then the output variances will be copied from the input variances (if the input NDF has no variances, then the output NDF will not have any variances either).). [FALSE]

IN = NDF (Read)

The one- or two-dimensional NDF containing the input image to be cleaned.

OUT = NDF (Write)

The NDF to contain the cleaned image.

THRESH(2) = _DOUBLE (Read)

The range between which data values must lie if cleaning is to occur. Thus it is possible to clean the background without removing the cores of images by a judicious choice of these thresholds. If null, !, is given, then there is no limit on the data range. [!]

TITLE = LITERAL (Read)

The title of the output NDF. A null (!) value means using the title of the input NDF. [!]

WLIM = _REAL (Read)

If the input image contains bad pixels, then this parameter may be used to determine the number of good pixels which must be present within the smoothing box before a valid output pixel is generated. It can be used, for example, to prevent output pixels from being generated in regions where there are relatively few good pixels to contribute to the smoothed result.

By default, a null (!) value is used for WLIM, which causes the pattern of bad pixels to be propagated from the input image to the output image unchanged. In this case, smoothed output values are only calculated for those pixels which are not bad in the input image.

If a numerical value is given for WLIM, then it specifies the minimum fraction of good pixels which must be present in the smoothing box in order to generate a good output pixel. If this specified minimum fraction of good input pixels is not present, then a bad output pixel will result, otherwise a smoothed output value will be calculated. The value of this parameter should lie between 0.0 and 1.0 (the actual number used will be rounded up if necessary to correspond to at least one pixel). [!]

Results Parameters:

SIGMA = _DOUBLE (Write)
The estimation of the RMS noise per pixel of the output image.

Examples:

```
ffclean dirty clean \
```

The NDF called dirty is filtered such that pixels that deviate by more than three standard deviations from the smoothed version of dirty are rejected. Three iterations are performed. Each pixel in the smoothed image is the average of the neighbouring nine pixels. The filtered NDF is called clean.

```
ffclean out=clean in=dirty thresh=[-100,200]
```

As above except only those pixels whose values lie between −100 and 200 can be cleaned.

```
ffclean poxy dazed [2.5,2.8] [5,5]
```

The two-dimensional NDF called poxy is filtered such that pixels that deviate by more than 2.5 then 2.8 standard deviations from the smoothed version of poxy are rejected. The smoothing is an average of a 5-by-5-pixel neighbourhood. The filtered NDF is called dazed.

Notes:

- There are different facts reported, their verbosity depending on the current message-reporting level set by environment variable MSG_FILTER. When the filtering level is at least as verbose as NORMAL, the application will report the intermediate results after each iteration during processing. In addition, it will report the section of the input NDF currently being processed (but only if the NDF is being processed in sections—see Parameter AXES).

Related Applications :

KAPPA: CHPIX, FILLBAD, GLITCH, MEDIAN, MSTATS, ZAPLIN; FIGARO: BCLEAN, COSREJ, CLEAN, ISEDIT, MEDFILT, MEDSKY, TIPPEX.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single- or double-precision floating point as appropriate.

FILLBAD
Removes regions of bad values from an NDF

Description:

This application replaces bad values in an NDF with a smooth function which matches the surrounding data. It can fill arbitrarily shaped regions of bad values within n-dimensional arrays.

It forms a smooth replacement function for the regions of bad values by forming successive approximations to a solution of Laplace's equation, with the surrounding valid data providing the boundary conditions.

Usage:

```
fillbad in out [niter] [size]
```

Parameters:

BLOCK = _INTEGER (Read)

The maximum number of pixels along either dimension when the array is divided into blocks for processing. It is ignored unless MEMORY=TRUE. This must be at least 256. [512]

IN = NDF (Read)

The NDF containing the input image with bad values.

MEMORY = _LOGICAL (Read)

If this is FALSE, the whole array is processed at the same time. If it is TRUE, the array is divided into chunks whose maximum dimension along an axis is given by Parameter BLOCK. [FALSE]

NITER = _INTEGER (Read)

The number of iterations of the relaxation algorithm. This value cannot be fewer than two, since this is the minimum number required to ensure that all bad values are assigned a replacement value. The more iterations used, the finer the detail in the replacement function and the closer it will match the surrounding good data. [2]

OUT = NDF (Write)

The NDF to contain the image free of bad values.

SIZE( ) = _REAL (Read)

The initial scale lengths in pixels to be used in the first iteration, along each axis. If fewer values are supplied than pixel axes in the NDF, the last value given is repeated for the remaining axes. The size 0 means no fitting across a dimension. For instance, [0,0,5] would be appropriate if the spectra along the third dimension of a cube are independent, and the replacement values are to be derived only within each spectrum.

For maximum efficiency, a scale length should normally have a value about half the 'size' of the largest invalid region to be replaced. (See "Notes" section for more details.) [5.0]

TITLE = LITERAL (Read)
The title of the output NDF. A null (`!`) value means using the title of the input NDF. `[!]`

VARIANCE = _LOGICAL (Read)
If VARIANCE is `TRUE`, variance information is to be propagated; any bad values therein are filled. Also the variance is used to weight the calculation of the replacement data values. If VARIANCE is `FALSE`, there will be no variance processing thus requiring two less arrays in memory. This parameter is only accessed if the input NDF contains a VARIANCE component. `[TRUE]`

Results Parameters:

CNGMAX = _DOUBLE (Write)
The maximum absolute change in output values which occurred in the final iteration.

CNGRMS = _DOUBLE (Write)
The root-mean-squared change in output values which occurred in the last iteration.

Examples:

```
fillbad aa bb
```

The NDF called aa has its bad pixels replaced by good values derived from the surrounding good pixel values using two iterations of a relaxation algorithm. The initial scale length is 5 pixels. The resultant NDF is called bb.

```
fillbad aa bb 6 20 title="Cleaned image"
```

As above except the initial scale length is 20 pixels, 5 iterations will be performed, and the output title is `"Cleaned image"` instead of the title of NDF aa.

```
fillbad aa bb memory novariance
```

As in the first example except that processing is performed with blocks up to 512 by 512 pixels to reduce the memory requirements, and no variance information will be used or propagated.

```
fillbad in=speccube out=speccube_fill size=[0,0,128] iter=5
```

Suppose NDF speccube is a spectral imaging cube with the spectral axis third. This example replaces the bad pixels by valid values derived from the surrounding good pixel values within each spectrm, using an initial scale length of 128 channels, iterating five times. The filled NDF is called speccube_fill.

```
fillbad in=speccube out=speccube_fill size=[5,5,128] iter=5
```

As the previous example, but now the relaxation occurs along the spatial axes too, initially with a scale length of five pixels.

Notes:

- The algorithm is based on the relaxation method of repeatedly replacing each bad pixel with the mean of its two nearest neighbours along each pixel axis. Such a method converges to the required solution, but information about the good regions only propagates at a rate of about one pixel per iteration into the bad regions, resulting in slow convergence if large areas are to be filled.

  This application speeds convergence to an acceptable function by forming the replacement mean from all the pixels in the same axis (such as row or a column), using a weight which decreases exponentially with distance and goes to zero after the first good pixel is encountered in any direction. If there is variance information, this is included in the weighting so as to give more weight to surrounding values with lower variance. The scale length of the exponential weight is initially set large, to allow rapid propagation of an approximate 'smooth' solution into the bad regions—an initially acceptable solution is thus rapidly obtained (often in the first one or two iterations). The scale length is subsequently reduced by a factor of 2 whenever the maximum absolute change occurring in an iteration has decreased by a factor of 4 since the current scale length was first used. In this way, later iterations introduce progressively finer detail into the solution. Since this fine detail occurs predominantly close to the 'crinkly' edges of the bad regions, the slower propagation of the solution in the later iterations is then less important.

  When there is variance processing the output variance is reassigned if either the input variance or data value was bad. Where the input value is good but its associated variance is bad, the calculation proceeds as if the data value were bad, except that only the variance is substituted in the output. The new variance is approximated as twice the inverse of the sum of the weights.
- The price of the above efficiency means that considerable workspace is required, typically two or three times the size of the input image, but even larger for the one and two-byte integer types. If memory is at a premium, there is an option to process in blocks (*cf.* Parameter MEMORY). However, this may not give as good results as processing the array in full, especially when the bad-pixel regions span blocks.
- The value of the Parameter SIZE is not critical and the default value will normally prove effective. It primarily affects the efficiency of the algorithm on various size scales. If the smoothing scale is set to a large value, large scale variations in the replacement function are rapidly found, while smaller scale variations may require many iterations. Conversely, a small value will rapidly produce the small scale variations but not the larger scale ones. The aim is to select an initial value SIZE such that during the course of a few iterations, the range of size scales in the replacement function are all used. In practice this means that the value of SIZE should be about half the size of the largest scale variations expected. Unless the valid pixels are very sparse, this is usually determined by the 'size' of the largest invalid region to be replaced.
- An error results if the input NDF has no bad values to replace.
- The progress of the iterations is reported at the normal reporting level. The format of the output is slightly different if the scale lengths vary with pixel axis; an extra axis

column is included.

Timing :

The time taken increases in proportion to the value of NITER. Adjusting the SIZE parameter to correspond to the largest regions of bad values will reduce the processing time. See the "Notes" section.

Related Applications :

KAPPA: CHPIX, GLITCH, MEDIAN, ZAPLIN; FIGARO: BCLEAN, COSREJ, CLEAN, ISEDIT, MEDFILT, MEDSKY, REMBAD, TIPPEX.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported. The output bad-pixel flag is set to indicate no bad values in the data and variance arrays.
- All non-complex numeric data types can be handled. Arithmetic is performed using single- or double-precision floating point as appropriate.

FITSDIN
Reads a FITS disc file composed of simple, group or table objects

Description:

This application reads selected disc-FITS files. The files may be Basic (simple) FITS, and/or have TABLE extensions (Harten *et al.* 1988).

The programme reads a simple or a random-groups-format FITS file (Wells *et al.* 1981; Greisen & Harten 1981), and writes the data into an NDF , and the headers into the NDF's FITS extension. Table-format files (Grosbøl *et al.* 1988) are read, and the application creates two files: a text formatted table/catalogue and a FACTS description file (as used by SCAR) based upon the FITS header cards. Composite FITS files can be processed. You may specify a list of files, including wildcards. A record of the FITS headers, and group parameters (for a group-format file) can be stored in a text file.

There is an option to run in automatic mode, where the names of output NDF data structures are generated automatically, and you can decide whether or not format conversion is to be applied to all files (rather than being prompted for each). This is very useful if there is a large number of files to be processed. Even if you want unique file names, format-conversion prompting may be switched off globally.

Usage:

```
fitsdin files out [auto] fmtcnv [logfile] dscftable=?  table=?
```

Parameters:

AUTO = _LOGICAL (Read)

It is TRUE if automatic mode is required, where the name of each output NDF structure or table file is to be generated by the application, and therefore not prompted; and a global format-conversion switch may be set. In manual mode the FITS header is reported, but not in automatic.

In automatic mode the application generates a filename beginning with the input filename, less any extension. For example, if the input file was saturn.fits the filename of the output NDF would be saturn.sdf, and an output table would be saturn.dat with a description file dscfsaturn.dat. If there are sub-files (more than one FITS object in the file) a suffix _<subfile> is appended. So if saturn.fits comprised a simple file followed by a table, the table would be called SATURN_2.DAT and the description file DSCFSATURN_2.DAT. For group format a suffix G<groupnumber> is appended. Thus if saturn.fits is a group format file, the first NDF created would be called saturn.sdf, the second would be saturnG2.sdf. [FALSE]

DSCFTABLE = FILENAME (Read)

Name of the text file to contain the FACTS descriptors, which defines the table's format for SCAR. Since SCAR is now deprecated, this parameter has little use, except perhaps to give a summary of the format of the file specified by Parameter TABLE. A null value (!) means that no description file will be created, so this is now the recommended usage. If your FITS file comprises just tables, you should consider other

tools such as the CURSA package, which has facilities for examining and processing ASCII and binary tables in FITS files.

A suggested filename for the description file is reported immediately prior to prompting in manual mode. It is the name of the catalogue, as written in the FITS header, with a `"dscf"` prefix.

ENCODINGS = LITERAL (Read)

Determines which FITS keywords should be used to define the world co-ordinate systems to be stored in the NDF's WCS component. The allowed values (case-insensitive) are as follows.

- `"FITS-IRAF"` — This uses keywords CRVAL*i* CRPIX*i*, CD*i_j*, and is the system commonly used by IRAF. It is described in the document "World Coordinate Systems Representations Within the FITS Format" by R.J. Hanisch and D.G. Wells, 1988, available by ftp from fits.cv.nrao.edu `/fits/documents/wcs/wcs88.ps.Z`.
- `"FITS-WCS"` — This is the FITS standard WCS encoding scheme described in the paper "Representation of celestial coordinates in FITS" (`http://www.cv.nrao.edu/fits/documents/wcs/wcs.html`).

  It is very similar to FITS-IRAF but supports a wider range of projections and co-ordinate systems. Once the standard has been agreed, this encoding should be understood by any FITS-WCS compliant software and it is likely to be adopted widely for FITS data in future.
- `"FITS-PC"` — This uses keywords CRVAL*i*, CDELT*i*, CRPIX*i*, PC*iiijjj*, *etc*, as in a previous (now superseded) draft of the above FITS world co-ordinate system paper by E.W. Greisen and M. Calabretta.
- `"FITS-AIPS"` — This uses conventions described in the document "Non-linear Coordinate Systems in AIPS" by Eric W. Greisen (revised 9th September, 1994), available by ftp from fits.cv.nrao.edu `/fits/documents/wcs/aips27.ps.Z`. It is currently employed by the AIPS data analysis facility, so its use will facilitate data exchange with AIPS. This encoding uses CROTA*i* and CDELT*i* keywords to describe axis rotation and scaling.
- `"DSS"` — This is the system used by the Digital Sky Survey, and uses keywords AMDX*n*, AMDY*n*, PLTRAH, *etc*.
- `"Native"` — This is the native system used by the AST library (see SUN/210), and provides a loss-free method for transferring WCS information between AST-based application. It allows more complicated WCS information to be stored and retrieved than any of the other encodings.

A comma-separated list of up to six values may be supplied, in which case the value actually used is in the first in the list for which corresponding keywords can be found in the FITS header.

A FITS header may contain keywords from more than one of these encodings, in which case it is possible for the encodings to be inconsistent with each other. This may happen for instance if an application modifies the keyword associated with one encoding but fails to make equivalent modifications to the others. If a null parameter value (`!`) is supplied for ENCODINGS, then an attempt is made to determine the most reliable encoding to use as follows. 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 (if present). [!]

FILES() = LITERAL (Read)
A list of (optionally wild-carded) file specifications which identify the disc-FITS files to be processed. Up to ten values may be given, but only a single specification such as "*.fits" is normally required. Be careful not to include non-FITS files in this list.

FMTCNV = _LOGICAL (Read)
This specifies whether or not format conversion will occur. If FALSE, the HDS type of the data array in the NDF will be the equivalent of the FITS data format in the file (*e.g.* BITPIX=16 creates a _WORD array). If TRUE, the data array in the current file, or all files in automatic mode, will be converted from the FITS data type in the FITS file to _REAL in the NDF. The conversion applies the values of the FITS keywords BSCALE and BZERO to the FITS-file data to generate the 'true' data values. If BSCALE and BZERO are not given in the FITS header, they are taken to be 1.0 and 0.0 respectively. The suggested default is TRUE.

GLOCON = _LOGICAL (Read)
If FALSE a format-conversion query occurs for each FITS file. If TRUE, the value of Parameter FMTCNV is obtained before any file numbers and will apply to all data arrays. It is ignored in automatic mode—in effect it becomes TRUE. [FALSE]

LOGFILE = FILENAME (Read)
The file name of the text log of the FITS header cards. For group-format data the group parameters are evaluated and appended to the full header. The log includes the names of the output files used to store the data array or table. A null value (!) means that no log file is produced. [!]

OUT = NDF (Write)
Output NDF structure holding the full contents of the FITS file. If the null value (!) is given no NDF will be created. This offers an opportunity to review the descriptors before deciding whether or not the data are to be extracted.

TABLE = FILENAME (Read)
Name of the text file to contain the table itself, read from the file. In manual mode, the suggested default filename is the name of description file less the "dscf" prefix, or if there is no description file or if the description file does not have the "dscf" prefix, the suggested name reverts to the catalogue name in the FITS header.

Examples:

```
fitsdin files=*.fit auto nofmtcnv
```

This reads all the files with extension "fit" in the default directory. If the files were sao.fit and moimp.fit and each contained just an image array, the output NDFs will be sao and moimp respectively. The data will not have format conversion.

```
fitsdin files=ccd.ifits fmtcnv logfile=jkt.log
```

This reads the file ccd.ifits and processes all the FITS objects within it. Integer data arrays are converted to real using the scale and zero found in the FITS header. A record of the headers and the names of the output files are written to the text file jkt.log.

```
fitsdin files=[*.*fits,*.mt] glocon fmtcnv
```

This reads the files `*.*fits` and `*.mt` and processes all the FITS objects within them. Integer data arrays are converted to real using the scale and zero found in the FITS header. Any IEEE-format data will not be converted although the global conversion switch is on.

References :

Wells, D.C., Greisen, E.W. & Harten, R.H. 1981, *Astron. Astrophys. Suppl. Ser.* **44**, 363.
Greisen, E.W. & Harten, R.H. 1981, *Astron. Astrophys. Suppl. Ser.* **44**, 371.
Grosbøl, P., Harten, R.H., Greisen, E.W & Wells, D.C. 1988 *Astron. Astrophys. Suppl. Ser.* **73**, 359.
Harten, R.H., Grosbøl, P., Greisen, E.W & Wells, D.C. 1988 *Astron. Astrophys. Suppl. Ser.* **73**, 365.

Related Applications :

KAPPA: FITSHEAD, FITSIMP, FITSIN, FITSLIST; CONVERT: FITS2NDF; CURSA; FIGARO: RDFITS.

Implementation Status:

- The application processes FITS files blocked at other than an integer multiple of 2880 bytes up to a maximum of 28800, provided it is a multiple of the number of bytes per data value.
- For simple or group format FITS:
  - IEEE floating point is supported.
  - If BUNIT is present its value will appear as the NDF's UNITS component.
  - If OBJECT is present its value will appear as the NDF's TITLE component.
  - If the BLANK item is present in the header, undefined pixels are converted from the BLANK value to Starlink-standard bad value during data conversion.
  - An AXIS component will be stored in the NDF if the CRVAL*n* keyword is present. (*n* is the number of the dimension.) If the CRPIX*n* keyword is absent it defaults to 1, and likewise for the CDELT*n* keyword. The value of CTYPE*n* is made the label of the axis structure.
- For groups format, a new NDF is created for each data array. The name of the NDF of the second and subsequent data arrays is generated by the application as the `<filename>G<number>`, where `<filename>` is the name of the first NDF, you supply or generated automatically, and `<number>` is the number of the group.
  Each group NDF contains the full header in the FITS extension, appended by the set of group parameters. The group parameters are evaluated using their scales and offsets, and made to look like FITS cards, whose keywords are derived from the values of PTYPE*m* in the main header. (*m* is the number of the group parameter.) The same format is used in the log file.
- If there is no data array in the FITS file, *i.e.* the FITS file comprises header cards only, then a dummy vector data array of dimension two is created to make the output a valid NDF. This data array is undefined.

FITSEDIT
Edits the FITS extension of an NDF

Description:

This procedure allows you to use your favourite editor to modify the FITS headers stored in an NDF's FITS extension. There is limited validation of the FITS headers after editing. A FITS extension is created if the NDF does not already have one.

Usage:

```
fitsedit ndf
```

Parameters:

NDF = NDF (Read)

The name of the NDF whose FITS extension is to be edited.

Examples:

```
fitsedit m51b
```

This allows editing of the FITS headers in the NDF called m51b.

Notes:

- This uses the environmental variable, EDITOR, to select the editor. If this variable is undefined vi is assumed.
- The script lists the headers to a temporary file; allows text editing; and then replaces the former FITS extension with the modified version, performing some validation at this stage.

Related Applications :

KAPPA: FITSMOD, FITSEXP, FITSHEAD, FITSIMP, FITSLIST; FIGARO: FITSKEYS.

FITSEXIST
Inquires whether or not a keyword exists in a FITS extension

Description:

This application reports whether or not a keyword exists in the FITS extension of an NDF file.

Usage:

```
fitsexist ndf keyword
```

Parameters:

KEYWORD = LITERAL (Read)

The name of the keyword whose existence in the FITS extension is to be tested. 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 only alphanumeric characters, hyphen, and underscore. Any lowercase letters are converted to uppercase and blanks are removed before comparison with the existing keywords.

KEYWORD may have an occurrence specified in brackets [] following the name. This enables testing for the existence of multiple occurrences. 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 between the brackets other than a positive integer is interpreted as the first occurrence.

The suggested value is the current value.

NDF = NDF (Read)

The NDF to be tested for the presence of the FITS keyword.

Results Parameters:

EXISTS = _LOGICAL (Write)

The result of the final existence test.

Examples:

```
fitsexist abc bscale
```

This reports TRUE or FALSE depending on whether or not the FITS keyword BSCALE exists in the FITS extension of the NDF called abc.

```
fitsexist ndf=abc keyword=date[2]
```

This reports TRUE or FALSE depending on whether or not the FITS there are at least two occurrences of the keyword DATE.

Related Applications :

KAPPA: FITSEDIT, FITSHEAD, FITSLIST, FITSMOD, FITSVAL.

FITSEXP
Exports NDF-extension information into an NDF FITS extension

Description:

This application places the values of components of an NDF extension into the FITS extension within the same NDF. This operation is needed if auxiliary data are to appear in the header of a FITS file converted from the NDF. The list of extension components whose values are to be copied, their corresponding FITS keyword names, optional FITS inline comments, and the location of the new FITS header are specified in a *keyword translation table* held in a separate text file.

Usage:

```
fitsexp ndf table
```

Parameters:

NDF = NDF (Read and Write)

The NDF in which the extension data are to be exported to the FITS extension.

TABLE = FILE (Read)

The text file containing the keyword translation table. The format of this file is described under "Table Format".

Examples:

```
fitsexp datafile fitstable.txt
```

This writes new FITS-extension elements for the NDF called datafile, creating the FITS extension if it does not exist. The selection of auxiliary components to export to the FITS extension, their keyword names, locations, and comments are under the control of a keyword translation table held in the file `fitstable.txt`.

Notes:

- Requests to assign values to the following reserved keywords in the FITS extension are ignored: SIMPLE, BITPIX, NAXIS, NAXISn, EXTEND, PCOUNT, GCOUNT, XTENSION, BLOCKED, and END.
- Only scalar or one-element vector components may be transferred to the FITS extension.
- The data type of the component selects the type of the FITS value.
- If the destination keyword exists, the existing value and comment are replaced with the new values.
- If an error is found within a line, processing continues to the next line and the error reported.
- To be sure that the resultant FITS extension is what you desired, you should inspect it using the command FITSLIST before exporting the data. If there is something wrong, you may find it convenient to use command fitsedit to make minor corrections.

Timing :

Approximately proportional to the number of FITS keywords to be translated.

Table Format :

The keyword translation table should be held in a text file, with one extension component specified per line. Each line should contain two or three fields, separated by spaces and/or tabs, as follows.

- Field 1: The name of the input extension component whose value is to be copied to the FITS extension. For example, `CCDPACK.FILTER` would copy the value of the component called FILTER in the extension called CCDPACK; and `IRAS90.ASTROMETRY.EQUINOX` would copy the value of component EQUINOX in the structure ASTROMETRY in the extension IRAS90. The extension may not be FITS.
- Field 2: The name of the FITS keyword to which the value is to be copied. Hierarchical keywords are not permissible. The keyword name may be followed by a further keyword name in parentheses (and no spaces). This second keyword defines the card before which the new keyword is to be placed. If this second keyword is not present in the FITS extension or is not supplied, the new header card is placed at the end of the existing cards, but immediately before any END card. For example, `EQUINOX(EPOCH)` would write the keyword EQUINOX immediately before the existing card with keyword EPOCH. FITS keywords are limited to 8 characters and may only comprise uppercase alphabetic characters, digits, underscore, and hyphen. While it is possible to have multiple occurrences of the same keyword in a FITS header, it is regarded as bad practice. For this and efficiency reasons, this programme only looks for the first appearance of a keyword when substituting the values, and so only the last value inserted appears in the final FITS extension. (See `"Implementation Status"`.)
- Field 3: The comment to appear in the FITS header card for the chosen keyword. This field is optional. As much of the comment will appear in the header card as the value permits up to a maximum of 47 characters.

Comments may appear at any point in the table and should begin with an exclamation mark. The remainder of the line will then be ignored.

References :

"A User's Guide for the Flexible Image Transport System (FITS)", NASA/Science Office of Science and Technology (1994).

Related Applications :

KAPPA: FITSEDIT, FITSHEAD, FITSLIST, FITSMOD; CONVERT: NDF2FITS.

Implementation Status:

- The replacements are made in blocks of 32 to reduce the number of time-consuming shuffles of the FITS extension. Thus it is possible to locate a new keyword before another keyword, provided the latter keyword appears in an earlier block, though reliance on this feature is discouraged; instead run the application twice.

- For each block the application inserts new cards or relocates old ones, marking each with different tokens, and then sorts the FITS extension into the requested order, removing the relocated cards. It then inserts the new values. If there are multiple occurrences of a keyword, this process can leave behind cards having the token value `'{undefined}'`.

FITSHEAD
Lists the headers of FITS files

Description:

This procedure lists to standard output the headers of the primary header and data unit, and any extensions present that are contained within a set of input FITS files, or a range of specified files on a tape.

Usage:

```
fitshead file [block] [start] [finish]
```

Parameters:

BLOCK = _INTEGER (Read)

The FITS blocking factor of the tape to list. This is the tape blocksize in bytes divided by the FITS record length of 2880 bytes. BLOCK must be a positive integer, between 1 and 12, otherwise you will be prompted for a new value. Should the first argument not be a tape device, this argument will be treated as a file name. [1]

FILE = FILENAME (Read)

A space-separated list of FITS files whose headers are to be listed, or the name of a single no-rewind tape device. The list of files can include wildcard characters.

FINISH = _INTEGER (Read)

The last file on the tape to list. This defaults to the end of the tape. It must be a positive integer and at least equal to the value of start, otherwise you will be prompted for a new value. Should the first argument not be a tape device, this argument will be treated as a file name. []

START = _INTEGER (Read)

The first file on the tape to list. This defaults to 1, *i.e.* the start of the tape. It must be a positive integer, otherwise you will be prompted for a new value. Should the first argument not be a tape device, this argument will be treated as a file name. [1]

Examples:

```
fitshead /dev/nrmt1
```

This lists the FITS headers for all the files of the tape mounted on device /dev/nrmt1. The tape block size is 2880 bytes.

```
fitshead /dev/nrmt1 10 > tape.lis
```

This lists to file tape.lis the FITS headers for all the files of the tape mounted on device /dev/nrmt1. The tape blocking factor is 10, the tape's blocksize is 28800 bytes.

```
fitshead /dev/rmt/0n 2 3 5 >> tape.lis
```

This appends the FITS headers for files 3 to 5 of the tape mounted on device

/dev/rmt/0n to the file tape.lis. The tape blocking factor is 2, *i.e.* the tape's blocksize is 5760 bytes.

```
fitshead /dev/nrst2 prompt
```

This lists the FITS headers for files of the tape mounted on device /dev/nrst2. The command prompts you for the file limits and tape blocking factor.

```
fitshead ~/fits/*.fit ~/data/p?.fi* | lpr
```

This prints the FITS headers in the files ~/fits/*.fit and ~/data/p?.fi*.

Notes:

- Prompting is directed to the standard error, so that the listings may be redirected to a file.
- If the blocking factor is unknown it is possible to obtain only a part of the headers and some of the FITS data. Unless the FITS file is small, it is usually safe to set Parameter BLOCK higher than its true value.

Related Applications :

KAPPA: FITSEDIT, FITSEXP, FITSIMP, FITSLIST; FIGARO: FITSKEYS.

FITSIMP
Imports FITS information into an NDF extension

Description:

This application extracts the values of FITS keywords from a FITS extension in an NDF and uses them to construct another NDF extension. The list of new extension components required, their data types and the names of the FITS keywords from which to derive their values are specified in a *keyword translation table* held in a separate text file.

Usage:

```
fitsimp ndf table xname xtype
```

Parameters:

NDF = NDF (Read and Write)
The NDF in which the new extension is to be created.

TABLE = FILENAME (Read)
The text file containing the keyword translation table. The format of this file is described under "Table Format".

XNAME = LITERAL (Read)
The name of the NDF extension which is to receive the values read from the FITS extension . If this extension does not already exist, then it will be created. Otherwise, it should be a scalar structure extension within which new components may be created (existing components of the same name will be over-written). Extension names may contain up to 15 alpha-numeric characters, beginning with an alphabetic character.

XTYPE = LITERAL (Read)
The HDS data type of the output extension. This value will only be required if the extension does not initially exist and must be created. New extensions will be created as scalar structures.

Examples:

```
fitsimp datafile fitstable ccdinfo ccd_ext
```

Creates a new extension called CCDINFO (with a data type of CCD_EXT) in the NDF structure called datafile. Keyword values are read from the NDF's FITS extension and written into the new extension as separate components under control of a keyword translation table held in the file `fitstable`.

```
fitsimp ndf=n1429 table=std_table xname=std_extn
```

FITS keyword values are read from the FITS extension in the NDF structure n1429 and written into the pre-existing extension STD_EXTN under control of the translation table `std_table`. Components which already exist within the extension may be over-written by this process.

Timing :

Approximately proportional to the number of FITS keywords to be translated.

Table Format :

The keyword translation table should be held in a text file, with one extension component specified per line. Each line should contain 3 fields, separated by spaces and/or tabs, as follows.

- Field 1: The name of the component in the output extension for which a value is to be obtained.
- Field 2: The data type of the output component, to which the keyword value will be converted (one of `_INTEGER`, `_REAL`, `_DOUBLE`, `_LOGICAL` or `_CHAR`).
- Field 3: The name of the FITS keyword from which the value is to be obtained. Hierarchical keywords are permissible; the format is concatenated keywords joined with full stops and no spaces, *e.g.* `HIERARCH.ESO.NTT.HUMIDITY`, `ING.DETHEAD`.

Comments may appear at any point in the table and should begin with an exclamation mark. The remainder of the line will then be ignored.

Related Applications :

KAPPA: FITSHEAD, FITSLIST, FITSDIN, FITSIN; CONVERT: FITS2NDF; FIGARO: RDFITS.

FITSIN
Reads a FITS tape composed of simple, group or table files

Description:

This application reads selected files from a FITS tape. The files may be Basic (simple) FITS, and/or have TABLE extensions (Harten *et al.* 1988).

The programme reads a simple or a random-groups-format FITS file (Wells *et al.* 1981; Greisen & Harten 1981), and writes the data into an NDF , and the headers into the NDF's FITS extension. Table-format files (Grosbøl *et al.* 1988) are read, and the application creates two files: a text formatted table/catalogue and a FACTS description file (as used by SCAR) based upon the FITS header cards. Composite FITS files can be processed. You may specify a list of files, including wildcards. A record of the FITS headers, and group parameters (for a group-format file) can be stored in a text file.

There is an option to run in automatic mode, where the names of output NDF data structures are generated automatically, and you can decide whether or not format conversion is to be applied to all files (rather than being prompted for each). This is very useful if there is a large number of files to be processed. Even if you want unique file names, format-conversion prompting may be switched off globally.

Usage:

```
fitsin mt files out [auto] fmtcnv [logfile] more=?  dscftable=?  table=?
```

Parameters:

AUTO = _LOGICAL (Read)

It is TRUE if automatic mode is required, where the name of each output NDF structure or table file is to be generated by the application, and therefore not prompted; and a global format-conversion switch may be set. In manual mode the FITS header is reported, but not in automatic.

For simple or group format FITS objects in automatic mode the application generates a filename beginning with a defined prefix followed by the number of the file on tape. For example, if the prefix was "XRAY" and the 25th file of the tape was being processed, the filename of the NDF would be XRAY25.

For table-format FITS objects in the automatic mode the application generates a filename beginning with a defined prefix followed by the number of the file on tape. For example, if the prefix was "cat" and the 9th file of the tape was being processed, the filename of the table and its associated FACTS description file would be cat9.dat and dscfcat9.dat respectively. [FALSE]

DSCFTABLE = FILENAME (Read)

Name of the text file to contain the FACTS descriptors, which defines the table's format for SCAR. Since SCAR is now deprecated, this parameter has little use, except perhaps to give a summary of the format of the file specified by Parameter TABLE. A null value (!) means that no description file will be created, so this is now the recommended usage. If your FITS file comprises just tables, you should consider other

tools such as the CURSA package, which has facilities for examining and processing ASCII and binary tables in FITS files.

A suggested filename for the description file is reported immediately prior to prompting in manual mode. It is the name of the catalogue, as written in the FITS header, with a `"dscf"` prefix.

ENCODINGS = LITERAL (Read)

Determines which FITS keywords should be used to define the world co-ordinate systems to be stored in the NDF's WCS component. The allowed values (case-insensitive) are as follows.

- `"FITS-IRAF"` — This uses keywords CRVAL*i* CRPIX*i*, CD*i_j*, and is the system commonly used by IRAF. It is described in the document "World Coordinate Systems Representations Within the FITS Format" by R.J. Hanisch and D.G. Wells, 1988, available by ftp from fits.cv.nrao.edu `/fits/documents/wcs/wcs88.ps.Z`.
- `"FITS-WCS"` — This is the FITS standard WCS encoding scheme described in the paper "Representation of celestial coordinates in FITS" (`http://www.cv.nrao.edu/fits/documents/wcs/wcs.html`).

  It is very similar to FITS-IRAF but supports a wider range of projections and co-ordinate systems.
- `"FITS-PC"` — This uses keywords CRVAL*i*, CDELT*i*, CRPIX*i*, PC*iiijjj*, *etc*, as in a previous (now superceded) draft of the above FITS world co-ordinate system paper by E.W. Greisen and M. Calabretta.
- `"FITS-AIPS"` — This uses conventions described in the document "Non-linear Coordinate Systems in AIPS" by Eric W. Greisen (revised 9th September, 1994), available by ftp from fits.cv.nrao.edu `/fits/documents/wcs/aips27.ps.Z`. It is currently employed by the AIPS data analysis facility, so its use will facilitate data exchange with AIPS. This encoding uses CROTA*i* and CDELT*i* keywords to describe axis rotation and scaling.
- `"DSS"` — This is the system used by the Digital Sky Survey, and uses keywords AMDX*n*, AMDY*n*, PLTRAH, *etc*.
- `"Native"` — This is the native system used by the AST library (see SUN/210), and provides a loss-free method for transferring WCS information between AST-based application. It allows more complicated WCS information to be stored and retrieved than any of the other encodings.

A comma-separated list of up to six values may be supplied, in which case the value actually used is in the first in the list for which corresponding keywords can be found in the FITS header.

A FITS header may contain keywords from more than one of these encodings, in which case it is possible for the encodings to be inconsistent with each other. This may happen for instance if an application modifies the keyword associated with one encoding but fails to make equivalent modifications to the others. If a null parameter value (`!`) is supplied for ENCODINGS, then an attempt is made to determine the most reliable encoding to use as follows. 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 (if present). `[!]`

FILES() = _CHAR (Read)
The list of the file numbers to be processed. Files are numbered consecutively from 1 from the start of the tape. Single files or a set of adjacent files may be specified, *e.g.* typing `[4,6-9,12,14-16]` will read files 4,6,7,8,9,12,14,15,16. (Note that the brackets are required to distinguish this array of characters from a single string including commas. The brackets are unnecessary when there only one item.) For efficiency reasons it is sensible to give the file numbers in ascending order.

If you wish to extract all the files enter the wildcard *. `5-*` will read from 5 to the last file. The processing will continue until the end of the tape is reached; no error will result from this.

FMTCNV = _LOGICAL (Read)
This specifies whether or not format conversion will occur. If `FALSE`, the HDS type of the data array in the NDF will be the equivalent of the FITS data format on tape (*e.g.* BITPIX=16 creates a _WORD array). If `TRUE`, the data array in the current file, or all files in automatic mode, will be converted from the FITS data type on tape to _REAL in the NDF. The conversion applies the values of the FITS keywords BSCALE and BZERO to the tape data to generate the 'true' data values. If BSCALE and BZERO are not given in the FITS header, they are taken to be 1.0 and 0.0 respectively. The suggested default is `TRUE`.

GLOCON = _LOGICAL (Read)
If `FALSE`, a format-conversion query occurs for each FITS file. If `TRUE`, the value of FMTCNV is obtained before any file numbers and will apply to all data arrays. It is ignored in automatic mode—in effect it becomes `TRUE`. `[FALSE]`

LABEL = _LOGICAL (Read)
It should be `TRUE` if the tape has labelled files. Labelled files are non-standard. If `TRUE`, the application skips three file marks per file, rather that one. `[FALSE]`

LOGFILE = FILENAME (Read)
The file name of the text log of the FITS header cards. For group-format data the group parameters are evaluated and appended to the full header. The log includes the names of the output files used to store the data array or table. A null value (`!`) means that no log file is produced. `[!]`

MORE = _LOGICAL (Read)
A prompt asking if any more files are to be processed once the current list has been exhausted.

MT = DEVICE (Read)
Tape deck containing the data, usually an explicit device, though it can be a pre-assigned environment variable.

OUT = NDF (Write)
Output NDF structure holding the full contents of the FITS file. If the null value (`!`) is given no NDF will be created. This offers an opportunity to review the descriptors before deciding whether or not the data are to be extracted.

PREFIX = LITERAL (Read)
The prefix of the NDF's or table's file name. It is only used in the automatic mode.

REWIND = _LOGICAL (Read)
If it is `TRUE`, the tape drive is rewound before the reading of the FITS files commences. If it is `FALSE`, the tape is not rewound, and the current tape position is read

from file USRDEVDATASET.sdf. Note that file numbers are absolute and not relative. REWIND=FALSE is useful if you need to read a series of files, process them, then read some more, without having to remember the tape's position or apply unnecessary wear to the tape. [TRUE]

TABLE = FILENAME (Read)
Name of the text file to contain the table itself, read from the file. In manual mode, the suggested default filename is the name of description file less the "dscf" prefix, or if there is no description file or if the description file does not have the "dscf" prefix, the suggested name reverts to the catalogue name in the FITS header.

Examples:

```
fitsin mt=/dev/rmt/1n files=[2-4,9] auto prefix=ccd nofmtcnv
```

This reads files 2, 3, 4, and 9 from the FITS tape on device /dev/rmt/1n. The output NDF names will be ccd2, ccd3, ccd4, and ccd9 (assuming there are no groups). The data will not have format conversion.

```
fitsin mt=$TAPE files=* auto prefix=ccd fmtcnv logfile=jkt.log
```

This reads all the files from the FITS tape on the device assigned to the environment variable TAPE. The output files begin with a prefix "ccd". Integer data arrays are converted to real using the scale and zero found in the FITS header. A record of the headers and the names of the output files are written to the text file jkt.log.

References :

Wells, D.C., Greisen, E.W. & Harten, R.H. 1981, *Astron. Astrophys. Suppl. Ser.* **44**, 363.
Greisen, E.W. & Harten, R.H. 1981, *Astron. Astrophys. Suppl. Ser.* **44**, 371.
Grosbøl, P., Harten, R.H., Greisen, E.W & Wells, D.C. 1988 *Astron. Astrophys. Suppl. Ser.* **73**, 359.
Harten, R.H., Grosbøl, P., Greisen, E.W & Wells, D.C. 1988 *Astron. Astrophys. Suppl. Ser.* **73**, 365.

Related Applications :

KAPPA: FITSDIN, FITSHEAD, FITSIMP, FITSLIST; CONVERT: FITS2NDF; CURSA; FIGARO: RDFITS.

Implementation Status:

- The application processes tapes blocked at other than an integer multiple of 2880 bytes up to a maximum of 63360, provided it is a multiple of the number of bytes per data value.
- For simple or group format FITS:
  - IEEE floating point is supported.
  - If BUNIT is present its value will appear as the NDF's UNITS component.
  - If OBJECT is present its value will appear as the NDF's TITLE component.
  - If the BLANK item is present in the header, undefined pixels are converted from the BLANK value to Starlink-standard bad value during data conversion.

  - An AXIS component will be stored in the NDF if the CRVAL*n* keyword is present. (*n* is the number of the dimension.) If the CRPIX*n* keyword is absent it defaults to 1, and likewise for the CDELT*n* keyword. The value of CTYPE*n* is made the label of the axis structure.

- For groups format, a new NDF is created for each data array. The name of the NDF of the second and subsequent data arrays is generated by the application as the `<filename>G<number>`, where `<filename>` is the name of the first NDF, supplied by you or generated automatically, and `<number>` is the number of the group.

  Each group NDF contains the full header in the FITS extension, appended by the set of group parameters. The group parameters are evaluated using their scales and offsets, and made to look like FITS cards, whose keywords are derived from the values of PTYPE*m* in the main header. (*m* is the number of the group parameter.) The same format is used in the log file.

- If there is no data array on tape, *i.e.* the FITS file comprises header cards only, then a dummy vector data array of dimension two is created to make the output a valid NDF. This data array is undefined.

FITSLIST
Lists the FITS extension of an NDF

Description:

This application lists the FITS header stored in an NDF FITS extension. The list may either be reported directly to you, or written to a text file. The displayed list of headers can be augmented, if required, by the inclusion of FITS headers representing the current World Co-ordinate System defined by the WCS component in the NDF (see Parameter ENCODING).

Usage:

```
fitslist in [logfile]
```

Parameters:

ENCODING = LITERAL (Read)

If a non-null value is supplied, the NDF WCS component is used to generate a set of FITS headers describing the WCS, and these headers are added into the displayed list of headers (any WCS headers inherited from the FITS extension are first removed). The value supplied for ENCODING controls the FITS keywords that will be used to represent the WCS. The value supplied should be one of the encodings listed in the "World Co-ordinate Systems" section below. An error is reported if the WCS cannot be represented using the supplied encoding. A trailing minus sign appended to the end of the encoding indicates that only the WCS headers should be displayed (that is, the contents of the FITS extension are not displayed if the encoding ends with a minus sign). Also see the FULLWCS parameter. [!]

FULLWCS = _LOGICAL (Read)

Only accessed if ENCODING is non-null. If TRUE then all co-ordinate frames in the WCS component are written out. Otherwise, only the current Frame is written out. [FALSE]

IN = NDF (Read)

The NDF whose FITS extension is to be listed.

LOGFILE = FILENAME (Read)

The name of the text file to store a list of the FITS extension. If it is null (!) the list of the FITS extension is reported directly to you. [!]

Examples:

```
fitslist saturn
```

The contents of the FITS extension in NDF saturn are reported to you.

```
fitslist saturn fullwcs encoding=fits-wcs
```

As above but it also lists the standard FITS world-co-ordinate headers derived from saturn's WCS component, provided such information exists.

```
fitslist saturn fullwcs encoding=fits-wcs-
```

As the previous example except that it only lists the standard FITS world-co-ordinate headers derived from saturn's WCS component. The headers in the FITS extension are not listed.

```
fitslist ngc205 logfile=ngcfits.lis
```

The contents of the FITS extension in NDF ngc205 are written to the text file `ngcfits.lis`.

Notes:

- If the NDF does not have a FITS extension the application will exit.

World Co-ordinate Systems :

The ENCODING parameter can take any of the following values.

- `"FITS-IRAF"` — This uses keywords CRVAL*i* CRPIX*i*, CD*i_j*, and is the system commonly used by IRAF. It is described in the document "World Coordinate Systems Representations Within the FITS Format" by R.J. Hanisch and D.G. Wells, 1988, available by ftp from fits.cv.nrao.edu `/fits/documents/wcs/wcs88.ps.Z`.
- `"FITS-WCS"` — This is the FITS standard WCS encoding scheme described in the paper "Representation of celestial coordinates in FITS" (`http://www.cv.nrao.edu/fits/documents/wcs/wcs.html`).
  It is very similar to `"FITS-IRAF"` but supports a wider range of projections and co-ordinate systems.
- `"FITS-WCS(CD)"` — This is the same as `"FITS-WCS"` except that the scaling and rotation of the data array is described by CD matrix instead of a PC matrix with associated CDELT values.
- `"FITS-PC"` — This uses keywords CRVAL*i*, CDELT*i*, CRPIX*i*, PC*iiijjj*, *etc*, as in a previous (now superseded) draft of the above FITS world co-ordinate system paper by E.W. Greisen and M. Calabretta.
- `"FITS-AIPS"` — This uses conventions described in the document "Non-linear Coordinate Systems in AIPS" by Eric W. Greisen (revised 9th September, 1994), available by ftp from fits.cv.nrao.edu `/fits/documents/wcs/aips27.ps.Z`. It is currently employed by the AIPS data analysis facility, so its use will facilitate data exchange with AIPS. This encoding uses CROTA*i* and CDELT*i* keywords to describe axis rotation and scaling.
- `"FITS-AIPS++"` — This is an extension to `"FITS-AIPS"` that allows the use of a wider range of celestial projections, and is used by the AIPS++ project.
- `"FITS-CLASS"` — This uses the conventions of the CLASS project. CLASS is a software package for reducing single-dish radio and sub-mm spectroscopic data. It supports double-sideband spectra.

See `http://www.iram.fr/IRAMFR/GILDAS/doc/html/class-html/class.html`.

- `"DSS"` — This is the system used by the Digital Sky Survey, and uses keywords AMDX*n*, AMDY*n*, PLTRAH, *etc*.
- `"Native"` — This is the native system used by the AST library (see SUN/210), and provides a loss-free method for transferring WCS information between AST-based application. It allows more complicated WCS information to be stored and retrieved than any of the other encodings.

Related Applications :

KAPPA: FITSEDIT, FITSHEAD; FIGARO: FITSKEYS.

FITSMOD
Edits an NDF FITS extension via a text file or parameters

Description:

This application edits the FITS extension of an NDF file in a variety of ways. It permits insersion of new keywords, including comment lines; revision of existing keyword, values, and inline comments; relocation of keywords; deletion of keywords; printing of keyword values; and it can test whether or not a keyword exists. 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.

Control of the editing can be through parameters, or from a text file whose format is described in topic `"File Format"`.

Usage:

```
                 { keyword edit value comment position
fitsmod ndf      {
                 { table=?
            mode
```

Parameters:

COMMENT = LITERAL (Read)

The comments to be written to the KEYWORD keyword for the `"Update"`, `"Write"`, and `"Amend"`editing commands. A null value (`!`) gives a blank comment. 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. The same applies to the `"Amend"` edit, the choice depending on whether or not the KEYWORD keyword exists.

EDIT = LITERAL (Read)

The editing command to apply to the keyword. The allowed options are listed below.

`"Amend"` — acts as option `"Update"` if the keyword exists, but as the `"Write"` option should the keyword be 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 (see Parameter POSITION). 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 Parameter NEWKEY to obtain the new keyword.

"Update" — revises the value and/or the comment. If a secondary keyword is defined explicitly (Parameter POSITION), 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. The FITS extension is created first should it not exist.

KEYWORD = LITERAL (Read)

The name of the keyword to be edited in the FITS extension. 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 only alphanumeric characters, hyphen, and underscore. Any lowercase letters are converted to uppercase and blanks are removed before insertion, 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.

Both KEYWORD and POSITION keywords may have an occurrence specified in brackets [] following the name. 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 between the brackets other than a positive integer is interpreted as the first occurrence.

MODE = LITERAL (Read)

The mode by which the editing instructions are supplied. The alternatives are "File", which uses a text file; and "Interface" which uses parameters. ["Interface"]

NDF = NDF (Read and Write)

The NDF in which the FITS extension is to be edited.

NEWKEY = LITERAL (Read)

The name of the keyword to replace the KEYWORD keyword. It is only accessed when EDIT="Rename". 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 only alphanumeric characters, hyphen, and underscore.

POSITION = LITERAL (Read)

The position keyword name. 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. An occurrence may be specified (see Parameter KEYWORD for details).

The new keywords are inserted immediately before each corresponding position keyword. If any name in it does not exist in FITS array, or the null value (`!`) is supplied the consequences will be as follows. For a `"Write"`, `"Amend"` (new keyword), or `"Move"` edit, the KEYWORD keyword will be inserted just before the END card or appended to FITS array when the END card does not exist; for an `"Update"` or `"Amend"` (new keyword) edit, the edit keyword is not relocated.

A positional keyword is only accessed by the `"Move"`, `"Amend"`, `"Write"`, and `"Update"` editing commands.

READONLY = _LOGICAL (Read)

Determines if read or write access is requested for the NDF. If a `TRUE` value is supplied for READONLY, the NDF is opened for reading only. An error will then be reported if any of the requested editing operations would change the contents of the NDF. If a `FALSE` value is supplied for READONLY, the NDF is opened for both reading and writing, but an error will be reported if the NDF file is write-protected on disk. If the MODE parameter is set to `"File"`, the dynamic default value for READONLY is `FALSE`. If MODE is set to `"Interface"`, the dynamic default value for READONLY depends on the value of the EDIT parameter: `TRUE` for `"Print"` and `"Exist"`, and `FALSE` for all other editing commands. `[]`

STRING = _LOGICAL (Read)

When STRING is `FALSE`, inferred data typing is used for the `"Write"`, `"Update"`, and `"Amend"` editing commands. So for instance, if Parameter VALUE = `"Y"`, it would appears as logical `TRUE` rather than the string `'Y '` in the FITS header. See topic `"Value Data Type"`. When STRING is `TRUE`, the value will be treated as a string for the purpose of writing the FITS header. `[FALSE]`

TABLE = FILENAME (Read)

The text file containing the keyword translation table. The format of this file is described under `"File Format"`. For illustrations, see under `"Examples of the File Format"`.

VALUE = LITERAL (Read)

The new value of the KEYWORD keyword for the `"Update"`, `"Write"`, and `"Amend"` editing commands. The special value `"$V"` means use the current value of the KEYWORD keyword. This makes it possible to modify a comment, leaving the value unaltered. In addition `"$V(keyword)"` requests that the value of the reference keyword given between the parentheses be assigned to the keyword being edited. This reference keyword must exist and have a value for a `"Write"` or `"Amend"` (new keyword) edit; whereas the FITS-header value is unchanged for `"Update"` or `"Amend"` (keyword exists) if there are problems with this reference keyword.

Results Parameters:

EXISTS = _LOGICAL (Write)

The result of the final `"Exist"` operation (see Parameter EDIT).

Examples:

```
fitsmod dro42 bscale exist
```

This reports `TRUE` or `FALSE` depending on whether or not the FITS keyword BSCALE exists in the FITS extension of the NDF called dro42.

```
fitsmod dro42 bscale p
```

This reports the value of the keyword BSCALE stored in the FITS extension of the NDF called dro42. An error message will appear if BSCALE does not exist.

```
fitsmod abc edit=move keyword=bscale position=bzero
```

This moves the keyword BSCALE to lie immediately before keyword BZERO in the FITS extension of the NDF called abc. An error will result if either BSCALE or BZERO does not exist.

```
fitsmod dro42 airmass dele
```

This deletes the keyword AIRMASS, if it exists, in the FITS extension of the NDF called dro42.

```
fitsmod ndf=dro42 edit=d keyword=airmass[2]
```

This deletes the second occurrence of keyword AIRMASS, if it exists, in the FITS extension of the NDF called dro42.

```
fitsmod @100 airmass w 1.456 "Airmass at mid-observation"
```

This creates the keyword AIRMASS in the FITS extension of the NDF called 100, assigning the keyword the real value 1.456 and comment `"Airmass at mid-observation"`. The header is located just before the end. The FITS extension is created if it does not exist.

```
fitsmod @100 airmass w 1.456 "Airmass at mid-observation" phase
```

As the previous example except that the new keyword is written immediately before keyword PHASE.

```
fitsmod obe observer u value="O'Leary" comment=$C
```

This updates the keyword OBSERVER with value `"O'Leary"`, retaining its old comment. The modified FITS extension lies within the NDF called obe.

```
fitsmod test filter w position=end value=27 comment=!  string
```

This creates the keyword FILTER in the FITS extension of the NDF called test, assigning the keyword the string value `"27"`. There is no comment. The keyword is located at the end of the headers, but before any END card. The FITS extension is created if it does not exist.

```
fitsmod test edit=w keyword=detector comment="     Detector name"
```

```
value=$V(ing.dethead) accept
```

This creates the keyword DETECTOR in the FITS extension of the NDF called test, assigning the keyword the value of the existing hierarchical keyword ING.DETHEAD. The comment is `"  Detector name"`, the leading spaces are significant. The keyword is located at the current position keyword. The FITS extension is created if it does not exist.

```
fitsmod datafile mode=file table=fitstable.txt
```

This edits the FITS-extension of the NDF called datafile, creating the FITS extension if it does not exist. The editing instructions are stored in the text file called `fitstable.txt`.

Notes:

- Requests to move, assign values or comments, the following reserved keywords in the FITS extension are ignored: SIMPLE, BITPIX, NAXIS, NAXIS*n*, EXTEND, PCOUNT, GCOUNT, XTENSION, BLOCKED, and END.
- 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.
- To be sure that the resultant FITS extension is what you desired, you should inspect it using the command FITSLIST before exporting the data. If there is something wrong, you may find it convenient to use command FITSEDIT to make minor corrections.

Parameter Defaults :

All the parameters have a suggested default of their current value, except NDF, which uses the global current dataset.

Timing :

Approximately proportional to the number of FITS keywords to be edited. `"Update"` and `"Write"` edits require the most time.

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{[occurrence]}{(keyword{[occurrence]})} {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`, `Print`, `Rename`, `Write`, and `Update`, and can be abbreviated to the initial upper-case letter. It is not case insensitive to afford some protection against typing errors.

  - `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` acts like `Update` if the keyword supplied in `"Field 2"` exists, and like `Write` otherwise.
  - `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 eight 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 between the brackets 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 appearing 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 `Write`, `Update`, and `Amend` 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 Type".

For the Update, Write, and Amend 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 to Update, Write, or Amend.

- Field 4: This specifies the comment to assign to the edited keyword for the Write, Update, and Amend 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 Type :

The data type of a value is determined as follows:

- For the text-file, 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.
- For prompting the value is a string when Parameter STRING is TRUE.
- 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"`. Note the full stop.

`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.

References :

"A User's Guide for the Flexible Image Transport System (FITS)", NASA/Science Office of Science and Technology (1994).

Related Applications :

KAPPA: FITSEDIT, FITSEXIST, FITSEXP, FITSHEAD, FITSIMP, FITSLIST, FITSVAL, FITSWRITE.

FITSTEXT
Creates an NDF FITS extension from a text file

Description:

This application takes a version of a FITS header stored in a text file, and inserts it into the FITS extension of an NDF . The header is not copied verbatim as some validation of the headers as legal FITS occurs. An existing FITS extension is removed.

Usage:

```
fitstext ndf file
```

Parameters:

NDF = NDF (Read and Write)

The name of the NDF to store the FITS header information.

FILE = FILENAME (Read)

The text file containing the FITS headers. Each record should be the standard 80-character 'card image'. If the file has been edited care is needed to ensure that none of the cards are wrapped on to a second line.

Examples:

```
fitstext hh73 headers.lis
```

This places the FITS headers stored in the text file called `headers.lis` in the FITS extension of the NDF called hh73.

Notes:

- The validation process performs the following checks on each header 'card':
  a) the length of the header is no more than 80 characters, otherwise it is truncated;
  b) the keyword only contains uppercase Latin alphabetic characters, numbers, underscore, and hyphen (the header will not be copied to the extension except when the invalid characters are 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, except for non-mandatory keywords which have a double-precision value requiring more than 20 digits;
  h) the comment delimiter is in column 32 or two characters following the value, whichever is greater;
  i) an equals sign in column 9 of a commentary card is replaced by a space; and
  j) comments begin at least two columns after the end of the comment delimiter.
- The validation issues warning messages at the normal reporting level for violations a), b), c), d), and i).

- The validation can only go so far. If any of your header lines are ambiguous, the resulting entry in the FITS extension may not be what you intended. Therefore, you should inspect the resulting FITS extension using the command FITSLIST before exporting the data. If there is something wrong, you may find it convenient to use command FITSEDIT to make minor corrections.

Related Applications :

KAPPA: FITSEDIT, FITSEXP, FITSLIST; CONVERT: NDF2FITS.

FITSURFACE
Fits a polynomial surface to two-dimensional data array

Description:

This task fits a surface to a two-dimensional data array stored array within an NDF data structure. At present it only permits a fit with a polynomial, and the coefficients of that surface are stored in a POLYNOMIAL structure (SGP/38) as an extension to that NDF.

Unlike SURFIT, neither does it bin the data nor does it reject outliers.

Usage:

```
fitsurface ndf [fittype] { nxpar nypar
                         { [knots]
                  fittype
```

Parameters:

COSYS = LITERAL (Read)

The co-ordinate system to be used. This can be either "World" or "Data". If COSYS="World" the co-ordinates used to fits the surface are pixel co-ordinates. If COSYS="Data" the data co-ordinates used are used in the fit, provided there are axis centres present in the NDF. COSYS="World" is recommended. [Current co-ordinate system]

FITTYPE = LITERAL (Read)

The type of fit. It must be either "Polynomial" for a polynomial or "Spline" for a bi-cubic spline. ["Polynomial"]

KNOTS(2) = _INTEGER (Read)

The number of interior knots used for the bi-cubic-spline fit along the *x* and *y* axes. These knots are equally spaced within the image. Both values must be in the range 0 to 11. If you supply a single value, it applies to both axes. Thus 1 creates one interior knot, [5,4] gives five along the *x* axis and four along the *y* direction. Increasing this parameter values increases the flexibility of the surface. Normally, 4 is a reasonable value. The upper limit of acceptable values will be reduced along each axis when its binned array dimension is fewer than 29. KNOTS is only accessed when FITTYPE="Spline". The default is the current value, which is 4 initially. []

NDF = NDF (Update)

The NDF containing the two-dimensional data array to be fitted.

NXPAR = _INTEGER (Read)

The number of fitting parameters to be used in the *x* direction. It must be in the range 1 to 15 for a polynomial fit. Thus 1 gives a constant, 2 a linear fit, 3 a quadratic *etc*. Increasing this parameter increases the flexibility of the surface in the *x* direction. The upper limit of acceptable values will be reduced for arrays with an *x* dimension fewer than 29. NXPAR is only accessed when FITTYPE="Polynomial".

NYPAR = _INTEGER (Read)

The number of fitting parameters to be used in the *y* direction. It must be in the range 1 to 15 for a polynomial fit. Thus 1 gives a constant, 2 a linear fit, 3 a quadratic *etc*.

Increasing this parameter increases the flexibility of the surface in the *y* direction. The upper limit of acceptable values will be reduced for arrays with a *y* dimension fewer than 29. NYPAR is only accessed when FITTYPE="Polynomial".

OVERWRITE = _LOGICAL (Read)
OVERWRITE=TRUE, allows an NDF extension containing an existing surface fit to be overwritten. OVERWRITE=FALSE protects an existing surface-fit extension, and should one exist, an error condition will result and the task terminated. [TRUE]

VARIANCE = _LOGICAL (Read)
A flag indicating whether any variance array present in the NDF is used to define the weights for the fit. If VARIANCE is TRUE and the NDF contains a variance array this will be used to define the weights, otherwise all the weights will be set equal. [TRUE]

XMAX = _DOUBLE (Read)
The maximum *x* value to be used in the fit. This must be greater than or equal to the *x* co-ordinate of the right-hand pixel in the data array. Normally this parameter is automatically set to the maximum *x* co-ordinate found in the data, but this mechanism can be overridden by specifying XMAX on the command line. The parameter is provided to allow the fit limits to be fine tuned for special purposes. It should not normally be altered. If a null (!) value is supplied, the value used is the maximum *x* co-ordinate of the fitted data. [!]

XMIN = _DOUBLE (Read)
The minimum *x* value to be used in the fit. This must be smaller than or equal to the *x* co-ordinate of the left-hand pixel in the data array. Normally this parameter is automatically set to the minimum *x* co-ordinate found in the data, but this mechanism can be overridden by specifying XMIN on the command line. The parameter is provided to allow the fit limits to be fine tuned for special purposes. It should not normally be altered. If a null (!) value is supplied, the value used is the minimum *x* co-ordinate of the fitted data. [!]

YMAX = _DOUBLE (Read)
The maximum *y* value to be used in the fit. This must be greater than or equal to the *y* co-ordinate of the top pixel in the data array. Normally this parameter is automatically set to the maximum *y* co-ordinate found in the data, but this mechanism can be overridden by specifying YMAX on the command line. The parameter is provided to allow the fit limits to be fine tuned for special purposes. It should not normally be altered. If a null (!) value is supplied, the value used is the maximum *y* co-ordinate of the fitted data. [!]

YMIN = _DOUBLE (Read)
The minimum *y* value to be used in the fit. This must be smaller than or equal to the *y* co-ordinate of the bottom pixel in the data array. Normally this parameter is automatically set to the minimum *y* co-ordinate found in the data, but this mechanism can be overridden by specifying YMIN on the command line. The parameter is provided to allow the fit limits to be fine tuned for special purposes. It should not normally be altered. If a null (!) value is supplied, the value used is the minimum *y* co-ordinate of the fitted data. [!]

Examples:

```
fitsurface virgo nxpar=4 nypar=4 novariance
```

This fits a bi-cubic polynomial surface to the data array in the NDF called virgo. All the data values are given equal weight. The coefficients of the fitted surface are stored in an extension of virgo.

```
fitsurface virgo nxpar=4 nypar=4
```

As the first example except the data variance, if present, is used to weight the data values.

```
fitsurface virgo fittype=spl
```

As the previous example except a B-spline fit is made using four interior knots along both axes.

```
fitsurface virgo fittype=spl knots=[10,7]
```

As the previous example except now there are ten interior knots along the x axis and seven along the y axis.

```
fitsurface mkn231 nxpar=6 nypar=2 cosys=d xmin=-10.0 xmax=8.5
```

This fits a polynomial surface to the data array in the NDF called mkn231. A fifth order is used along the x direction, but only a linear fit along the y direction. The fit is made between x data co-ordinates -10.0 to 8.5. The variance weights the data values. The coefficients of the fitted surface are stored in an extension of mkn231.

Notes:

A polynomial surface fit is stored in a SURFACEFIT extension, component FIT of type POLYNOMIAL, variant CHEBYSHEV or BSPLINE. This is read by MAKESURFACE to create a NDF of the fitted surface.

For further details of the CHEBYSHEV variant see SGP/38. The CHEBYSHEV variant includes the fitting variance for each coefficient.

The BSPLINE variant structure is provisional. It contain the spline coefficients in the two-dimensional DATA_ARRAY component, the knots in XKNOTS and YKNOTS arrays, and a scaling factor to restore the original values after spline evaluation recorded in component SCALE. All of these components have type _REAL.

Also stored in the SURFACEFIT extension are the r.m.s. deviation to the fit (component RMS), the maximum absolute deviation (component RSMAX), and the co-ordinate system (component COSYS) translated to AST Domain names AXIS (for Parameter COSYS="Data") and PIXEL ("World").

Related Applications :

KAPPA: MAKESURFACE, SURFIT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, and HISTORY components of an NDF data structure.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using double-precision floating point.

FITSVAL
Reports the value of a keyword in the FITS extension.

Description:

This application reports the value of a keyword in the FITS extension ('airlock') of an NDF file. The keyword's value and comment are also stored in output parameters.

Usage:

```
fitsval ndf keyword
```

Parameters:

KEYWORD = LITERAL (Read)

The name of an existing keyword in the FITS extension whose value is to be reported. 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 only alphanumeric characters, hyphen, and underscore. Any lowercase letters are converted to uppercase and blanks are removed before comparison with the existing keywords.

KEYWORD may have an occurrence specified in brackets [] following the name. This enables the values to be obtained for keywords that appear more than once. 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 between the brackets other than a positive integer is interpreted as the first occurrence.

The suggested value is the current value.

NDF = NDF (Read)

The NDF containing the FITS keyword.

Results Parameters:

COMMENT = LITERAL (Write)

The comment of the keyword.

VALUE = LITERAL (Write)

The value of the keyword.

Examples:

```
fitsval abc bscale
```

This reports the value of the FITS keyword BSCALE, which is located within the FITS extension of the NDF called abc.

```
fitsval ndf=abc keyword=date[2]
```

This reports the value of the second occurrence FITS keyword DATE, which is located within the FITS extension of the NDF called abc.

Related Applications :

KAPPA: FITSEDIT, FITSEXIST, FITSHEAD, FITSLIST, FITSMOD.

FITSWRITE
Writes a new keyword to the FITS extension

Description:

This application writes a new keyword in an NDF's FITS extension given a value and an optional inline comment. It allows the location of the new keyword to be specified. The FITS extension is created if it does not exist.

It is a synonym for `fitsmod edit=write mode=interface position=!`.

Usage:

```
fitswrite ndf keyword value=?  comment=?
```

Parameters:

COMMENT = LITERAL (Read)

The comments to be written to the KEYWORD keyword. A null value (`!`) gives a blank comment. 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 is blank.

KEYWORD = LITERAL (Read)

The name of the new keyword in the FITS extension. 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 only alphanumeric characters, hyphen, and underscore. Any lowercase letters are converted to uppercase and blanks are removed before comparison with the existing keywords.

Note that it is not normal to have multiple occurrences of a keyword in a FITS header, unless it is blank, COMMENT or HISTORY.

The suggested value is the current value.

NDF = NDF (Read and Write)

The NDF containing the FITS extension into which the new FITS keyword.

POSITION = LITERAL (Read)

The position keyword name. 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 20. Each keyword must be no longer than 8 characters. When locating the position card, comparisons are made in uppercase and with the blanks removed. An occurrence may be specified (see Parameter KEYWORD for details).

The new keywords are inserted immediately before each corresponding position keyword. If any name in it does not exist in FITS array, or the null value (`!`) is supplied, the KEYWORD keyword will be inserted just before the END card or appended to FITS array when the END card does not exist. `[!]`

STRING = _LOGICAL (Read)

When STRING is `FALSE`, inferred data typing is used. So for instance if Parameter

VALUE = "Y", it would appears as logical TRUE rather than the string 'Y ' in the FITS header. See topic "Value Data Type". When STRING is TRUE, the value will be treated as a string for the purpose of writing the FITS header. [FALSE]

VALUE = LITERAL (Read)
The new value of the KEYWORD keyword. The special value "$V" means use the current value of the KEYWORD keyword. This makes it possible to modify a comment, leaving the value unaltered. In addition "$V(keyword)" requests that the value of the reference keyword given between the parentheses be assigned to the keyword being written. This reference keyword must exist and have a value.

Examples:

```
fitswrite abc bscale value=1.234
```

This writes the FITS keyword BSCALE just before the end of the FITS extension, which is located within the NDF called abc. It assigns BSCALE a value of 1.234. There is no inline comment.

```
fitswrite @100 airmass value=1.456 comment="Airmass at mid-observation"
```

This creates the keyword AIRMASS in the FITS extension of the NDF called 100, assigning the keyword the real value 1.456 and comment "Airmass at mid-observation". The header is located just before the end.

```
fitswrite @100 airmass value=1.456 "Airmass at mid-observation"
position=phase
```

As the previous example except that the new keyword is written immediately before keyword PHASE.

```
fitswrite afcyg observer value="O'Leary" comment=$C(prininv)
```

This writes the keyword OBSERVER with value "O'Leary", and its comment is copied from keyword PRININV. The modified FITS extension lies within the NDF called afcyg.

```
fitswrite test filter position=end value=27 comment=!  string
```

This creates the keyword FILTER in the FITS extension of the NDF called test, assigning the keyword the string value "27". There is no comment. The keyword is located at the end of the headers, but before any END card.

```
fitswrite ndf=test keyword=detector comment="     Detector name"
value=$V(ing.dethead) accept
```

This creates the keyword DETECTOR in the FITS extension of the NDF called test, assigning the keyword the value of the existing hierarchical keyword ING.DETHEAD.

The comment is `"  Detector name"`, the leading spaces are significant. The keyword is located at the current position keyword.

Value Data Type :

The data type of a value is determined as follows:

- For the text-file, 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.
- For prompting the value is a string when Parameter STRING is `TRUE`.
- 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.

Related Applications :

KAPPA: FITSEDIT, FITSEXP, FITSMOD.

FLIP
Reverses an NDF's pixels along a specified dimension

Description:

This application reverses the order of an NDF's pixels along a specified dimension, leaving all other aspects of the data structure unchanged.

Usage:

```
flip in out dim
```

Parameters:

AXIS = _LOGICAL (Read)
If a TRUE value is given for this parameter (the default), then any axis values associated with the NDF dimension being reversed will also be reversed in the same way. If a FALSE value is given, then all axis values will be left unchanged. [TRUE]

DIM = _INTEGER (Read)
The number of the dimension along which the NDF's pixels should be reversed. The value should lie between 1 and the total number of NDF dimensions. If the NDF has only a single dimension, then this parameter is not used, a value of 1 being assumed.

IN = NDF (Read)
The input NDF data structure whose pixel order is to be reversed.

OUT = NDF (Write)
The output NDF data structure.

TITLE = LITERAL (Read)
A title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

Examples:

```
flip a b 2
```

Reverses the pixels in the NDF called a along its second dimension to create the new NDF called b.

```
flip specin specout
```

If specin is a one-dimensional spectrum, then this example reverses the order of its pixels to create a new spectrum specout. Note that no value for the DIM parameter need be supplied in this case.

```
flip in=cube out=newcube dim=2 noaxis
```

Reverses the order of the pixels along dimension 2 of the NDF called cube to give newcube, but leaves the associated axis values in their original order.

Notes:

The pixel-index bounds of the NDF are unchanged by this routine.

Related Applications :

KAPPA: ROTATE, REGRID; FIGARO: IREVX, IREVY, IROT90.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. The data type of the input pixels is preserved in the output NDF.

FOURIER
Performs forward and inverse Fourier transforms of one- or two-dimensional NDFs

Description:

This application performs forward or reverse Fast Fourier Transforms (FFTs) of one- or two-dimensional NDFs . The output in the forward transformation (from the space domain to the Fourier) can be produced in Hermitian form in a single NDF, or as two NDFs giving the real and imaginary parts of the complex transform, or as two NDFs giving the power and phase of the complex transform. Any combination of these may also be produced. The inverse procedure accepts any of these NDFs and produces a purely real output NDF.

Any bad pixels in the input NDF may be replaced by a constant value. Input NDFs need neither be square, nor be a power of 2 in size in either dimension; their shape is arbitrary.

The Hermitian transform is a single image in which each quadrant consisting of a linear combination of the real and imaginary parts of the transform. This form is useful if you just want to multiply the Fourier transform by some known purely real mask and then invert it to get a filtered image. However, if you want to multiply the Fourier transform by a complex mask (*e.g.* the Fourier transform of another NDF), or do any other operation involving combining complex values, then the Hermitian NDF must be untangled into separate real and imaginary parts.

There is an option to swap the quadrants of the input NDF around before performing a forward FFT. This is useful if you want to perform convolutions with the FFTs, since the point-spread function (PSF) image can be created with the PSF centre at the array centre, rather than at pixel (1, 1) as is usually required.

Usage:

```
fourier in hermout
```

Parameters:

FILLVAL = LITERAL (Read)

A value to replace bad pixels before performing the transform. The input image is also padded with this value if necessary to form an image of acceptable size. A value of `"Mean"` will cause the mean value in the array to be used. `[0.0]`

HERMIN = NDF (Read)

Hermitian frequency-domain input NDF containing the complex transform. If null is entered no Hermitian NDF is read and then the application should be supplied either separate real and imaginary NDFs, or the power and phase NDFs. Prompting will not occur if one of the other (inverse) input NDFs has been given on the command line, but not HERMIN as well. This parameter is only relevant for an inverse transformation.

HERMOUT = NDF (Write)

Hermitian output NDF from a forward transform. If a null value is given then this NDF is not produced.

HM_TITLE = LITERAL (Read)
Title for the Hermitian Fourier-transform output NDF. A null (`!`) value means using the title of the input NDF. `["KAPPA - Fourier - Hermitian"]`

IM_TITLE = LITERAL (Read)
Title for the frequency-domain imaginary output NDF. A null (`!`) value means using the title of the input NDF. `["KAPPA - Fourier - Imaginary"]`

IMAGIN = NDF (Read)
Input frequency-domain NDF containing the imaginary part of the complex transform. If a null is given then an image of zeros is assumed unless a null is also given for REALIN, in which case the input is requested in power and phase form. This parameter is only available if HERMIN is not used. One way to achieve that is to supply IMAGIN, but not HERMIN, on the command line. This parameter is only relevant for an inverse transformation.

IMAGOUT = NDF (Write)
Frequency-domain output NDF containing the imaginary part of the complex Fourier transform. If a null value is given then this NDF is not produced. `[!]`

IN = NDF (Read)
Real (space-domain) input NDF for a forward transformation. There are no restrictions on the size or shape of the input NDF, although the it may have to be padded or trimmed before being transformed. This parameter is only used if a forward transformation was requested.

INVERSE = _LOGICAL (Read)
If `TRUE`, then the inverse transform—frequency domain to space domain—is required, otherwise a transform from the space to the frequency domain is undertaken. `[FALSE]`

OUT = NDF (Write)
Real space-domain output NDF. This parameter is only used if an inverse transformation is requested.

PH_TITLE = LITERAL (Read)
Title for the frequency-domain phase output NDF. A null (`!`) value means using the title of the input NDF. `["KAPPA - Fourier - Phase"]`

PHASEIN = NDF (Read)
Input frequency-domain NDF containing the phase of the complex transform. If a null is given then an image of zeros is assumed unless a null is also given for PHASEIN, in which case the application quits. This parameter is only available if HERMIN, REALIN, and IMAGIN are all not used. One way to achieve that is to supply PHASEIN, but none of the aforementioned parameters, on the command line. This parameter is only relevant for an inverse transformation.

PHASEOUT = NDF (Write)
Frequency-domain output NDF containing the phase of the complex Fourier transform. If a null value is given then this NDF is not produced. `[!]`

POWERIN = NDF (Read)
Input frequency-domain NDF containing the modulus of the complex transform. Note, this should be the square root of the power rather than the power itself. If a null is given then an image of zeros is assumed unless a null is also given for PHASEIN, in which case the application quits. This parameter is only available if

HERMIN, REALIN, and IMAGIN are all not used. One way to achieve that is to supply POWERIN, but none of the aforementioned parameters, on the command line. This parameter is only relevant for an inverse transformation.

POWEROUT = NDF (Write)
Frequency-domain output NDF containing the modulus of the complex Fourier transform. Note, this is the square root of the power rather than the power itself. If a null value is given then this NDF is not produced. [!]

PW_TITLE = LITERAL (Read)
Title for the frequency-domain power output NDF. A null (!) value means using the title of the input NDF. ["KAPPA - Fourier - Power"]

REALIN = NDF (Read)
Input frequency-domain NDF containing the real part of the complex transform. If a null is given then an image of zeros is assumed unless a null is also given for IMAGIN, in which case the input is requested in power and phase form. This parameter is only available if HERMIN is not used. One way to achieve that is to supply REALIN, but not HERMIN, on the command line. This parameter is only relevant for an inverse transformation.

REALOUT = NDF (Write)
Frequency-domain output NDF containing the real part of the complex Fourier transform. If a null value is given then this NDF is not produced. [!]

RL_TITLE = LITERAL (Read)
Title for the frequency-domain real output NDF. A null (!) value means using the title of the input NDF. ["KAPPA - Fourier - Real"]

SHIFT = _LOGICAL (Read)
If TRUE, the transform origin is to be located at the array's centre. This is implemented by swapping bottom-left and top-right, and bottom-right and top-left array quadrants, before doing the transform. This results in the transformation effectively being done about pixel x = INT(NAXIS1/2)+1 and y = INT(NAXIS2/2)+1, where NAXISn are the padded or trimmed dimensions of the NDF. [FALSE]

TRIM = _LOGICAL (Read)
If TRUE, when the input array dimension cannot be processed by the transform, the output arrays will be trimmed rather than padded with the fill value. [FALSE]

TITLE = LITERAL (Read)
Title for the real space-domain output NDF. A null (!) value means using the title of the input NDF. ["KAPPA - Fourier"]

Examples:

```
fourier galaxy ft_gal
```

Makes an Hermitian Fourier transform stored in an NDF called ft_gal from the two-dimensional NDF called galaxy.

```
fourier hermin=ft_gal out=galaxy inverse
```

Takes an Hermitian Fourier transform stored in an NDF called ft_gal and performs the inverse transformation to yield a normal (spatial domain) image in NDF galaxy.

```
fourier in=galaxy powerout=galpow hermout=ft_gal fillval=mean
```

Makes an Hermitian Fourier transform stored in an NDF called ft_gal from the two-dimensional NDF called galaxy. Any bad values in galaxy are replaced by the mean data value of galaxy. In addition the power of the transform is written to an NDF called galpow.

```
fourier realin=real_gal out=galaxy inverse
```

Takes the real component of a Fourier transform stored in an NDF called real_gal and performs the inverse transformation to yield a normal image in NDF galaxy.

Notes:

- See the NAG documentation, Chapter C06, and/or KAPLIBS routine `kpg1_hmltx.gen` for more details of Hermitian Fourier transforms.

Related Applications :

KAPPA: CONVOLVE, LUCY, MEM2D, WIENER; FIGARO: BFFT, CMPLX*, COSBELL, FFT, *2CMPLX.

Implementation Status:

- AXIS, VARIANCE and QUALITY are not propagated from the input to output NDFs, but the LABEL, TITLE, HISTORY components and all extensions are. Arithmetic is performed using single- or double-precision floating point, as appropriate for the type of the data array.

GAUSMOOTH
Smooths a one- or two-dimensional image using a Gaussian filter

Description:

This application smooths an NDF using a one- or two-dimensional symmetrical Gaussian point spread function (PSF) of specified width, or widths and orientation. Each output pixel is the PSF-weighted mean of the input pixels within the filter box.

The NDF may have up to three dimensions. If it has three dimensions, then the filter is applied in turn to each plane in the cube and the result written to the corresponding plane in the output cube. The orientation of the smoothing plane can be specified using the AXES parameter.

Usage:

```
gausmooth in out fwhm
```

Parameters:

AXES(2) = _INTEGER (Read)

This parameter is only accessed if the NDF has exactly three significant pixel axes. It should be set to the indices of the NDF pixel axes which span the plane in which smoothing is to be applied. All pixel planes parallel to the specified plane will be smoothed independently of each other. The dynamic default is the indices of the first two significant axes in the NDF. `[]`

BOX() = _INTEGER (Read)

The *x* and *y* sizes (in pixels) of the rectangular region over which the Gaussian PSF should be applied at each point. The smoothing PSF will be set to zero outside this rectangle, which should therefore be sufficiently large not to truncate the PSF too early. A square region is defined should only one size be given. For a one-dimensional or circular Gaussian a second size is ignored. Two values are expected when an elliptical PSF is requested (see the description of Parameter FWHM).

The values given will be rounded up to positive odd integers if necessary. If a null (`!`) value is supplied, the value used is just sufficient to accommodate the Gaussian PSF out to a radius of 3 standard deviations. Note that the time taken to perform the smoothing increases in approximate proportion to the value of this parameter for a circular Gaussian, and in proportion to the product of the two box sizes for an elliptical Gaussian. `[!]`

FWHM() = _REAL (Read)

This specifies whether a circular or elliptical Gaussian point-spread function is used in smoothing a two-dimensional image. If one value is given it is the full-width at half-maximum of a one-dimensional or circular Gaussian PSF. (Indeed only one value is permitted for a one-dimensional array.) If two values are supplied, this parameter becomes the full-width at half-maximum of the major and minor axes of an elliptical Gaussian PSF. Values between 0.1 and 10000.0 pixels should be given. Note that unless a non-default value is specified for the BOX parameter, the time taken to perform the smoothing will increase in approximate proportion to the value(s) of FWHM. The suggested default is the current value.

IN = NDF (Read)
The input NDF containing the one-, two-, or three-dimensional image to which Gaussian smoothing is to be applied.

ORIENT = _REAL (Read)
The orientation of the major axis of the elliptical Gaussian PSF, measured in degrees in an anti-clockwise direction from the *x* axis of the NDF. ORIENT is not obtained if FWHM has one value, *i.e.* a circular Gaussian PSF will be used to smooth the image, or the input NDF is one-dimensional. The suggested default is the current value.

OUT = NDF (Write)
The output NDF which is to contain the smoothed image.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the input NDF to be used. [!]

WLIM = _DOUBLE (Read)
If the input image contains bad pixels, then this parameter may be used to determine the number of good pixels which must be present within the PSF area before a valid output pixel is generated. It can be used, for example, to prevent output pixels from being generated in regions where good pixels are only present in the wings of the PSF.

By default, a null (!) value is used for WLIM, which causes the pattern of bad pixels to be propagated from the input image to the output image unchanged. In this case, smoothed output values are only calculated for those pixels which are not bad in the input image.

If a numerical value is given for WLIM, then it specifies the minimum PSF-weighted fraction of good pixels which must be present in the PSF area (*i.e.* box) in order to generate a good output pixel. The maximum value, in the absence of bad pixels, is unity. If the specified minimum fraction of good input pixels is not present, then a bad output pixel will result, otherwise a smoothed output value will be calculated. The value of this parameter should lie between 1E-6 and 1.0. [!]

Examples:

```
gausmooth image1 image2 5.0
```

Smooths the two-dimensional image held in the NDF structure image1 using a symmetrical Gaussian PSF with a full-width at half-maximum of 5 pixels. The smoothed image is written to image2. If any pixels in the input image are bad, then the corresponding pixels in the output image will also be bad.

```
gausmooth spectrum1 spectrum2 5.0 box=9
```

Smooths the one-dimensional image held in the NDF structure spectrum1 using a symmetrical Gaussian PSF with a full-width at half-maximum of 5, and is evaluated over a length of 9 pixels. The smoothed image is written to spectrum2. If any pixels in the input image are bad, then the corresponding pixels in the output image will also be bad.

```
gausmooth in=a out=b fwhm=3.5 box=31
```

Smooths the two-dimensional image held in the NDF structure a, writing the result into the structure b. The Gaussian smoothing PSF has a full-width at half-maximum of 3.5 pixels and is evaluated over a large square of size 31x31 pixels.

```
gausmooth in=a out=b fwhm=[4,3] orient=52.7 box=[29,33]
```

Smooths the two-dimensional image held in the NDF structure a, writing the result into the structure b. The elliptical Gaussian smoothing PSF has full-width at half-maximum of 4 pixels along its major axis and three pixels along its minor axis, and is evaluated over a large rectangle of size 29x33 pixels. The major axis of the PSF is oriented 52.7 degrees anti-clockwise from the x axis of the data array.

```
gausmooth ngc1097 ngc1097s fwhm=7.2 wlim=0.1
```

Smooths the specified image data using a Gaussian PSF with a full-width at half-maximum of 7.2. An output value is calculated for any pixel for which the PSF-weighted fraction of good input pixels is at least 0.1. This will cause the smoothing operation to fill in moderately sized regions of bad pixels.

Timing :

For a circular PSF, the execution time is approximately proportional to the number of pixels in the image to be smoothed and to the value given for the BOX parameter. By default, this latter value is proportional to the value given for FWHM. For an elliptical PSF, the execution time is approximately proportional to the number of pixels in the image to be smoothed and to the product of the values given for the BOX parameter. By default, these latter values are approximately proportional to the values given for FWHM. Execution time will be approximately doubled if a variance array is present in the input NDF.

Related Applications :

KAPPA: BLOCK, CONVOLVE, FFCLEAN, MATHS, MEDIAN, PSF; FIGARO: ICONV3, ISMOOTH, IXSMOOTH, MEDFILT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported. The bad-pixel flag is also written for the data and variance arrays.
- All non-complex numeric data types can be handled. Arithmetic is performed using single-precision floating point, or double precision, if appropriate.

GDCLEAR
Clears a graphics device and purges its database entries

Description:

This application software resets a graphics device. In effect the device is cleared. It purges the graphics-database entries for the device. Optionally, only the current picture is cleared and the database unchanged. (Note the clearing of the current picture may not work on some graphics devices.)

Usage:

```
gdclear [device] [current]
```

Parameters:

CURRENT = _LOGICAL (Read)
If TRUE then only the current picture is cleared. [FALSE]

DEVICE = DEVICE (Read)
The graphics device to be cleared. [Current graphics device]

Examples:

```
gdclear
```

Clears the current graphics device and purges its graphics-database entries.

```
gdclear current
```

Clears the current picture on the current graphics device.

```
gdclear xw
```

Clears the xw device and purges its graphics-database entries.

Related Applications :

KAPPA: GDSET, GDSTATE.

GDNAMES
Shows which graphics devices are available

Description:

The routine displays a list of the graphics devices available and the names (both traditional Starlink GNS names and the equivalent PGPLOT names) which identify them. Each name is accompanied by a brief descriptive comment.

Usage:

```
gdnames
```

GDSET
Selects a current graphics device

Description:

This application selects a current graphics device. This device will be used for all applications requiring an graphics device until changed explicitly.

Usage:

```
gdset device
```

Parameters:

DEVICE = DEVICE (Read)

The graphics device to become the current graphics device.

Examples:

```
gdset xwindows
```

Makes the xwindows device the current graphics device.

GDSTATE
Shows the current status of a graphics device

Description:

This application displays information about the current graphics database picture on a graphics device, including the extreme axis values in any requested co-ordinate Frame (see Parameter FRAME). Information is written to various output parameters for use by other applications, and is also written to the screen by default (see Parameter REPORT). An outline may be drawn around the current picture if required (see Parameter OUTLINE).

A list of the colours in the current palette is also produced.

Usage:

```
gdstate [device] [frame]
```

Parameters:

COMMENT = LITERAL (Write)
The comment of the current picture. Up to 132 characters will be written.

DESCRIBE = _LOGICAL (Read)
If `TRUE`, a detailed description is displayed of the co-ordinate Frame in which the picture bounds are reported (see Parameter FRAME). [current value]

DEVICE = DEVICE (Read)
Name of the graphics device about which information is required. [Current graphics device]

EPOCH = _DOUBLE (Read)
If a 'Sky Co-ordinate System' specification is supplied (using Parameter FRAME) for a celestial co-ordinate system, then an epoch value is needed to qualify it. This is the epoch at which the displayed sky co-ordinates 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.

FRAME = LITERAL (Read)
A string determining the co-ordinate Frame in which the bounds of the current picture are to be reported. When a picture is created by an application such as PICDEF, DISPLAY, the WCS information describing the available co-ordinate systems are stored with the picture in the graphics database. This application can report bounds in any of the co-ordinate Frames stored with the current picture. The string supplied for FRAME can be one of the following:

- A domain name such as SKY, AXIS, PIXEL, NDC, BASEPIC, CURPIC. The special domain AGI_WORLD is used to refer to the world co-ordinate system stored in the AGI graphics database. This can be useful if no WCS information was store with the picture when it was created.
- An integer value giving the index of the required Frame.

- An IRAS90 *Sky Co-ordinate System* (SCS) values such as `"EQUAT(J2000)"` (see SUN/163).

If a null value (`!`) is supplied, bounds are reported in the co-ordinate Frame which was current when the picture was created. `[!]`

OUTLINE = _LOGICAL (Read)
If OUTLINE is `TRUE`, then an outline will be drawn around the current picture to indicate its position. `[FALSE]`

REPORT = _LOGICAL (Read)
If this is `FALSE`, the state of the graphics device is not reported, merely the results are written to the output parameters. It is intended for use within procedures. `[TRUE]`

STYLE = GROUP (Read)
A group of attribute settings describing the plotting style to use when drawing the outline (see Parameter OUTLINE). The format of the axis values reported on the screen may also be controlled.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the outline is controlled by the attributes Colour(Border), Width(Border), *etc.* (the synonym Outline may be used in place of Border). In addition, the following attributes may be set in order to control the appearance of the formatted axis values reported on the screen: Format, Digits, Symbol, Unit. These may be suffixed with an axis number (*e.g.* Digits(2)) to refer to the values displayed for a specific axis. [current value]

Results Parameters:

DOMAIN = LITERAL (Write)
The Domain name of the current co-ordinate Frame for the current picture.

LABEL = LITERAL (Write)
The label of the current picture. It is blank if there is no label.

NAME = LITERAL (Write)
The name of the current picture.

REFNAM = LITERAL (Write)
The reference object associated with the current picture. It is blank if there is no reference object. Up to 132 characters will be written.

X1 = LITERAL (Write)
The lowest value found within the current picture for Axis 1 of the requested co-ordinate Frame (see Parameter FRAME).

X2 = LITERAL (Write)
The highest value found within the current picture for Axis 1 of the requested co-ordinate Frame (see Parameter FRAME).

Y1 = LITERAL (Write)
The lowest value found within the current picture for Axis 2 of the requested co-ordinate Frame (see Parameter FRAME).

Y2 = LITERAL (Write)
The highest value found within the current picture for Axis 2 of the requested co-ordinate Frame (see Parameter FRAME).

Examples:

```
gdstate
```

Shows the status of the current graphics device. The bounds of the picture are displayed in the current co-ordinate Frame of the picture.

```
gdstate ps_l basepic
```

Shows the status of the ps_l device. The bounds of the picture are displayed in the BASEPIC Frame (normalised device co-ordinates in which the short of the two dimensions of the display surface has length 1.0).

```
gdstate outline frame=pixel style="'colour=red,width=3'"
```

Shows the status of the current graphics device and draws a thick, red outline around the current database picture. The bounds of the picture are displayed in the PIXEL co-ordinate Frame (if available).

```
gdstate refnam=(ndfname)
```

Shows the status of the current graphics device. If there is a reference data object, its name is written to the ICL variable NDFNAME.

```
gdstate x1=(x1) x2=(x2) y1=(y1) y2=(y2) frame=basepic
```

Shows the status of the current graphics device. The bounds of the current picture in normalised device co-ordinates are written to the ICL variables: X1, X2, Y1, Y2.

Notes:

- The displayed bounds are the extreme axis values found anywhere within the current picture. In some situations these extreme values may not occur on the edges of the picture. For instance, if the current picture represents a region including the north celestial pole, then displaying the picture bounds in celestial co-ordinates will give a declination upper limit of +90 degrees, whilst the RA limits will be 0 hours and (close to) 24 hours.
- Previous versions of this application reported bounds in 'Normalised Device Co-ordinates' (see Section 11.3). Similar functionality is now provided by setting Parameter FRAME to `"BASEPIC"`. Be aware though, that Normalised Device Co-ordinates were normalised so that the longer of the two axes had a length of 1.0, but BASEPIC co-ordinates are normalised so that the shorter of the two axes has length 1.0.
- The 'NDC' Frame is now a normalized co-ordinate system in which each axis of the graphics device has unit length.

Related Applications :

KAPPA: GDSET, GDCLEAR.

GLITCH
Replaces bad pixels in a two-dimensional NDF with the local median

Description:

This routine removes bad pixels from a two-dimensional NDF , replacing them with the median of the eight (or less at the edges) neighbouring pixels. At least three of these eight neighbouring pixels must have good values (that is, they must not set to the bad value) otherwise the resultant pixel becomes bad.

The positions of the pixels to be removed can be supplied in four ways (see Parameter MODE):

- In response to parameter prompts. A single bad pixel position is supplied at each prompt, and the user is re-prompted until a null value is supplied.
- Within a positions list such as produced by applications CURSOR, LISTMAKE.
- Within a simple text file. Each line contains the position of a pixel to be replaced.
- Alternatively, each bad pixel in the input NDF can be used (subject to the above requirement that at least three out of the eight neighbouring pixels are not bad).

Usage:

```
                          { incat=?  }
glitch in out [title]     { infile=? }
                          { pixpos=? }
                        mode
```

Parameters:

IN = NDF (Read)
The input image.

INCAT = FILENAME (Read)
A catalogue containing a positions list giving the pixels to be replaced, such as produced by applications CURSOR, LISTMAKE. Only accessed if Parameter MODE is given the value `"Catalogue"`.

INFILE = FILENAME (Read)
The name of a text file containing the positions of the pixels to be replaced. The positions should be given in the current co-ordinate Frame of the input NDF, one per line. Spaces or commas can be used as delimiters between axis values. The file may contain comment lines with the first character `#` or `!`. This parameter is only used if Parameter MODE is set to `"File"`.

MODE = LITERAL (Read)
The method used to obtain the positions of the pixels to be replaced. The supplied string can be one of the following options.

- `"Bad"` — The bad pixels in the input NDF are used.
- `"Catalogue"` — Positions are obtained from a positions list using Parameter INCAT.
- `"File"` — The pixel positions are read from a text file specified by Parameter INFILE.
- `"Interface"` — The position of each pixel is obtained using Parameter PIXPOS. The number of positions supplied must not exceed 200.

[current value]

OUT = NDF (Write)
The output image.

PIXPOS = LITERAL (Read)
The position of a pixel to be replaced, in the current co-ordinate Frame of the input NDF. Axis values should be separated by spaces or commas. This parameter is only used if Parameter MODE is set to `"Interface"`. If a value is supplied on the command line, then the application exits after processing the single specified pixel. Otherwise, the application loops to obtain multiple pixels to replace, until a null (`!`) value is supplied. Entering a colon (`":"`) will result in a description of the required co-ordinate Frame being displayed, followed by a prompt for a new value.

TITLE = LITERAL (Read)
Title for the output image. A null value (`!`) propagates the title from the input image to the output image. `[!]`

Examples:

```
glitch m51 cleaned mode=cat incat=badpix.FIT
```

Reads pixel positions from the positions list stored in the FITS file badpix.FIT, and replaces the corresponding pixels in the two-dimensional NDF m51 by the median of the surrounding neighbouring pixels. The cleaned image is written to cleaned.sdf.

Notes:

- If the current co-ordinate Frame of the input NDF is not PIXEL, then the supplied positions are first mapped into the PIXEL Frame before being used.

Related Applications :

KAPPA: ARDMASK, CHPIX, FILLBAD, ZAPLIN, NOMAGIC, REGIONMASK, SEGMENT, SETMAGIC; FIGARO: CSET, ICSET, NCSET, TIPPEX.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- Only single- and double-precision floating-point data can be processed directly. All integer data will be converted to floating point before being processed.

GLOBALS
Displays the values of the KAPPA global parameters

Description:

This procedure lists the meanings and values of the KAPPA global parameters. If a global parameter does not have a value, the string `"<undefined>"` is substituted where the value would have been written.

Usage:

```
globals
```

HISCOM
Adds commentary to the history of an NDF

Description:

This task allows application-independent commentary to be added to the history records of an NDF . The text may be read from a text file or obtained through a parameter.

Usage:

```
                          { file=?
hiscom ndf [mode]         {
                          { comment=?
                        mode
```

Parameters:

APPNAME = LITERAL (Read)

The application name to be recorded in the new history record. If a null value (`!`) is supplied, a default of `"HISCOM"` is used and the new history record describes the parameter values supplied when HISCOM was invoked. If a non-null value is supplied, the new history record refers to the specified application name instead of `"HISCOM"` and does not describe the HISCOM parameter values. `[!]`

COMMENT = LITERAL (Read)

A line of commentary limited to 72 characters. If the value is supplied on the command line only that line of commentary will be written into the history. Otherwise repeated prompting enables a series of commentary lines to be supplied. A null value (`!`) terminates the loop. Blank lines delimit paragraphs. Paragraph wrapping is enabled by Parameter WRAP. There is no suggested default to allow more room for entering the value.

DATE = LITERAL (Read)

The date and time to associated with the new history record. Normally, a null (`!`) value should be supplied, in which case the current UTC date and time will be used. If a value is supplied, it should be in one of the following forms.

- Gregorian Calendar Date — With the month expressed either as an integer or a three-character abbreviation, and with optional decimal places to represent a fraction of a day (`"1996-10-2"` or `"1996-Oct-2.6"` for example). If no fractional part of a day is given, the time refers to the start of the day (zero hours).
- Gregorian Date and Time — Any calendar date (as above) but with a fraction of a day expressed as hours, minutes and seconds (`"1996-Oct-2 12:13:56.985"` for example). The date and time can be separated by a space or by a `"T"` (as used by ISO 8601 format).
- Modified Julian Date — With or without decimal places (`"MJD 54321.4"` for example).
- Julian Date — With or without decimal places (`"JD 2454321.9"` for example).

`[!]`

FILE = FILENAME (Read)
Name of the text file containing the commentary. It is only accessed if MODE="File".

MODE = LITERAL (Read)
The interaction mode. The allowed values are described below.

"File" — The commentary is to be read from a text file. The formatting and layout of the text is preserved in the history unless WRAP=TRUE and there are lines longer than the width of the history records.

"Interface" — The commentary is to be supplied through a parameter. See Parameter COMMENT.

["Interface"]

NDF = (Read and Write)
The NDF for which commentary is to be added to the history.

WRAP = _LOGICAL (Read)
WRAP=TRUE requests that the paragraphs of comments are wrapped to make as much text fit on to each line of the history record as possible. WRAP=FALSE means that the commentary text beyond the width of the history records (72 characters) is lost. If a null (!) value is supplied, the value used is TRUE when MODE="Interface" and FALSE if MODE="File". [!]

Examples:

```
hiscom frame256 comment="This image has a non-uniform background"
```

This adds the comment "This image has a non-uniform background" to the history records of the NDF called frame256.

```
hiscom ndf=eso146-g14 comment="This galaxy is retarded" mode=i
```

This adds the comment "This galaxy is retarded" to the history records of the NDF called eso146-g14.

```
hiscom hh14_k file file=ircam_info.lis
```

This reads the file ircam_info.lis and places the text contained therein into the history records of the NDF called hh14_k. Any lines longer than 72 characters are truncated to that length.

```
hiscom hh14_k file file=ircam_info.lis wrap
```

As the previous example except the text in each paragraph is wrapped to a width of 72 characters within the history records.

Notes:

- A HISTORY component is created if it does not exist within the NDF. The width of the history record is 72 characters.

- An error will result if the current history update mode of the NDF is `"Disabled"`, and no commentary is written. Otherwise the commentary is written at the priority equal to the current history update mode.
- A warning messages (at the normal reporting level) is issued if lines in the text file are too long for the history record and WRAP=`FALSE`, though the first 72 characters are stored.
- The maximum line length in the file is 200 characters.
- Paragraphs should have fewer than 33 lines. Longer ones will be divided.

HISLIST
Lists NDF history records

Description:

This lists all the history records in an NDF . The reported information comprises the date, time, and application name, and optionally the history text.

Usage:

```
hislist ndf
```

Parameters:

BRIEF = _LOGICAL (Read)

This controls whether a summary or the full history information is reported.

BRIEF=TRUE requests that only the date and application name in each history record is listed. BRIEF=FALSE causes the task to report the history text in addition. [FALSE]

NDF = NDF (Read)

The NDF whose history information is to be reported.

Examples:

```
hislist vcc953
```

This lists the full history information for the NDF called vcc935. The information comprises the names of the applications and the times they were used, and the associated history text.

```
hislist vcc953 brief
```

This gives a summary of the history information for the NDF called vcc935. It comprises the names of the applications and the times they were used.

Related Applications :

KAPPA: HISCOM, HISSET, NDFTRACE.

HISSET
Sets the NDF history update mode

Description:

This task controls the level of history recording in an NDF , and can also erase the history information.

The level is called the history update mode and it is a permanent attribute of the HISTORY component of the NDF, and remains with the NDF and any NDF created therefrom until the history is erased or the update mode is modified (say by this task).

Usage:

```
hisset ndf [mode] ok=?
```

Parameters:

MODE = LITERAL (Read)

The history update mode. It can take one of the following values.

"Disabled" — No history recording is to take place.

"Erase" — Erases the history of the NDF.

"Normal" — Normal history recording is required.

"Quiet" — Only brief history information is to be recorded.

"Verbose" — The fullest-possible history information is to be recorded.

The suggested default is "Normal". ["Normal"]

NDF = (Read and Write)

The NDF whose history update mode to be modified or history information erased.

OK = _LOGICAL (Read)

This is used to confirm whether or not the history should be erased. OK=TRUE lets the history records be erased; if OK=FALSE the history is retained and a message will be issued to this effect.

Examples:

```
hisset final
```

This sets the history-recording level to be normal for the NDF called final.

```
hisset final erase ok
```

This erases the history information from the NDF called final.

```
hisset mode=disabled ndf=spectrum
```

This disables history recording in the NDF called spectrum.

```
hisset test42 v
```

This sets the history-recording level to be verbose for the NDF called test42 so that the fullest-possible history is included.

```
hisset ndf=test42 mode=q
```

This sets the history-recording level to be quiet for the NDF called test42, so that only brief information is recorded.

Notes:

- A HISTORY component is created if it does not exist within the NDF, except for MODE="Erase".
- The task records the new history update mode within the history records, even if MODE="Disabled" provided the mode has changed. Thus the history information will show where there may be gaps in the recording.

Related Applications :

KAPPA: HISCOM, HISLIST, NDFTRACE.

HISTAT
Computes ordered statistics for an NDF's pixels using an histogram

Description:

This application computes and displays simple ordered statistics for the pixels in an NDF's data, quality, error, or variance array. The statistics available are:

- the pixel sum,
- the pixel mean,
- the pixel median,
- the pixel mode,
- the pixel value at selected percentiles,
- the value and position of the minimum- and maximum-valued pixels,
- the total number of pixels in the NDF,
- the number of pixels used in the statistics, and
- the number of pixels omitted.

The mode may be obtained in different ways (see Parameter METHOD).

Usage:

```
histat ndf [comp] [percentiles] [logfile]
```

Parameters:

COMP = LITERAL (Read)

The name of the NDF array component for which statistics are required. The options are limited to the arrays within the supplied NDF. In general the value may `"Data"`, `"Error"`, `"Quality"` or `"Variance"` (note that `"Error"` is the alternative to `"Variance"` and causes the square root of the variance values to be taken before computing the statistics). If `"Quality"` is specified, then the quality values are treated as numerical values (in the range 0 to 255). `["Data"]`

LOGFILE = FILENAME (Write)

A text file into which the results should be logged. If a null value is supplied (the default), then no logging of results will take place. `[!]`

METHOD = LITERAL (Read)

The method used to evaluate the mode. The choices are as follows.

- `"Histogram"` — This finds the peak of an optimally binned histogram, the mode being the central value of that bin. The number of bins may be altered given through Parameter NUMBIN, however it is recommended to use the optimal binsize derived from the prescription of Freedman & Diatonis.

- "Moments" — As "Histogram" but the mode is the weighted centroid from the moments of the peak bin and its neighbours. The neighbours are those bins either side of the peak in a continuous sequence whose membership exceeds the peak value less three times the Poisson error of the peak bin. Thus it gives an interpolated mode and does reduce the effect of noise.
- "Pearson" — This uses the 3 * median - 2 * mean formula devised by Pearson. See the first two References. This assumes that the median is bracketed by the mode and mean and only a mildly skew unimodal distribution. This often applies to an image of the sky.

["Pearson"]

NDF = NDF (Read)
The NDF data structure to be analysed.

NUMBIN = _INTEGER (Read)
The number of histogram bins to be used for the coarse histogram to evaluate the mode. It is only accessed when METHOD="Histogram" or "Moments". This must lie in the range 10 to 10000. The suggested default is calculated dynamically depending on the data spread and number of values (using the prescription of Freedman & Diaconis). For integer data it is advisble to use the dynamic default or an integer multiple thereof to avoid creating non-integer wide bins. []

PERCENTILES(100) = _REAL (Read)
A list of percentiles to be found. None are computed if this parameter is null (!). The percentiles must be in the range 0.0 to 100.0 [!]

Results Parameters:

MAXCOORD( ) = _DOUBLE (Write)
A one-dimensional array of values giving the WCS co-ordinates of the centre of the (first) maximum-valued pixel found in the NDF array. The number of co-ordinates is equal to the number of NDF dimensions.

MAXIMUM = _DOUBLE (Write)
The maximum pixel value found in the NDF array.

MAXPOS( ) = _INTEGER (Write)
A one-dimensional array of pixel indices identifying the (first) maximum-valued pixel found in the NDF array. The number of indices is equal to the number of NDF dimensions.

MAXWCS = LITERAL (Write)
The formatted WCS co-ordinates at the maximum pixel value. The individual axis values are comma separated.

MEAN = _DOUBLE (Write)
The mean value of all the valid pixels in the NDF array.

MEDIAN = _DOUBLE (Write)
The median value of all the valid pixels in the NDF array.

MINCOORD( ) = _DOUBLE (Write)
A one-dimensional array of values giving the user co-ordinates of the centre of the (first) minimum-valued pixel found in the NDF array. The number of co-ordinates is equal to the number of NDF dimensions.

MINIMUM = _DOUBLE (Write)
The minimum pixel value found in the NDF array.

MINPOS( ) = _INTEGER (Write)
A one-dimensional array of pixel indices identifying the (first) minimum-valued pixel found in the NDF array. The number of indices is equal to the number of NDF dimensions.

MINWCS = LITERAL (Write)
The formatted WCS co-ordinates at the minimum pixel value. The individual axis values are comma separated.

MODE = _DOUBLE (Write)
The modal value of all the valid pixels in the NDF array. The method used to obtain the mode is governed by Parameter METHOD.

NUMBAD = _INTEGER (Write)
The number of pixels which were either not valid or were rejected from the statistics during iterative κ-sigma clipping.

NUMGOOD = _INTEGER (Write)
The number of NDF pixels which actually contributed to the computed statistics.

NUMPIX = _INTEGER (Write)
The total number of pixels in the NDF (both good and bad).

PERVAL() = _DOUBLE (Write)
The values of the percentiles of the good pixels in the NDF array. This parameter is only written when one or more percentiles have been requested.

TOTAL = _DOUBLE (Write)
The sum of the pixel values in the NDF array.

Examples:

```
histat image
```

Computes and displays simple ordered statistics for the data array in the NDF called image.

```
histat image method=his
```

As above but the mode is the centre of peak bin in the optimally distributed histogram rather than sub-bin interpolated using neighbouring bins.

```
histat ndf=spectrum variance
```

Computes and displays simple ordered statistics for the variance array in the NDF called spectrum.

```
histat spectrum error
```

Computes and displays ordered statistics for the variance array in the NDF called spectrum, but takes the square root of the variance values before doing so.

```
histat halley logfile=stats.dat method=pearson
```

Computes ordered statistics for the data array in the NDF called halley, and writes the results to a logfile called `stats.dat`. The mode is derived using the Pearson formula.

```
histat ngc1333 percentiles=[0.25,0.75]
```

Computes ordered statistics for the data array in the NDF called ngc1333, including the quartile values.

Notes:

- Where the histogram contains a few extreme outliers, the histogram limits are adjusted to reduce greatly the bias upon the statistics, even if a chosen percentile corresponds to an extreme outlier. The outliers are still accounted in the median and percentiles. The histogram normally uses 10000 bins. For small arrays the number of bins is at most a half of the number of array elements. Integer arrays have a minimum bin width of one; this can also reduce the number of bins. The goal is to avoid most histogram bins being empty artificially, since the sparseness of the histogram is the main criterion for detecting outliers. Outliers can also be removed (flagged) via application THRESH prior to using this application.
- There is quantisation bias in the statistics, but for non-pathological distributions this should be insignificant. Accuracy to better than 0.01 of a percentile is normal. Linear interpolation within a bin is used, so the largest errors arise near the median.

References :

Moroney, M.J., 1957, *Facts from Figures* (Pelican)

Goad, L.E. 1980, *Statistical Filtering of Cosmic-Ray Events from Astronomical CCD Images* in *Applications of Digital Image Processing to Astronomy*, SPIE **264**, 136.

Freedman, D. & Diaconis, P. 1981, *On the histogram as a density estimator: L2 theory*, Zeitschrift f"ur Wahrscheinlichkeitstheorie und verwandte Gebiete **57**, 453.

Related Applications :

KAPPA: HISTOGRAM, MSTATS, NDFTRACE, NUMB, STATS; ESP: HISTPEAK; FIGARO: ISTAT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, QUALITY, TITLE, and HISTORY components of the NDF.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single- or double-precision floating point, as appropriate.
- Any number of NDF dimensions is supported.

HISTEQ
Performs an histogram equalisation on an NDF

Description:

This application transforms an NDF via histogram equalisation. Histogram equalisation is an image-processing technique in which the distribution (between limits) of data values in the input array is adjusted so that in the output array there are approximately equal numbers of elements in each histogram bin. To achieve this the histogram bin size is no longer a constant. This technique is commonly known as histogram equalisation. It is useful for displaying features across a wide dynamic range, sometimes called a maximum-information picture. The transformed array is output to a new NDF.

Usage:

```
histeq in out [numbin]
```

Parameters:

IN = NDF (Read)
The NDF structure to be transformed.

NUMBIN = _INTEGER (Read)
The number of histogram bins to be used. This should be a large number, say 2000, to reduce quantisation errors. It must be in the range 100 to 10000. [2048]

OUT = NDF (Write)
The NDF structure to contain the transformed data array.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (!) propagates the title from the input NDF to the output NDF. [!]

Examples:

```
histeq halley maxinf
```

The data array in the NDF called halley is remapped via histogram equalisation to form the new NDF called maxinf.

```
histeq halley maxinf 10000 title="Maximum information of Halley"
```

The data array in the NDF called halley is remapped via histogram equalisation to form the new NDF called maxinf. Ten thousand bins in the histogram are required rather than the default of 2048. The title of NDF maxinf is "Maximum information of Halley".

Notes:

If there are a few outliers in the data and most of the points concentrated about a value it may be wise to truncate the data array via THRESH, or have a large number of histogram bins.

Related Applications :

KAPPA: LAPLACE, LUTABLE, LUTEDIT, SHADOW, THRESH; FIGARO: HOPT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, WCS, and HISTORY components of an NDF data structure and propagates all extensions. UNITS and VARIANCE become undefined by the transformation, and so are not propagated.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

HISTOGRAM
Computes an histogram of an NDF's values

Description:

This application derives histogram information for an NDF array between specified limits, using either a set number of bins (Parameter NUMBIN) or a chosen bin width (Parameter WIDTH). The histogram is reported, and may optionally be written to a text log file, and/or plotted graphically.

By default, each data value contributes a value of one to the corresponding histogram bin, but alternative weights may be supplied via Parameter WEIGHTS.

Usage:

```
histogram in numbin range [comp] [logfile]
```

Parameters:

AXES = _LOGICAL (Read)

TRUE if labelled and annotated axes are to be drawn around the plot. The width of the margins left for the annotation may be controlled using Parameter MARGIN. The appearance of the axes (colours, founts, *etc.*) can be controlled using the Parameter STYLE. The dynamic default is TRUE if CLEAR is TRUE, and FALSE otherwise. []

CLEAR = _LOGICAL (Read)

If TRUE the current picture is cleared before the plot is drawn. If CLEAR is FALSE not only is the existing plot retained, but also an attempt is made to align the new picture with the existing picture. Thus you can generate a composite plot within a single set of axes, say using different colours or modes to distinguish data from different datasets. [TRUE]

COMP = LITERAL (Read)

The name of the NDF array component to have its histogram computed: "Data", "Error", "Quality" or "Variance" (where "Error" is the alternative to "Variance" and causes the square root of the variance values to be taken before computing the statistics). If "Quality" is specified, then the quality values are treated as numerical values (in the range 0 to 255). ["Data"]

CUMUL = _LOGICAL (Read)

If TRUE then a cumulative histogram is reported. [FALSE]

DEVICE = DEVICE (Read)

The graphics workstation on which to produce the plot. If it is null (!), no plot will be made. [Current graphics device]

IN = NDF (Read)

The NDF data structure to be analysed.

LOGFILE = FILENAME (Write)

A text file into which the results should be logged. If a null value is supplied (the default), then no logging of results will take place. [!]

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. If a null (`!`) value is supplied, the value used is `0.15` (for all edges) if either annotated axes or a key are produced, and zero otherwise. [current value]

NUMBIN = _INTEGER (Read)
The number of histogram bins to be used. This must lie in the range 2 to 10000. The suggested default is the current value. It is ignored if WIDTH is not null.

OUT = NDF (Read)
Name of the NDF structure to save the histogram in its data array. If null (`!`) is entered the histogram NDF is not created. [`!`]

RANGE = LITERAL (Read)
RANGE specifies the range of values for which the histogram is to be computed. The supplied string should consist of up to three sub-strings, separated by commas. For all but the option where you give explicit numerical limits, the first sub-string must specify the method to use. If supplied, the other two sub-strings should be numerical values as described below (default values will be used if these sub-strings are not provided). The following options are available.

- lower,upper — You can supply explicit lower and upper limiting values. For example, `"10,200"` would set the histogram lower limit to 10 and its upper limit to 200. No method name prefixes the two values. If only one value is supplied, the `"Range"` method is adopted. The limits must be within the dynamic range for the data type of the NDF array component.
- `"Percentiles"` — The default values for the histogram data range are set to the specified percentiles of the data. For instance, if the value `"Per,10,99"` is supplied, then the lowest 10% and highest 1% of the data values are excluded from the histogram. If only one value, $p1$, is supplied, the second value, $p2$, defaults to (100 - $p1$). If no values are supplied, the values default to `"5,95"`. Values must be in the range 0 to 100.
- `"Range"` — The minimum and maximum array values are used. No other sub-strings are needed by this option. Null (`!`) is a synonym for the `"Range"` method.
- `"Sigmas"` — The histogram limiting values are set to the specified numbers of standard deviations below and above the mean of the data. For instance, if the supplied value is `"sig,1.5,3.0"`, then the histogram extends from the mean of the data minus 1.5 standard deviations to the mean plus 3 standard deviations. If only one value is supplied, the second value defaults to the supplied value. If no values are supplied, both default to `"3.0"`.

The `"Percentiles"` and `"Sigmas"` methods are useful to generate a first pass at the histogram. They reduce the likelihood that all but a small number of values lie within a few histogram bins.

The extreme values are reported unless Parameter RANGE is specified on the command line. In this case extreme values are only calculated where necessary for the chosen method.

The method name can be abbreviated to a single character, and is case insensitive. The initial value is `"Range"`. The suggested defaults are the current values, or `!` if these do not exist. [current value]

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use when drawing the annotated axes and data values.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the histogram curve is controlled by the attributes Colour(Curves), Width(Curves), *etc*. (The synonym Line may be used in place of Curves.) [current value]

TITLE = LITERAL (Read)

Title for the histogram NDF. `["KAPPA - Histogram"]`

WEIGHTS = NDF (Read)

An optional NDF holding weights associated with each input pixel value (supplied via Parameter IN). Together with Parameter WEIGHTSTEP, these determine the count added to the corresponding histogram bin for each input pixel value. For instance, weights could be related to the variance of the data values, or to the position of the data values within the input NDF. If a null value (`!`) is supplied for WEIGHTS, all input values contribute a count of one to the corresponding histogram bin. If an NDF is supplied, the histogram count for a particular input pixel is formed by dividing its weight value (supplied in the WEIGHTS NDF) by the value of Parameter WEIGHTSTEP, and then taking the nearest integer. Input pixels with bad or zero weights are excluded from the histogram. `[!]`

WEIGHTSTEP = _DOUBLE (Read)

Only accessed if a value is supplied for Parameter WEIGHTS. WEIGHTSTEP is the increment in weight value that corresponds to a unit increment in histogram count.

WIDTH = _DOUBLE (Read)

The bin width. This is the alternative to setting the number of bins. The bins of the chosen width start from the minimum value and do not exceed the maximum value. Values are constrained to give between 2 and 10000 bins. If this parameter is set to

null (!), the data range and Parameter NUMBIN are used to specify the bin width. [!]

XLEFT = _DOUBLE (Read)
The axis value to place at the left hand end of the horizontal axis of the plot. If a null (!) value is supplied, the minimum data value in the histogram is used. The value supplied may be greater than or less than the value supplied for XRIGHT. [!]

XLOG = _LOGICAL (Read)
TRUE if the plot *x* axis is to be logarithmic. Any histogram bins which have negative or zero central data values are omitted from the plot. [FALSE]

XRIGHT = _DOUBLE (Read)
The axis value to place at the right hand end of the horizontal axis of the plot. If a null (!) value is supplied, the maximum data value in the histogram is used. 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 of the plot. If a null (!) value is supplied, the lowest count the histogram is used. The value supplied may be greater than or less than the value supplied for YTOP. [!]

YLOG = _LOGICAL (Read)
TRUE if the plot *y* axis is to be logarithmic. Empty bins are removed from the plot if the *y* axis is logarithmic. [FALSE]

YTOP = _DOUBLE (Read)
The axis value to place at the top end of the vertical axis of the plot. If a null (!) value is supplied, the largest count in the histogram is used. The value supplied may be greater than or less than the value supplied for YBOT. [!]

Examples:

```
histogram image 100 !  device=!
```

Computes and reports the histogram for the data array in the NDF called image. The histogram has 100 bins and spans the full range of data values.

```
histogram ndf=spectrum comp=variance range="100,200" numbin=20
```

Computes and reports the histogram for the variance array in the NDF called spectrum. The histogram has 20 bins and spans the values between 100 and 200. A plot is made to the current graphics device.

```
histogram ndf=spectrum comp=variance range="100,204" width=5
```

This behaves the same as the previous example, even though it specifies a larger maximum, as the same number of width=5 bins are used.

```
histogram cube(3,4,) 10 si out=c3_4_hist device=!
```

Computes and reports the histogram for the *z*-vector at (*x*,*y*) element (3,4) of the data array in the three-dimensional NDF called cube. The histogram has 10 bins and

spans a range three standard deviations either side of the mean of the data values. The histogram is written to a one-dimensional NDF called c3_4_hist.

```
histogram cube numbin=32 !  device=xwindows style="title=cube"
```

Computes and reports the histogram for the data array in the NDF called cube. The histogram has 32 bins and spans the full range of data values. A plot of the histogram is made to the XWINDOWS device, and is titled `"cube"`.

```
histogram cube numbin=32 !  device=xwindows ylog style=^style.dat
```

As in the previous example except the logarithm of the number in each histogram bin is plotted, and the contents of the text file `style.dat` control the style of the resulting graph. The plotting style specified in file `style.dat` becomes the default plotting style for future invocations of HISTOGRAM.

```
histogram cube numbin=32 !  device=xwindows ylog tempstyle=^style.dat
```

This is the same as the previous example, except that the style specified in file `style.dat` does not become the default style for future invocations of HISTOGRAM.

```
histogram halley(~200,~300) "pe,10,90" logfile=hist.dat \
```

Computes the histogram for the central 200 by 300 elements of the data array in the NDF called halley, and writes the results to a logfile called `hist.dat`. The histogram uses the current number of bins, and includes data values between the 10 and 90 percentiles. A plot appears on the current graphics device.

Related Applications :

KAPPA: HISTAT, MSTATS, NUMB, STATS; FIGARO: HIST, ISTAT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, QUALITY, LABEL, TITLE, UNITS, and HISTORY components of the input NDF.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

INTERLEAVE
Forms a higher-resolution NDF by interleaving a set of NDFs

Description:

This routine performs interleaving, also known as interlacing, in order to restore resolution where the pixel dimension undersamples data. Resolution may be improved by integer factors along one or more dimensions. For an N-fold increase in resolution along a dimension, INTERLEAVE demands N NDF structures that are displaced from each other by i/N pixels, where i is an integer from 1 to $N-1$. It creates an NDF whose dimensions are enlarged by N along that dimension.

The supplied NDFs should have the same dimensionality.

Usage:

```
interleave in out expand
```

Parameters:

EXPAND() = _INTEGER (Read)

Linear expansion factors to be used to create the new data array. The number of factors should equal the number of dimensions in the input NDF. If fewer are supplied the last value in the list of expansion factors is given to the remaining dimensions. Thus if a uniform expansion is required in all dimensions, just one value need be entered. If the net expansion is one, an error results. The suggested default is the current value.

FILL = LITERAL (Read)

Specifies the value to use where the interleaving does not fill the array, say because the shapes of the input NDFs are not the same, or have additional shifts of origin. Allowed values are `"Bad"` or `"Zero"`. `["Bad"]`

IN = NDF (Read)

A group of input NDFs to be interweaved. They may have different shapes, but must all have the same number of dimensions. This should be given as a comma-separated list, in which each list element can be:

- an NDF name, optionally containing wild-cards and/or regular expressions (`"*"`, `"?"`, `"[a-z]"` *etc.*).
- the name of a text file, preceded by an up-arrow character `"^"`. Each line in the text file should contain a comma-separated list of elements, each of which can in turn be an NDF name (with optional wild-cards, *etc.*), or another file specification (preceded by a caret). Comments can be included in the file by commencing lines with a hash character `"#"`.

If the value supplied for this parameter ends with a hyphen `"-"`, then you are re-prompted for further input until a value is given which does not end with a hyphen. All the datasets given in this way are concatenated into a single group.

OUT = NDF (Write)

Output NDF structure.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (!) propagates the title from the input NDF to the output NDF. [!]

TRIM = _LOGICAL (Read)
This parameter controls the shape of the output NDF before the application of the expansion. If TRIM=TRUE, then the output NDF reflects the shape of the intersection of all the input NDFs, *i.e.* only pixels which appear in all the input arrays will be represented in the output. If TRIM=FALSE, the output reflects shape of the union of the inputs, *i.e.* every pixel which appears in the input arrays will be represented in the output. [TRUE]

Examples:

```
interleave "vector1,vector2" weave 2
```

This interleaves the one-dimensional NDFs called vector1 and vector2 and stores the result in NDF weave. Only the intersection of the two input NDFs is used.

```
interleave 'image*' weave [3,2] title="Interlaced image"
```

This interleaves the two-dimensional NDFs with names beginning with "image" into an NDF called weave. The interleaving has three datasets along the first dimension and two along the second. Therefore there should be six input NDFs. The output NDF has title "Interlaced image".

```
interleave in='image*' out=weave expand=[3,2] notrim
```

As above except the title is not set and the union of the bounds of the input NDFs is expanded to form the shape of the weave NDF.

```
interleave ^frames.lis finer 2
```

This interleaves the NDFs listed in the text file frames.lis to form an enlarged NDF called finer. The interleaving is twofold along each axis of those NDFs.

Related Applications :

KAPPA: PIXDUPE; CCDPACK: DRIZZLE.

Implementation Status:

- This routine processes the AXIS, DATA, QUALITY, and VARIANCE from the all input NDF data structures. It also processes the WCS, LABEL, TITLE, UNITS, and HISTORY components of the primary NDF data structure, and propagates all of its extensions.
- The AXIS centre values along each axis are formed by interleaving the corresponding centres from the first NDF, and linearly interpolating between those to complete the array.

- The AXIS width and variance values in the output are formed by interleaving the corresponding input AXIS values. Each array element is assigned from the first applicable NDF. For example, for a two-dimensional array with expansion factors of 2 and 3 respectively, the first two NDFs would be used to define the array elements for the first axis. The second axis's elements come from the first, third, and fifth NDFs.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

KAPHELP
Gives help about KAPPA

Description:

Displays help about KAPPA. The help information has classified and alphabetical lists of commands, general information about KAPPA and related material; it describes individual commands in detail.

Here are some of the main options.

`kaphelp`
: No parameter is given so the introduction and the top-level help index is displayed.

`kaphelp application/topic`
: This gives help about the specified application or topic.

`kaphelp application/topic subtopic`
: This lists help about a subtopic of the specified application or topic. The hierarchy of topics has a maximum of four levels.

`kaphelp Hints`
: This gives hints for new and intermediate users.

`kaphelp summary`
: This shows a one-line summary of each application.

`kaphelp classified classification`
: This lists a one-line summary of each application in the given functionality classification.

See the Section `"Navigating the Help Library"` for details how to move around the help information, and to select the topics you want to view.

Usage:

```
kaphelp [topic] [subtopic] [subsubtopic] [subsubsubtopic]
```

Parameters:

TOPIC = LITERAL (Read)
: Topic for which help is to be given. `[" "]`

SUBTOPIC = LITERAL (Read)
: Subtopic for which help is to be given. `[" "]`

SUBSUBTOPIC = LITERAL (Read)
: Subsubtopic for which help is to be given. `[" "]`

SUBSUBSUBTOPIC = LITERAL (Read)
: Subsubsubtopic for which help is to be given. `[" "]`

Navigating the Help Library :

The help information is arranged hierarchically. You can move around the help information whenever KAPHELP prompts. This occurs when it has either presented a screen's worth of

text or has completed displaying the previously requested help. The information displayed by KAPHELP on a particular topic includes a description of the topic and a list of subtopics that further describe the topic.

At a prompt you may enter:

- a topic and/or subtopic name(s) to display the help for that topic or subtopic, so for example, `block parameters box` gives help on `BOX`, which is a subtopic of `Parameters`, which in turn is a subtopic of `BLOCK`;
- a <CR> to see more text at a `Press RETURN to continue ...` request;
- a <CR> at topic and subtopic prompts to move up one level in the hierarchy, and if you are at the top level it will terminate the help session;
- a `CTRL/D` (pressing the CTRL and D keys simultaneously) in response to any prompt will terminate the help session;
- a question mark `?` to redisplay the text for the current topic, including the list of topic or subtopic names; or
- an ellipsis `...` to display all the text below the current point in the hierarchy. For example, `BLOCK...` displays information on the BLOCK topic as well as information on all the subtopics under BLOCK.

You can abbreviate any topic or subtopic using the following rules.

- Just give the first few characters, *e.g.* `PARA` for `Parameters`.
- Some topics are composed of several words separated by underscores. Each word of the keyword may be abbreviated, *e.g.* `Colour_Set` can be shortened to `C_S`.
- The characters `%` and `*` act as wildcards, where the percent sign matches any single character, and asterisk matches any sequence of characters. Thus to display information on all available topics, type an asterisk in reply to a prompt.
- If a word contains, but does end with an asterisk wildcard, it must not be truncated.
- The entered string must not contain leading or embedded spaces.

Ambiguous abbreviations result in all matches being displayed.

Implementation Status:

- Uses the portable help system.

KAPVERSION
Checks the package version number

Description:

This application will display the installed package version number, or compare the version number of the installed package against a specified version number, reporting whether the installed package is older, or younger, or equal to the specified version.

Usage:

```
kapversion [compare]
```

Parameters:

COMPARE = LITERAL (Read)

A string specifying the version number to be compared to the version of the installed package. If a null (`!`) value is supplied, the version string of the installed package is displayed, but no comparison takes place. If a non-null value is supplied, the version of the installed package is not displayed.

The supplied string should be in the `"V<ddd>.<ddd>-<ddd>"`, where `"<ddd>"` represents a set of digits. The leading `"V"` can be omitted, as can any number of trailing fields (missing trailing fields default to zero). `[!]`

Results Parameters:

RESULT = _INTEGER (Write)

If a value is given for the COMPARE parameter, then RESULT is set to one of the following values:

- `1` — The installed package is older than the version number specified by the COMPARE parameter.
- `0` — The version of the installed package is equal to the version specified by the COMPARE parameter.
- `-1` — The installed package is younger than the version number specified by the COMPARE parameter.

The same value is also written to standard output.

Examples:

```
kapversion
```

Displays the version number of the installed package.

```
kapversion compare="V0.14-1"
```

Compares the version of the installed package with the version `"V0.14-1"`, and sets the RESULT parameter appropriately. For instance, if the installed package was

"V0.13-6" then RESULT would be set to -1. If the installed package was "V0.14-1", RESULT would be set to 0. If the installed package was "V0.14-5" RESULT would be set to $+1$.

Notes:

- The package version number is obtained from the `version` file in the directory containing the package's installed executable files. This file is created when the package is installed using the "`mk install`" command. An error will be reported if this file cannot be found.

KSTEST
Compares data sets using the Kolmogorov-Smirnov test

Description:

This routine reads in a data array and performs a two sided Kolmogorov-Smirnov test on the vectorised data. It does this in two ways:

(1) If only one dataset is to be tested the data array is divided into subsamples. First it compares subsample 1 with subsample 2, if they are thought to be from the same sample they are concatenated. This enlarged sample is then compared with subsample 3 *etc.*, concatenating if consistent, until no more subsamples remain.

(2) If more than one dataset is specified, the datasets are compared to the reference dataset in turn. If the probability the two are from the same sample is greater than the specified confidence level, the datasets are concatenated, and the next sample is tested against this enlarged reference dataset.

The probability and maximum separation of the cumulative distribution function is written for each comparison (at the normal reporting level). The mean value of the consistent data and its error are also reported. In all cases the consistent data can be output to a new dataset. The statistics and probabilities are written to results parameters.

Usage:

```
kstest in out [limit]
```

Parameters:

COMP = LITERAL (Read)

The name of the NDF array component to be tested for consistency: `"Data"`, `"Error"`, `"Quality"` or `"Variance"` (where `"Error"` is the alternative to `"Variance"` and causes the square root of the variance values to be taken before performing the comparisons). If `"Quality"` is specified, then the quality values are treated as numerical values (in the range 0 to 255). `["Data"]`

LIMIT = _REAL (Read)

Confidence level at which samples are thought to be consistent. This must lie in the range 0 to 1. `[0.05]`

IN = LITERAL (Read)

The names of the NDFs to be tested. If just one dataset is supplied, it is divided into subsamples, which are compared (see Parameter NSAMPLE). When more than one dataset is provided, the first becomes the reference dataset to which all the remainder are compared.

It may be a list of NDF names or direction specifications separated by commas. If a list is supplied on the command line, the list must be enclosed in double quotes. NDF names may include the regular expressions (`"*"`, `"?"`, `"[a-z]"` *etc.*). Indirection may occur through text files (nested up to seven deep). The indirection character is `"^"`. If extra prompt lines are required, append the continuation character `"-"` to the end of the line. Comments in the indirection file begin with the character `"#"`.

NSAMPLE = _INTEGER (Read)
The number of the subsamples into which to divide the reference dataset. This parameter is only requested when a single NDF is to be analysed, *i.e.* when only one dataset name is supplied via Parameter IN. The allowed range is 2 to 20. [3]

OUT = NDF (Write)
Output one-dimensional NDF to which the consistent data are written. A null value (!)—the suggested default—prevents creation of this output dataset.

Results Parameters:

DIST() = _REAL (Write)
Maximum separation found in the cumulative distributions for each comparison subsample. Note that it excludes the reference dataset.

ERRMEAN = _DOUBLE (Write)
Error in the mean value of the consistent data.

FILES() = LITERAL (Write)
The names of the datasets intercompared. The first is the reference dataset.

MEAN = _DOUBLE (Write)
Mean value of the consistent data.

NKEPT = _INTEGER (Write)
Number of consistent data.

PROB() = _REAL (Write)
Probability that each comparison subsample is drawn from the same sample. Note that this excludes the reference sample.

SIGMA = _DOUBLE (Write)
Standard deviation of the consistent data.

Examples:

```
kstest arlac accept
```

This tests the NDF called arlac for self-consistency at the 95% confidence level using three subsamples. No output dataset is created.

The following applies to all the examples. If the reference dataset and a comparison subsample are consistent, the two merge to form an expanded reference dataset, which is then used for the next comparison. Details of the comparisons are presented.

```
kstest arlac arlac_filt 0.10 nsample=10
```

As above except data are retained if they exceed the 90% probability level, the comparisons are made with ten subsamples, and the consistent data are written to the one-dimensional NDF called arlac_filt.

```
kstest in="ref,obs*" comp=v out=master
```

This compares the variance in the NDF called ref with that in a series of other NDFs whose names begin "obs". The variance consistent with the reference dataset are written to the data array in the NDF called master. To be consistent, they must be the same at 95% probability.

```
kstest "ref,^96lc.lis,obs*" master comp=v
```

As the previous example, except the comparison files include those listed in the text file 96lc.lis.

Notes:

- The COMP array MUST exist in each NDF to be compared. The COMP array becomes the data array in the output dataset. When COMP="Data", the variance values corresponding to consistent data are propagated to the output dataset.
- Pixel bounds are ignored for the comparisons.
- The internal comparison of a single dataset follows the method outlined in Hughes D., 1993, *JCMT-UKIRT Newsletter*, #4, p32.
- The maximum number of files is 20.

Implementation Status:

- This routine correctly processes DATA, VARIANCE, HISTORY, LABEL, TITLE, and UNITS components, and propagates all extensions. AXIS information is lost. Propagation is from the reference dataset.
- Processing of bad pixels and automatic quality masking are supported.
- All numeric data types are supported, however, processing uses the _REAL data type, and the output dataset has this type.

LAPLACE
Performs a Laplacian convolution as an edge detector in a two-dimensional NDF

Description:

This routine calculates the Laplacian of the supplied two-dimensional NDF , and subtracts it from the original array to create the output NDF. The subtractions can be done a specified integer number of times. This operation can be approximated by a convolution with the kernel:

$$\begin{matrix} N & -N & -N \\ N & +8N & -N \\ N & -N & -N \end{matrix}$$

where N is the integer number of times the Laplacian is subtracted. This convolution is used as a uni-directional edge detector. Areas where the input data array is flat become zero in the output data array.

Usage:

```
laplace in [number] out [title]
```

Parameters:

IN = NDF (Read)
Input NDF.

NUMBER = _INTEGER (Read)
Number of Laplacians to remove. [1]

OUT = NDF (Write)
Output NDF.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

Examples:

```
laplace a 10 b
```

This subtracts ten Laplacians from the NDF called a, to make the NDF called b. NDF b inherits its title from a.

Related Applications :

KAPPA: SHADOW, MEDIAN; FIGARO: ICONV3.

Implementation Status:

- This routine correctly processes the WCS, AXIS, DATA, and VARIANCE components of an NDF data structure. QUALITY is propagated.

- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

LINPLOT
Draws a line plot of the data values in a one-dimensional NDF

Description:

This application creates a plot of array value against position for a one-dimensional NDF . The vertical axis of the plot represents array value, and the horizontal axis represents position. These can be mapped in various ways on to the graphics surface (*e.g.* linearly, logarithmically); see Parameters XMAP and YMAP.

The plot may take several different forms such as a `"join-the-dots"` plot, a `"staircase"` plot, a `"chain"` plot (see Parameter MODE). Errors on both the data values and the data positions may be represented in several different ways (see Parameters ERRBAR and SHAPE). The plotting style (colour, founts, text size, *etc.*) may be specified in detail using Parameter STYLE.

The bounds of the plot on both axes can be specified using Parameters XLEFT, XRIGHT, YBOT, and YTOP. If not specified they take default values which encompass the entire supplied data set. The current picture is usually cleared before plotting the new picture, but Parameter CLEAR can be used to prevent this, allowing several plots to be 'stacked' together. If a new plot is drawn over an existing plot, then there is an option to allow the new plot to be aligned with the existing plot (see Parameter ALIGN).

The input NDF may, for instance, contain a spectrum of data values against wavelength, or it may contain data values along a one-dimensional profile through an NDF of higher dimensionality. In the latter case, the current co-ordinate Frame of the NDF may have more than one axis. Any of the axes in the current co-ordinate Frame of the input NDF may be used to annotate the horizontal axis of the plot produced by this application. Alternatively, the horizontal axis may be annotated with offset from the first array element measured within the current co-ordinate Frame of the NDF. For instance, a one-dimensional slice through a two-dimensional image calibrated in RA/DEC could be annotated with RA, or DEC, or offset from the first element (in arcminutes, degrees, *etc*). This offset is measured along the path of the profile. The choice of annotation for the horizontal axis is controlled by Parameter USEAXIS.

Usage:

```
linplot ndf [comp] [mode] [xleft] [xright] [ybot] [ytop] [device]
```

Parameters:

ALIGN = _LOGICAL (Read)

Controls whether or not the new data plot should be aligned with an existing data plot. If ALIGN is `TRUE`, the *x* axis values of the new plot will be mapped into the co-ordinate system of the *x* axis in the existing plot before being used (if this is not possible an error is reported). In this case, the XLEFT, XRIGHT, YBOT and YTOP parameters are ignored and the bounds of the existing plot are used instead. If ALIGN is `FALSE`, the new *x* axis values are used without change. The bounds of the new plot are specified using Parameters XLEFT, XRIGHT, YBOT, and YTOP as usual, and these bounds are mapped to the edges of the existing picture. The ALIGN parameter is

only accessed if Parameter CLEAR is FALSE, and if there is another line plot within the current picture. If a null (!) value is supplied, a value of TRUE will be used if and only if a mapping can be found between the existing and the new plots. A value of FALSE will be used otherwise. [!]

ALIGNSYS = LITERAL (Read)

This is only used if a TRUE value is supplied for Parameter ALIGN. It specifies the co-ordinate system in which the new plot and the existing plot are aligned (for further details see the description of the AlignSystem attribute in SUN/210.). The supplied name should be a valid co-ordinate system name for the horizontal axis (see the description of the System attribute in SUN/210 for a list of these names). It may also take the special value "Data", in which case alignment occurs in the co-ordinate system represented by the current WCS Frame in the supplied NDF. If a null (!) value is supplied. The alignment system is determined by the current value of AlignSystem attribute in the supplied NDF. ["Data"]

AXES = _LOGICAL (Read)

TRUE if labelled and annotated axes are to be drawn around the plot. If a null (!) value is supplied, the value used is FALSE if the plot is being aligned with an existing plot (see Parameter ALIGN), and TRUE otherwise. Parameter USEAXIS determines the quantity used to annotated the horizontal axis. The width of the margins left for the annotation may be controlled using Parameter MARGIN. The appearance of the axes (colours, founts, *etc.*) can be controlled using the Parameter STYLE. [!]

CLEAR = _LOGICAL (Read)

If TRUE the current picture is cleared before the plot is drawn. If CLEAR is FALSE not only is the existing plot retained, but also the previous plot can be used to specify the axis limits (see Parameter ALIGN). Thus you can generate a composite plot within a single set of axes, say using different colours or modes to distinguish data from different datasets. Note, alignment between the two plots is controlled by the AlignSystem attribute of the data being displayed. For instance, if you have an existing plot showing a spectrum plotted against radio velocity and you overlay another spectrum, also in radio velocity but with a different rest frequency, the appearance of the final plot will depend on the value of the AlignSystem attribute of the second spectrum. If AlignSystem is "Wavelen" (this is the default) then the two spectra will be aligned in wavelength, but if AlignSystem is "vrad" they will be aligned in radio velocity. There will be no difference in effect between these two forms of alignment unless the rest frequency is different in the two spectra. Likewise, the AlignStdOfRest attribute of the second spectrum controls the standard of rest in which alignment occurs. These attributes (like all other attributes) may be examined and modified using WCSATTRIB.

COMP = LITERAL (Read)

The NDF component to be plotted. It may be "Data", "Quality", "Variance", or "Error" (where "Error" is an alternative to "Variance" and causes the square root of the variance values to be displayed). If "Quality" is specified, then the quality values are treated as numerical values (in the range 0 to 255). ["Data"]

DEVICE = DEVICE (Read)

The plotting device. [Current graphics device]

ERRBAR = _LOGICAL (Read)

TRUE if error bars are to be drawn. The error bars can comprise either or both of the

data and axis-centre errors, depending on what is available in the supplied dataset. The Parameter SHAPE controls the appearance of the error bars, and XSIGMA and YSIGMA control their lengths. The ERRBAR parameter is ignored unless the Parameter COMP is set to "Data". [FALSE]

FREQ = _INTEGER (Read)

The frequency at which error bars are to be plotted. For instance, a value of 2 would mean that alternate points have error bars plotted. This lets some plots be less cluttered. FREQ must lie in the range 1 to half of the number of points to be plotted. FREQ is only accessed when Parameter ERRBAR is TRUE. [1]

KEY = _LOGICAL (Read)

TRUE if a key is to be plotted below the horizontal axis giving the positions at the start and end of the plot, within the current co-ordinate Frame of the NDF. If Parameter USEAXIS is zero (*i.e.* if the horizontal axis is annotated with offset from the first array element), then the positions refer to the centres of the first and last elements in the supplied NDF, whether or not these elements are actually visible in the plot. If USEAXIS is not zero (*i.e.* if the horizontal axis is annotated with the value on one of the axes of the NDF's current co-ordinate Frame), then the displayed positions correspond to the two ends of the visible section of the horizontal axis. The appearance of the key can be controlled using Parameter KEYSTYLE. If a null (!) value is supplied, a key is produced if the current co-ordinate Frame of the supplied NDF has two or more axes, but no key is produced if it only has one axis. [!]

KEYSTYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the key (see Parameter KEY).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported). [current value]

LMODE = LITERAL (Read)

LMODE specifies how the defaults for Parameters YBOT and YTOP (the lower and upper limit of the vertical axis of the plot) should be found. The supplied string should consist of up to three sub-strings, separated by commas. The first sub-string must specify the method to use. If supplied, the other two sub-strings should be

numerical values as described below (default values will be used if these sub-strings are not provided). The following methods are available.

- `"Range"` — The minimum and maximum data values (including any error bars) are used as the defaults for YBOT and YTOP. No other sub-strings are needed by this option.
- `"Extended"` — The minimum and maximum data values (including error bars) are extended by percentages of the data range, specified by the second and third sub-strings. For instance, if the value `"Ex,10,5"` is supplied, then the default for YBOT is set to the minimum data value minus 10% of the data range, and the default for YTOP is set to the maximum data value plus 5% of the data range. If only one value is supplied, the second value defaults to the supplied value. If no values are supplied, both values default to `"2.5"`. Care should be taken with this mode if YMAP is set to `"Log"` since the extension to the data range caused by this mode may result in the axis encompassing the value zero.
- `"Percentile"` — The default values for YBOT and YTOP are set to the specified percentiles of the data (excluding error bars). For instance, if the value `"Per,10,99"` is supplied, then the default for YBOT is set so that the lowest 10% of the plotted points are off the bottom of the plot, and the default for YTOP is set so that the highest 1% of the points are off the top of the plot. If only one value, $p1$, is supplied, the second value, $p2$, defaults to $(100 - p1)$. If no values are supplied, the values default to `"5,95"`.
- `"Sigma"` — The default values for YBOT and YTOP are set to the specified numbers of standard deviations below and above the mean of the data. For instance, if the value `"sig,1.5,3.0"` is supplied, then the default for YBOT is set to the mean of the data minus 1.5 standard deviations, and the default for YTOP is set to the mean plus 3 standard deviations. If only one value is supplied, the second value defaults to the supplied value. If no values are provided, both default to `"3.0"`.

The method name can be abbreviated to a single character, and is case insensitive. The initial value is `"Extended"`. [current value]

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. If a null (`!`) value is supplied, the value used is `0.15` (for all edges) if either annotated axes or a key are produced, and zero otherwise. [current value]

MARKER = _INTEGER (Read)

This parameter is only accessed if Parameter MODE is set to `"Chain"` or `"Mark"`. It specifies the symbol with which each position should be marked, and should be given as an integer PGPLOT marker type. For instance, `0` gives a box, `1` gives a dot, `2` gives a cross, `3` gives an asterisk, `7` gives a triangle. The value must be larger than or equal to -31. [current value]

MODE = LITERAL (Read)

Specifies the way in which data values are represented. MODE can take the following

values.

- "Histogram" — An histogram of the points is plotted in the style of a 'staircase' (with vertical lines only joining the *y*-axis values and not extending to the base of the plot). The vertical lines are placed midway between adjacent *x*-axis positions. Bad values are flanked by vertical lines to the lower edge of the plot.
- "GapHistogram" — The same as the "Histogram" option except bad values are not flanked by vertical lines to the lower edge of the plot, leaving a gap.
- "Line" — The points are joined by straight lines.
- "Point" — A dot is plotted at each point.
- "Mark" — Each point is marker with a symbol specified by Parameter MARKER.
- "Chain" — A combination of "Line" and "Mark".
- "Step" — Each point is displayed as a horizontal line, whose length is specified by the axis width of the pixel.

The initial default is "Line". [current value]

NDF = NDF (Read)

NDF structure containing the array to be plotted.

SHAPE = LITERAL (Read)

Specifies the way in which errors are represented. SHAPE can take the following values.

- "Bars" — Bars with serifs (*i.e.* cross pieces at each end) are drawn joining the *x*-error limits and the *y*-error limits. The plotting attribute Size(ErrBars) (see Parameter STYLE) can be used to control the size of these serifs (the attribute value should be a magnification factor; 1.0 gives default serifs).
- "Cross" — San-serif bars are drawn joining the *x*-error limits and the *y*-error limits.
- "Diamond" — Adjacent error limits are joined to form an error diamond.

The length of the error bars can be controlled using Parameters XSIGMA and YSIGMA. The colour, line width and line style used to draw them can be controlled using the plotting attributes Colour(ErrBars), Width(ErrBars) and Style(ErrBars) (see Parameter STYLE). SHAPE is only accessed when Parameter ERRBAR is TRUE. [current value]

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use when drawing the annotated axes, data values, and error markers.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes.

All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the data values is controlled by the attributes Colour(Curves), Width(Curves), *etc.* (the synonym Lines may be used in place of Curves). The appearance of markers used if Parameter MODE is set to `"Point"`, `"Mark"` or `"Chain"` is controlled by Colour(Markers), Width(Markers), *etc.* (the synonym Symbols may be used in place of Markers). The appearance of the error symbols is controlled using Colour(ErrBars), Width(ErrBars), *etc*, (see Parameter SHAPE). [current value]

USEAXIS = LITERAL (Read)

Specifies the quantity to be used to annotate the horizontal axis of the plot using one of the following options.

- An integer index of an axis within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- An axis Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.
- The special value `0` (zero), asks for the distance along the profile from the centre of the first element in the supplied NDF to be used to annotate the axis. This will be measured in the current co-ordinate Frame of the NDF.

The quantity used to annotate the horizontal axis must have a defined value at all points in the array, and must increase or decrease monotonically along the array. For instance, if RA is used to annotate the horizontal axis, then an error will be reported if the profile passes through RA=0 because it will introduce a non-monotonic jump in axis value (from 0h to 24h, or 24h to 0h). If a null (`!`) value is supplied, the value used is `1` if the current co-ordinate Frame in the NDF is one-dimensional and `0` otherwise. `[!]`

XLEFT = LITERAL (Read)

The axis value to place at the left hand end of the horizontal axis. If a null (`!`) value is supplied, the value used is the value for the first element in the supplied NDF (with a margin to include any horizontal error bar). The value supplied may be greater than or less than the value supplied for XRIGHT. A formatted value for the quantity specified by Parameter USEAXIS should be supplied. See also Parameter ALIGN. `[!]`

XMAP = LITERAL (Read)

Specifies how the quantity represented by the *x* axis is mapped on to the plot. The options are as follows.

- `"Pixel"` — The mapping is such that pixel index within the input NDF increases linearly across the plot.

- "Distance" — The mapping is such that distance along the curve within the current WCS Frame of the input NDF increases linearly across the plot.
- "Log" — The mapping is such that the logarithm (base 10) of the value used to annotate the axis increases linearly across the plot. An error will be reported if the dynamic range of the axis is less than 100, or if the range specified by XLEFT and XRIGHT encompasses the value zero.
- "Linear" — The mapping is such that the value used to annotate the axis increases linearly across the plot. Note the corresponding pixel indices always increase left to right in this mode so the annotated values may possibly increase right to left depending on the nature of the WCS mapping.
- "LRLinear" — Like "Linear" except that the pixel indices are reversed if necessary to ensure that the annotated values always increases left to right.
- "Default" — One of "Linear" or "log" is chosen automatically, depending on which one produces a more-even spread of values on the plot.

["Default"]

XRIGHT = LITERAL (Read)
The axis value to place at the right hand end of the horizontal axis. If a null (!) value is supplied, the value used is the value for the last element in the supplied NDF (with a margin to include any horizontal error bar). The value supplied may be greater than or less than the value supplied for XLEFT. A formatted value for the quantity specified by Parameter USEAXIS should be supplied. See also Parameter ALIGN. [!]

XSIGMA = LITERAL (Read)
If horizontal error bars are produced (see Parameter ERRBAR), then XSIGMA gives the number of standard deviations that the error bars are to represent. [current value]

YBOT = LITERAL (Read)
The axis value to place at the bottom end of the vertical axis. If a null (!) value is supplied, the value used is determined by Parameter LMODE. The value of YBOT may be greater than or less than the value supplied for YTOP. If Parameter YMAP is set to "ValueLog", then the supplied value should be the logarithm (base 10) of the bottom data value. See also Parameter ALIGN. [!]

YMAP = LITERAL (Read)
Specifies how the quantity represented by the *y* axis is mapped on to the screen. The options are as follows.

- "Linear" — The data values are mapped linearly on to the screen.
- "Log" — The data values are logged logarithmically on to the screen. An error will be reported if the dynamic range of the axis is less than 100, or if the range specified by YTOP and YBOT encompasses the value zero. For this reason, care should be taken over the choice of value for Parameter LMODE, since some choices could result in the *y* range being extended so far that it encompasses zero.
- "ValueLog" — This is similar to "Log" except that, instead of mapping the data values logarithmically on to the screen, this option maps the log (base 10) of the data values linearly on to the screen. If this option is selected, the values supplied

for Parameters YTOP and YBOT should be values for the logarithm of the data value, not the data value itself.

`["Linear"]`

YSIGMA = LITERAL (Read)
If vertical error bars are produced (see Parameter ERRBAR), then YSIGMA gives the number of standard deviations that the error bars are to represent. [current value]

YTOP = LITERAL (Read)
The axis value to place at the top end of the vertical axis. If a null (`!`) value is supplied, the value used is determined by Parameter LMODE. The value of LTOP may be greater than or less than the value supplied for YBOT. If Parameter YMAP is set to `"ValueLog"`, then the supplied value should be the logarithm (base 10) of the bottom data value. See also Parameter ALIGN. `[!]`

Examples:

```
linplot spectrum
```

Plots data values versus position for the whole of the one-dimensional NDF called spectrum on the current graphics device. If the current co-ordinate Frame of the NDF is also one-dimensional, then the horizontal axis will be labelled with values on Axis 1 of the current co-ordinate Frame. Otherwise, it will be labelled with offset from the first element.

```
linplot map(,100)
```

Plots data values versus position for row 100 in the two-dimensional NDF called map on the current graphics device.

```
linplot spectrum(1:500) device=ps_l
```

Plots data values versus position for the first 500 elements of the one-dimensional NDF called spectrum. The output goes to a text file which can be printed on a PostScript printer.

```
linplot ironarc v style="title=Fe Arc variance"
```

Plots variance values versus position for the whole of the one-dimensional NDF called ironarc on the current graphics device. The plot has a title of `"Fe Arc variance"`.

```
linplot prof useaxis=dec xleft="23:30:22" xright="23:30:45"
```

This plots data values versus declination for those elements of the one-dimensional NDF called prof with declination value between 23d 30m 22s, and 23d 30m 45s. This assumes that the current co-ordinate Frame in the NDF has an axis with symbol `"dec"`.

```
linplot prof useaxis=2 ybot=10 ytop=1000.0 ymap=log xmap=log
```

This plots the data values in the entire one-dimensional NDF called prof, against the

value described by the second axis in the current co-ordinate Frame of the NDF. The values represented by both axes are mapped logarithmically on to the screen. The bottom of the vertical axis corresponds to a data value of 10.0 and the top corresponds to a data value of 1000.0.

```
linplot xspec mode=p errbar xsigma=3 ysigma=3 shape=d style=^my_sty
```

This plots the data values versus position for the dataset called xspec. Each pixel is plotted as a point surrounded by diamond-shaped error bars. The error bars are 3-sigma error bars. The plotting style is read from text file `my_sty`. This could, for instance, contain strings such as: `colour(err)=pink, colour(sym)=red, tickall=0, edge(2)=right`. These cause the error bars to be drawn in pink, the points to be drawn in red, tick marks to be restricted to the labelled edges of the plot, and the vertical axis (Axis 2) to be annotated on the right-hand edge of the plot. The plotting style specified in file `my_sty` becomes the default plotting style for future invocations of LINPLOT.

```
linplot xspec mode=p errbar xsigma=3 ysigma=3 shape=d style=+^my_sty
```

This is the same as the previous example, except that the style specified in file `my_sty` does not become the default style for future invocations of LINPLOT.

```
linplot ndf=spectrum noclear align
```

Plots data values versus pixel co-ordinate for the whole of the one-dimensional NDF called spectrum on the current graphics device. The plot is drawn over any existing plot and inherits the bounds of the previous plot on both axes. A warning will be reported if the labels for the horizontal axes of the two plots are different.

```
linplot spectrum system="'system(1)=freq,unit(1)=GHz'"
```

This example assumes that the current co-ordinate Frame of NDF `"spectrum"` is a SpecFrame. The horizontal axis (`"Axis 1"`) is labelled with frequency values, in units of GHz. If the SpecFrame represents some other system (such as wavelength, velocity, energy) or has some other units, then the conversion is done automatically. Note, a SpecFrame is a specialised class of Frame which knows how to do these conversions; the above command will fail if the current co-ordinate Frame in the NDF is a simple Frame (such as the AXIS Frame). A SpecFrame can be created from an AXIS Frame using application WCSADD.

Notes:

- If no Title is specified via the STYLE parameter, then the TITLE component in the NDF is used as the default title for the annotated axes. Should the NDF not have a TITLE component, then the default title is instead taken from current co-ordinate Frame in the NDF, unless this attribute has not been set explicitly, whereupon the name of the NDF is used as the default title.

- Default axis errors and widths are used, if none are present in the NDF. The defaults are the constants 0 and 1 respectively.
- The application stores a number of pictures in the graphics database in the following order: a FRAME picture containing the annotated axes, data plot, and optional key; a KEY picture to store the key if present; and a DATA picture containing just the data plot. Note, the FRAME picture is only created if annotated axes or a key has been drawn, or if non-zero margins were specified using Parameter MARGIN. The world co-ordinates in the DATA picture will correspond to offset along the profile on the horizontal axis, and data value (or logarithm of data value) on the vertical axis. On exit the current database picture for the chosen device reverts to the input picture.

Related Applications :

KAPPA: PROFILE, CLINPLOT; MLINPLOT; FIGARO: ESPLOT, IPLOTS, MSPLOT, SPECGRID; SPLOT, SPLAT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, QUALITY, LABEL, TITLE, WCS, and UNITS components of the NDF.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Only double-precision floating-point data can be processed directly. Other non-complex data types will undergo a type conversion before the plot is drawn.

LISTMAKE
Creates a catalogue holding a positions list

Description:

This application creates a catalogue containing a list of positions supplied by the user, together with information describing the co-ordinate Frames in which the positions are defined. Integer position identifiers which allow positions to be distinguished are also stored in the catalogue. The catalogue may be manipulated using the CURSA package, and is stored in either FITS binary format or the *Small Text List* (STL) format defined by CURSA.

If an NDF is specified using Parameter NDF, then the positions should be given in the current co-ordinate Frame of the NDF. Information describing the co-ordinate Frames available within the NDF will be copied to the output positions list. Subsequent applications can use this information in order to align the positions with other data sets.

If no NDF is specified, then the user must indicate the co-ordinate Frame in which the positions will be supplied using Parameter FRAME. A description of this Frame will be written to the output positions list for use by subsequent applications.

The positions themselves may be supplied within a text file, or may be given in response to repeated prompts for a parameter. Alternatively, pixel centres in the NDF supplied for Parameter NDF can be used (see Parameter MODE).

The output can be initialised by copying positions from an existing positions list. Any positions supplied directly by the user are then appended to the end of this initial list (see Parameter INCAT).

Usage:

```
listmake outcat [ndf] [mode] { file=?
                             { position=?
                        mode
```

Parameters:

CATFRAME = LITERAL (Read)

A string determining the co-ordinate Frame in which positions are to be stored in the output catalogue associated with Parameter OUTCAT. The string supplied for CATFRAME can be one of the following options.

- A domain name such as SKY, AXIS, PIXEL.
- An integer value giving the index of the required Frame.
- An IRAS90 *Sky Co-ordinate System* (SCS) values such as `"EQUAT(J2000)"` (see SUN/163).

If a null (`!`) value is supplied, the positions will be stored in the current Frame. `[!]`

CATEPOCH = _DOUBLE (Read)

The epoch at which the sky positions stored in the output catalogue were determined.

It will only be accessed if an epoch value is needed to qualify the co-ordinate Frame specified by COLFRAME. If required, 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.

DESCRIBE = _LOGICAL (Read)
If TRUE, a detailed description of the co-ordinate Frame in which positions are required will be displayed before the positions are obtained using either Parameter POSITION or FILE. [current value]

DIM = _INTEGER (Read)
The number of axes for each position. It is only accessed if a null value is supplied for Parameter NDF.

EPOCH = _DOUBLE (Read)
If an IRAS90 Sky Co-ordinate System specification is supplied (using Parameter DOMAIN) for a celestial co-ordinate system, 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.

FILE = FILENAME (Read)
A text file containing the positions to be stored in the output positions list. Each line should contain the formatted axis values for a single position, separated by white space. It is only accessed if Parameter MODE is given the value "File".

FRAME = LITERAL (Read)
Specifies the co-ordinate Frame of the positions supplied through Parameters POSITION or FILE. There is a cascade of allowed interpretations of this parameter value; the search for the co-ordinate Frame ends once there is a successful interpretation, otherwise the search moves on to the next possible meaning in the following order.

(1) An HDS path containing a WCS FrameSet, whose current Frame defines the co-ordinate Frame.
(2) The name of an NDF, whose current WCS co-ordinate Frame is used.
(3) If the parameter value ends with .FIT, an attempt is made to interpret the parameter value as the name of a FITS file. If successful, the primary WCS co-ordinate system from the primary HDU headers is used.
(4) A text file containing either an AST Frame dump (such as produced by commands in the ATOOLS package), or a set of FITS WCS headers.
(5) An IRAS90 *Sky Co-ordinate System* (SCS) string such as "EQUAT(J2000)" (see SUN/163), whereupon the positions are assumed to be two-dimensional celestial co-ordinates in the specified system.
(6) Domain name without any interpretation. Any Domain name may be supplied, but normally one of the standard Domain names, such as GRID, PIXEL, GRAPHICS should be given. Parameter DIM is used to determine the number of axes in the Frame.

This parameter is only accessed if the parameter NDF is given a null value.

INCAT = FILENAME (Read)
A catalogue containing an existing positions list which is to be included at the start

of the output positions list. These positions are mapped into the current co-ordinate Frame of the supplied NDF, or into the Frame specified by Parameter FRAME if no NDF was supplied. A message is displayed indicating the Frame in which alignment occurred. They are then stored in the output list before any further positions are added. A null value may be supplied if there is no input positions list. [!]

MODE = LITERAL (Read)
The mode by which the positions are to be obtained. The options are as follows.

- "Interface" — The positions are obtained using Parameter POSITION.
- "File" — The positions are to be read from a text file specified using Parameter FILE.
- "Good" — The positions used are the pixel centres in the data file specified by Parameter NDF. Only the pixels that have good values in the Data array of the NDF are used.
- "Pixel" — The positions used are the pixel centres in the data file specified by Parameter NDF. All pixel are used, whether the pixel values are good or not.

["Interface"]

NDF = NDF (READ)
The NDF which defines the available co-ordinate Frames in the output positions list. If an NDF is supplied, the positions obtained using Parameter POSITION or FILE are assumed to be in the current co-ordinate Frame of the NDF, and the WCS component of the NDF is copied to the output positions list. If a null value is supplied, the single co-ordinate Frame defined by Parameter FRAME is stored in the output positions list, and supplied positions are assumed to be in the same Frame. [!]

OUTCAT = FILENAME (Write)
The catalogue holding the output positions list.

POSITION = LITERAL (Read)
The co-ordinates of a single position to be stored in the output positions list. Supplying ":" will display details of the co-ordinate Frame in which the position is required. The position should be given as a list of formatted axis values separated by white space. You are prompted for new values for this parameter until a null value is entered. It is only accessed if Parameter MODE is given the value "Interface".

TITLE = LITERAL (Read)
A title for the output positions list. If a null (!) value is supplied, the value used is obtained from the input positions list if one is supplied. Otherwise, it is obtained from the NDF if one is supplied. Otherwise, it is "Output from LISTMAKE". [!]

Examples:

```
listmake newlist frame=pixel dim=2
```

This creates a FITS binary catalogue called newlist.FIT containing a list of positions, together with a description of a single two-dimensional pixel co-ordinate Frame. The positions are supplied as a set of space-separated pixel co-ordinates in response to repeated prompts for the Parameter POSITION.

```
listmake stars.txt frame=equat(B1950) epoch=1962.3
```

This creates a catalogue called `stars.txt` containing a list of positions, together with a description of a single FK4 equatorial RA/DEC co-ordinate Frame (referenced to the B1950 equinox). The catalogue is stored in a text file using the CAT *Small Text List* format (`"STL"`—see SUN/190). The positions were determined at epoch B1962.3. The epoch of observation is required since the underlying model on which the FK4 system is based is non-inertial and rotates slowly with time, introducing fictitious proper motions. The positions are supplied hours and degrees values in reponse to repeated prompts for Parameter POSITIONS.

```
listmake outlist ndf=allsky mode=file file=stars
```

This creates a FITS binary catalogue called `outlist.FIT` containing a list of positions, together with descriptions of all the co-ordinate Frames contained in the NDF allsky. The positions are supplied as co-ordinates within the current co-ordinate Frame of the NDF. Application WCSFRAME can be used to find out what this Frame is. The positions are supplied in a text file called `stars`.

```
listmake out.txt incat=old.fit frame=gal
```

This creates an STL format catalogue stored in a text file called `out.txt` containing a list of positions, together with a description of a single galactic co-ordinate Frame. The positions contained in the existing binary FITS catalogue `old.fit` are mapped into galactic co-ordinates (if possible) and stored in the output positions list. Further galactic co-ordinate positions are then obtained by repeated prompting for the Parameter POSITION. These positions are appended to the positions obtained from file `old.fit`.

```
listmake out.txt incat=old.fit ndf=cobe
```

As above but the output positions list contains copies of all the Frames in the NDF cobe. The positions in `old.fit` are mapped into the current co-ordinate Frame of the NDF (if possible) before being stored in the output positions list. The new positions must also be supplied in the same Frame (using Parameter POSITION).

```
listmake profpos.fit ndf=prof1 mode=pixel
```

This creates a positions list called `profpos.fit` containing the positions of all the pixel centres in the one-dimensional NDF called prof. This could for instance be used as input to application PROFILE in order to produce another profile in which the samples are at the same positions as those in NDF prof.

Notes:

- This application uses the conventions of the CURSA package for determining the formats of input and output catalogues. If a file type of .fit is given, then the catalogue is assumed to be a FITS binary table. If a file type of .txt is given, then the catalogue is

assumed to be stored in a text file in STL format. If no file type is given, then `".fit"` is assumed.

- There is a limit of 200 on the number of positions which can be given using Parameter POSITION. There is no limit on the number of positions which can be given using Parameter FILE.
- Position identifiers are asigned to the supplied positions in the order in which they are supplied. If no input positions list is given using Parameter INCAT, then the first supplied position will be assigned the identifier `"1"`. If an input positions list is given, then the first supplied position is assigned an identifier one greater than the largest identifier in the input positions list.

Related Applications :

KAPPA: CURSOR, LISTSHOW; CURSA: XCATVIEW, CATSELECT.

LISTSHOW
Reports the positions stored in a positions list

Description:

This application reports positions contained in a catalogue. The catalogue should have the form of a positions list as produced, for instance, by applications LISTMAKE and CURSOR. By default all positions in the catalogue are reported, but a subset may be reported by specifying a range of *position identifiers* (see Parameters FIRST, LAST, and STEP).

An NDF may be supplied (see Parameter NDF) in which case the NDF pixel values at the positions listed in the catalogue are reported, using the interpolation method specified by Parameter METHOD. The pixel values are also written to an output parameter (see Parameter PIXVALS).

Positions may be reported in a range of co-ordinate Frames dependent on the information stored in the supplied positions list (see Parameter FRAME). The selected positions are written to an output parameter (Parameter POSNS), and may also be written to an output positions list (see Parameter OUTCAT). The formatted screen output can be saved in a logfile (see Parameter LOGFILE). The formats used to report the axis values can be controlled using Parameter STYLE.

Graphics may also be drawn marking the selected positions (see Parameters PLOT and LABEL). The supplied positions are aligned with the picture specified by Parameter NAME. If possible, this alignment occurs within the co-ordinate Frame specified using Parameter FRAME. If this is not possible, alignment may occur in some other suitable Frame. A message is displayed indicating the Frame in which alignment occurred. If the supplied positions are aligned successfully with a picture, then the range of Frames in which the positions may be reported on the screen is extended to include all those associated with the picture.

Usage:

```
listshow incat [frame] [first] [last] [plot] [device]
```

Parameters:

CATFRAME = LITERAL (Read)

A string determining the co-ordinate Frame in which positions are to be stored in the output catalogue associated with Parameter OUTCAT. See Parameter FRAME for a description of the allowed values for this parameter. If a null (`!`) value is supplied, the positions will be stored the Frame used to specify positions within the input catalogue. `[!]`

CATEPOCH = _DOUBLE (Read)

The epoch at which the sky positions stored in the output catalogue were determined. It will only be accessed if an epoch value is needed to qualify the co-ordinate Frame specified by COLFRAME. If required, 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.

CLOSE = _LOGICAL (Read)
This parameter is only accessed if Parameter PLOT is set to "Chain" or "Poly". If TRUE, polgons will be closed by joining the first position to the last position. [current value]

COMP = LITERAL (Read)
The NDF array component to be displayed if a non-null value is supplied for Parameter NDF. It may be "Data", "Variance", "Error", or "Quality". ["Data"]

DESCRIBE = _LOGICAL (Read)
If TRUE, a detailed description of the co-ordinate Frame in which the positions will be reported is displayed before the positions. [current value]

DEVICE = DEVICE (Read)
The graphics workstation. Only accessed if Parameter PLOT indicates that graphics are required. [current graphics device]

EPOCH = _DOUBLE (Read)
If an IRAS90 Sky Co-ordinate System specification is supplied (using Parameter FRAME) for a celestial co-ordinate system, 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.

FIRST = _INTEGER (Read)
The identifier for the first position to be displayed. Positions are only displayed which have identifiers in the range given by Parameters FIRST and LAST. If a null (!) value is supplied, the value used is the lowest identifier value in the positions list. [!]

FRAME = LITERAL (Read)
A string determining the co-ordinate Frame in which positions are to be reported. This application can report positions in any of the co-ordinate Frames stored with the positions list. The string supplied for FRAME can be one of the following.

- A domain name such as SKY, AXIS, PIXEL.
- An integer value giving the index of the required Frame.
- An IRAS90 *Sky Co-ordinate System* (SCS) values such as "EQUAT(J2000)" (see SUN/163).

If a null value (!) is supplied, positions are reported in the co-ordinate Frame which was current when the positions list was created. The user is re-prompted if the specified Frame is not available within the positions list. The range of Frames available will include all those read from the supplied positions list. In addition, if a graphics device is opened (*i.e.* if Parameter PLOT is set to anything other than "None"), then all the Frames associated with the picture specified by Parameter NAME will also be available. [!]

GEODESIC = _LOGICAL (Read)
This parameter is only accessed if Parameter PLOT is set to "Chain" or "Poly". It specifies whether the curves drawn between positions should be stright lines, or should be geodesic curves. In many co-ordinate Frames geodesic curves will be

simple straight lines. However, in others (such as the majority of celestial co-ordinate Frames) geodesic curves will be more complex curves tracing the shortest path between two positions in a non-linear projection. [FALSE]

INCAT = FILENAME (Read)
A catalogue containing a positions list such as produced by applications LISTMAKE and CURSOR.

JUST = LITERAL (Read)
A string specifying the justification to be used when displaying text strings at the supplied positions. This parameter is only accessed if Parameter PLOT is set to "Text". The supplied string should contain two characters; the first should be "B", "C", or "T", meaning bottom, centre, or top respectively. The second should be "L", "C", or "R", meaning left, centre, or right respectively. The text is displayed so that the supplied position is at the specified point within the displayed text string. [CC]

LABEL = _LOGICAL (Read)
If TRUE the positions are labelled on the graphics device specified by Parameter DEVICE. The offset of the centre of each label from the corresponding position is controlled using the NumLabGap(1) and NumLabGap(2) plotting attributes, and the appearance of the labels is controlled using attributes Colour(NumLab), Size(NumLab), *etc*. These attributes may be specified using Parameter STYLE. The content of the label is determined by Parameter LABTYPE. [FALSE]

LABTYPE = LITERAL (Read)
Determines what sort of labels are drawn if the LABEL parameter is set TRUE. It can be either of the following.

- "ID" — causes the integer identifier associated with each row to be used as the label for the row.
- "LABEL" — causes the textual label associated with each row to be used as the label for the row. These strings are read from the "LABEL" column of the supplied catalogue.

If a null (!) value is supplied, a default of "LABEL" will be used if the input catalogue contains a "LABEL" column. Otherwise, a default of "ID" will be used. [!]

LAST = _INTEGER (Read)
The identifier for the last position to be displayed. Positions are only displayed which have identifiers in the range given by Parameters FIRST and LAST. If a null (!) value is supplied, the value used is the highest identifier value in the positions list. [!]

LOGFILE = FILENAME (Write)
The name of the text file in which the formatted co-ordinates of the selected positions may be stored. This is intended primarily for recording the screen output, and not for communicating positions to subsequent applications. A null string (!) means that no file is created. [!]

MARKER = _INTEGER (Read)
This parameter is only accessed if Parameter PLOT is set to "Chain" or "Mark". It specifies the type of marker with which each position should be marked, and should be given as an integer PGPLOT marker type. For instance, 0 gives a box, 1 gives a dot, 2 gives a cross, 3 gives an asterisk, 7 gives a triangle. The value must be larger than or equal to −31. [current value]

METHOD = LITERAL (Read)

The method to use when sampling the input NDF (if any) specified by Parameter NDF at each of the positions in the catalogue. For details on these schemes, see the description of routine AST_RESAMPLEx in SUN/210. Note, `"Nearest"` is always used if Parameter COMP is `"Quality"` . METHOD can take the following values.

- `"Bilinear"` — The displayed pixel values are calculated by bi-linear interpolation among the four nearest pixels values in the input NDF. This produces smoother output NDFs than the nearest-neighbour scheme, but is marginally slower.
- `"Nearest"` — Each displayed pixel value is the value of the nearest input pixel.
- `"Sinc"` — Uses the $\mathrm{sinc}(\pi x)$ kernel, where x is the pixel offset from the interpolation point, and $\mathrm{sinc}(z) = \sin(z)/z$. Use of this scheme is not recommended.
- `"SincSinc"` — Uses the $\mathrm{sinc}(\pi x)\mathrm{sinc}(k\pi x)$ A valuable general-purpose scheme, intermediate in its visual effect on NDFs between the bi-linear and nearest-neighbour schemes.
- `"SincCos"` — Uses the $\mathrm{sinc}(\pi x)\cos(k\pi x)$ kernel. Gives similar results to the `"SincSinc"` scheme.
- `"SincGauss"` — Uses the $\mathrm{sinc}(\pi x)e^{-kx^2}$ kernel. Good results can be obtained by matching the FWHM of the envelope function to the point-spread function of the input data (see Parameter PARAMS).
- `"Somb"` — Uses the $\mathrm{somb}(\pi x)$ kernel, where x is the pixel offset from the interpolation point, and $\mathrm{somb}(z) = 2 * J_1(z)/z$. J_1 is the first-order Bessel function of the first kind. This scheme is similar to the `"Sinc"` scheme.
- `"SombCos"` — Uses the $\mathrm{somb}(\pi x)\cos(k\pi x)$ kernel. This scheme is similar to the `"SincCos"` scheme.
- `"Gauss"` — Uses the e^{-kx^2} kernel. The FWHM of the Gaussian is given by Parameter PARAMS(2), and the point at which to truncate the Gaussian to zero is given by Parameter PARAMS(1).

The initial default is " Nearest" . [current value]

NAME = LITERAL (Read)

Determines the graphics database picture with which the supplied positions are to be aligned. Only accessed if Parameter PLOT indicates that some graphics are to be produced. A search is made for the most recent picture with the specified name (*e.g.* DATA, FRAME or KEY) within the current picture. If no such picture can be found, or if a null value is supplied, the current picture itself is used. The name BASE can also be supplied as a special case, which causes the BASE picture to be used even though it will not in general fall within the current picture. `["DATA"]`

NDF = NDF (Read)

If an NDF is supplied, the values within the NDF at the positions specified in the input catalogue are displayed on the screen and written to Output Parameter PIXVALS. The displayed values are calculated by interpolating between the NDF pixel values using the interpolation method specified by Parameter METHOD. The NDF array component to be displayed is specified by Parameter COMP. [!]

OUTCAT = FILENAME (Write)

The output catalogue in which to store the selected positions. If a null value is supplied, no output catalogue is produced. See Parameter COLFRAME. `[!]`

PARAMS(2) = _DOUBLE (Read)

An optional array which consists of additional parameters required by the Sinc, SincSinc, SincCos, SincGauss, Somb, SombCos, and Gauss methods (see Parameter METHOD).

PARAMS(1) is required by all the above schemes. It is used to specify how many pixels are to contribute to the interpolated result on either side of the interpolation or binning point in each dimension. Typically, a value of 2 is appropriate and the minimum allowed value is 1 (i.e. one pixel on each side). A value of zero or fewer indicates that a suitable number of pixels should be calculated automatically. [0]

PARAMS(2) is required only by the Gauss, SombCos, SincSinc, SincCos, and SincGauss schemes. For the SombCos, SincSinc, and SincCos schemes, it specifies the number of pixels at which the envelope of the function goes to zero. The minimum value is 1.0, and the run-time default value is 2.0. For the Gauss and SincGauss schemes, it specifies the full-width at half-maximum (FWHM) of the Gaussian envelope measured in output pixels. The minimum value is 0.1, and the run-time default is 1.0. On astronomical NDFs and spectra, good results are often obtained by approximately matching the FWHM of the envelope function, given by PARAMS(2), to the point-spread function of the input data. []

PLOT = LITERAL (Read)

The type of graphics to be used to mark the positions on the graphics device specified by Parameter DEVICE. The appearance of these graphics (colour, size, *etc.*) is controlled by the STYLE parameter. PLOT can take any of the following values.

- "None" — No graphics are produced.
- "Mark" — Each position is marked with a marker of type specified by Parameter MARKER.
- "Poly" — Causes each position to be joined by a line to the previous position. These lines may be simple straight lines or geodesic curves (see Parameter GEODESIC). The polygons may optionally be closed by joining the last position to the first (see Parameter CLOSE).
- "Chain" — This is a combination of "Mark" and "Poly". Each position is marked by a marker and joined by a line to the previous position. Parameters MARKER, GEODESIC, and CLOSE are used to specify the markers and lines to use.
- "Box" — A rectangular box with edges parallel to the edges of the graphics device is drawn between each pair of positions.
- "Vline" — A vertical line is drawn through each position, extending the entire height of the selected picture.
- "Hline" — A horizontal line is drawn through each position, extending the entire width of the selected picture.
- "Cross" — A combination of "Vline" and "Hline".
- "STCS" — Indicates that each position should be marked using the two-dimensional STC-S shape read from the catalogue column specified by Parameter STCSCOL.
- "Text" — A text string is used to mark each position. The string is drawn horizontally with the justification specified by Parameter JUST. The strings to use for each position are specified using Parameter STRINGS.

- "Blank" — The graphics device is opened and the picture specified by Parameter NAME is found, but no actual graphics are drawn to mark the positions. This can be useful if you just want to transform the supplied positions into one of the co-ordinate Frames associated with the picture, without drawing anything (see Parameter FRAME).

Each position may also be separately labelled with its integer identifier value by giving a TRUE value for Parameter LABEL. ["None"]

POSNS() = _DOUBLE (Write)

The unformatted co-ordinates of the positions selected by Parameters FIRST and LAST, in the co-ordinate Frame selected by FRAME. The axis values are stored as a one-dimensional vector. All the Axis-1 values for the selected positions are stored first, followed by the Axis-2 values, *etc*. The number of positions in the vector is written to the output Parameter NUMBER, and the number of axes per position is written to the output Parameter DIM. The axis values may not be in the same units as the formatted values shown on the screen. For instance, unformatted celestial co-ordinate values are stored in units of radians.

STEP = _INTEGER (Read)

The increment between position identifiers to be displayed. Specifying a value larger than 1 causes a subset of the position identifiers between FIRST and LAST to be displayed. [1]

STCSCOL = LITERAL (Read)

The name of a catalogue column containing an STC-S description of a two-dimensional spatial shape associated with each position. The STC-S format is an IVOA proposal for describing regions of space, time and spectral position. For further details, see the STC-S document on the IVOA web site (http://www.ivoa.net/Documents/). An STC-S description of a shape includes the co-ordinate system in which the shape is defined. This application assumes that all the STC-S shapes read from the specified column will be defined within the same co-ordinate system. The transformation from the STC-S co-ordinate system to the co-ordinate system of the displayed image is determined once from the first shape plotted, and then re-used for all later shapes. ["Shape"]

STRINGS = LITERAL (Read)

A group of text strings which are used to mark the supplied positions if Parameter PLOT is set to "Text". The first string in the group is used to mark the first position, the second string is used to mark the second position, *etc*. If more positions are given than there are strings in the group, then the extra positions will be marked with an integer value indicating the index within the list of supplied positions. (Note, these integers may be different from the identifiers in the supplied positions list). If a null value (!) is given for the parameter, then all positions will be marked with the integer indices, starting at 1.

A comma-separated list should be given in which each element is either a marker string, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Note, strings within text files can be separated by new lines as well as commas.

STYLE = GROUP (Read)

A group of attribute settings describing the style to use when formatting the co-

ordinate values displayed on the screen, and when drawing the graphics specified by Parameter PLOT.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

In addition to the attributes which control the appearance of the graphics (Colour, Font, *etc.*), the following attributes may be set in order to control the appearance of the formatted axis values reported on the screen: Format, Digits, Symbol, Unit. These may be suffixed with an axis number (*e.g.* Digits(2)) to refer to the values displayed for a specific axis. [current value]

Results Parameters:

DIM = _INTEGER (Write)
The number of axes for each position written to output Parameter POSNS.

NUMBER = _INTEGER (Write)
The number of positions selected.

PIXVALS() = _DOUBLE (Write)
The pixel values at the listed positions. Only used if a non-null value is supplied for Parameter NDF.

Examples:

```
listshow stars pixel
```

This displays the pixel co-ordinates of all the positions stored in the FITS binary catalogue `stars.fit`. They are all written to the output Parameter POSNS.

```
listshow star outcat=star-gal catframe=gal
```

This copies a position list from catalogue `star` to a new catalogue called `star-gal`. The positions are stored in galactic co-ordinates in the output catalogue.

```
listshow stars.fit equat(J2010) first=3 last=3
```

This extracts Position 3 from the catalogue `stars.fit` transforming it into FK5 equatorial RA/DEC co-ordinates (referenced to the J2010 equinox), if possible. The RA/DEC values (in radians) are written to the output Parameter POSNS.

```
listshow stars_2.txt style="digits(1)=5,digits(2)=7"
```

This lists the positions in the STL format catalogue contained in text file `stars_2.txt` in their original co-ordinate Frame. By default, five digits are used to format Axis-1 values, and seven to format Axis-2 values. These defaults are overridden if the attributes Format(1) and/or Format(2) are assigned values in the description of the current Frame stored in the positions list.

```
listshow s.txt plot=marker marker=3 style="colour(marker)=red,size=2"
```

This marks the positions in `s.txt` on the currently selected graphics device using PGPLOT Marker 3 (an asterisk). The positions are aligned with the most recent DATA picture in the current picture. The markers are red and are twice the default size. The positions are likely not to be reported on the screen.

Notes:

- This application uses the conventions of the CURSA package for determining the formats of input and output catalogues. If a file type of .fits is given, then the catalogue is assumed to be a FITS binary table. If a file type of .txt is given, then the catalogue is assumed to be stored in a text file in *Small Text List* (STL) format. If no file type is given, then `.fit` is assumed.
- The positions are not displayed on the screen when either the message filter environment variable MSG_FILTER is set to `NORMAL` and any graphics or labels are being plotted (see Parameters PLOT and LABEL); or when MSG_FILTER is set to `QUIET` and no graphics are produced. The creation of output parameters and files is unaffected by MSG_FILTER.

Related Applications :

KAPPA: CURSOR, LISTMAKE; CURSA: XCATVIEW, CATSELECT.

LOG10
Takes the base-10 logarithm of an NDF data structure

Description:

This routine takes the base-10 logarithm of each pixel of a NDF to produce a new NDF data structure.

This command is a synonym for `logar base=10D0`.

Usage:

```
log10 in out
```

Parameters:

IN = NDF (Read)
Input NDF data structure.

OUT = NDF (Write)
Output NDF data structure being the logarithm of the input NDF.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
log10 a b
```

This takes logarithms to base ten of the pixels in the NDF called a, to make the NDF called b. NDF b inherits its title from a.

```
log10 title="Abell 4321" out=b in=a
```

This takes logarithms to base ten of the pixels in the NDF called a, to make the NDF called b. NDF b has the title `"Abell 4321"`.

Related Applications :

KAPPA: LOGAR, LOGE, EXP10, EXPE, EXPON, POW; FIGARO: IALOG, ILOG, IPOWER.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

LOGAR
Takes the logarithm (specified base) of an NDF data structure

Description:

This routine takes the logarithm to a specified base of each pixel of a NDF to produce a new NDF data structure.

Usage:

```
logar in out base
```

Parameters:

BASE = LITERAL (Read)
The base of the logarithm to be applied. A special value "Natural" gives natural (base-e) logarithms.

IN = NDF (Read)
Input NDF data structure.

OUT = NDF (Write)
Output NDF data structure being the logarithm of the input NDF.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

Examples:

```
logar a b 10
```

This takes logarithms to base ten of the pixels in the NDF called a, to make the NDF called b. NDF b inherits its title from a.

```
logar base=8 title="HD123456" out=b in=a
```

This takes logarithms to base eight of the pixels in the NDF called a, to make the NDF called b. NDF b has the title "HD123456".

Related Applications :

KAPPA: LOG10, LOGE, EXP10, EXPE, EXPON, POW; FIGARO: IALOG, ILOG, IPOWER.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

LOGE
Takes the natural logarithm of an NDF data structure

Description:

This routine takes the natural logarithm of each pixel of a NDF to produce a new NDF data structure.

This command is a synonym for `logar base=natural`.

Usage:

```
loge in out
```

Parameters:

IN = NDF (Read)
Input NDF data structure.

OUT = NDF (Write)
Output NDF data structure being the logarithm of the input NDF.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
loge a b
```

This takes the natural logarithm of the pixels in the NDF called a, to make the NDF called b. NDF b inherits its title from a.

```
loge title="Cas A" out=b in=a
```

This takes natural logarithms of the pixels in the NDF called a, to make the NDF called b. NDF b has the title `"Cas A"`.

Related Applications :

KAPPA: LOG10, LOGAR, EXP10, EXPE, EXPON, POW; FIGARO: IALOG, ILOG, IPOWER.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

LOOK
List pixel values in a two-dimensional NDF

Description:

This application lists pixel values within a region of a two-dimensional NDF . The listing may be displayed on the screen and logged in a text file (see Parameter LOGFILE). The region to be listed can be specified either by giving its centre and size or its corners, or by giving an 'ARD Description' for the region (see Parameter MODE). The top-right pixel value is also written to an output parameter (VALUE). The listing may be produced in several different formats (see Parameter FORMAT), and the format of each individual displayed data value can be controlled using Parameter STYLE.

Usage:

```
look ndf centre [size] [logfile] [format] [comp] [mode]
     { arddesc=?
     { ardfile=?
     { lbound=?  ubound=?
     { centre=?
   mode
```

Parameters:

AGAIN = _LOGICAL (Read)

If `TRUE`, the user is prompted for further regions to list until a `FALSE` value is obtained. `[FALSE]`

ARDDESC = LITERAL (Read)

An 'ARD Description' for the parts of the image to be listed. Multiple lines can be supplied by ending each line with a hyphen, in which case further prompts for ARDDESC are made until a value is supplied which does not end with a hyphen. All the supplied values are then concatenated together (after removal of the trailing hyphens). ARCDESC is only acessed if MODE is `"ARD"`. Positions in the ARD description are assumed to be in the current co-ordinate Frame of the NDF unless there are COFRAME or WCS statements which indicate a different system. See "Notes" below.

ARDFILE = FILENAME (Read)

The name of an existing text file containing an 'ARD Description' for the parts of the image to be listed. ARDFILE is only accessed if MODE is `"ARDFile"`. Positions in the ARD description are assumed to be in pixel co-ordinates unless there are COFRAME or WCS statements that indicate a different system. See "Notes" below.

CENTRE = LITERAL (Read)

The co-ordinates of the data pixel at the centre of the area to be displayed, in the current co-ordinate Frame of the NDF (supplying a colon `":"` will display details of the current co-ordinate Frame). The position should be supplied as a list of formatted

axis values separated by spaces or commas. See also Parameter USEAXIS. Only acessed if MODE is "Centre".

COMP = LITERAL (Read)
The NDF array component to be displayed. It may be "Data", "Quality", "Variance", or "Error" (where "Error" is an alternative to "Variance" and causes the square root of the variance values to be displayed). If "Quality" is specified, then the quality values are treated as numerical values (in the range 0 to 255). ["Data"]

FORMAT = LITERAL (Read)
Specifies the format for the listing from the following options.

- "strips" — The area being displayed is divided up into vertical strips of limited width. Each strip is displayed in turn, with *y* pixel index at the left of each row, and *x* pixel index at the top of each column. The highest row is listed first in each strip. This format is intended for human readers — the others are primarily intended for being read by other software.
- "clist" — Each row of textual output consists of an *x* pixel index, followed by a *y* pixel index, followed by the pixel data value. No headers or blank lines are included. The pixels are listed in 'Fortran order'—the lower-left pixel first, and the upper-right pixel last.
- "cglist" — Like "clist" except that bad pixel are omitted from the list.
- "vlist" — Each row of textual output consists of just the pixel data value. No headers or blank lines are included. The pixels are listed in 'Fortran order'—the lower-left pixel first, and the upper-right pixel last.
- "wlist" — Each row of textual output consists of the WCS co-ordinate values, followed by the pixel data value. No headers or blank lines are included. The pixels are listed in 'Fortran order'—the lower-left pixel first, and the upper-right pixel last.
- "wglist" — Like "wlist" except that bad pixel are omitted from the list.
- "region" — The pixel data values are listed as a two-dimensional region. Each row of textual output contains a whole row of data values. The textual output may be truncated if it is too wide. The lowest row is listed first.

In all cases, adjacent values are separated by spaces, and bad pixel values are represented by the string "BAD". ["strips"]

LBOUND = LITERAL (Read)
The co-ordinates of the data pixel at the bottom-left of the area to be displayed, in the current co-ordinate Frame of the NDF (supplying a colon ":" will display details of the current co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas. See also Parameter USEAXIS. A null (!) value causes the bottom-left corner of the supplied NDF to be used. LBOUND is only accessed if MODE is "Bounds".

LOGFILE = FILENAME (Write)
The name of the text file in which the textual output may be stored. See MAXLEN. A null string (!) means that no file is created. [!]

MAXLEN = _INTEGER (Read)
The maximum number of characters in a line of textual output. The line is truncated after the last complete value if it would extend beyond this value. [80]

MODE = LITERAL (Read)
Indicates how the region to be listed will be specified:

- "All" — The entire NDF is used.
- "Centre" — The centre and size of the region are specified using Parameters CENTRE and SIZE.
- "Bounds" — The bounds of the region are specified using Parameters LBOUND and UBOUND.
- "ARDFile" — The region is given by an 'ARD Description' supplied within a text file specified using Parameter ARDFILE. Pixels outside the ARD region are represented by the string "OUT".
- "ARD" — The region is given using an ARD description supplied directly using Parameter ARDDESC. Pixels outside the ARD region are represented by the string "OUT".

["Centre"]

NDF = NDF (Read)
The input NDF structure containing the data to be displayed.

SIZE(2) = _INTEGER (Read)
The dimensions of the rectangular area to be displayed, in pixels. If a single value is given, it is used for both axes. The area is centred on the position specified by Parameter CENTRE. It is only accessed if MODE is "Centre". [7]

STYLE = GROUP (Read)
A group of attribute settings describing the format to use for individual data values.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

Data values are formatted using attributes Format(1) and Digits(1). [current value]

UBOUND = LITERAL (Read)
The co-ordinates of the data pixel at the top-right corner of the area to be displayed, in the current co-ordinate Frame of the NDF (supplying a colon ":" will display details of the current co-ordinate Frame). The position should be supplied as a list of

formatted axis values separated by spaces or commas. See also Parameter USEAXIS. A null (!) value causes the top-right corner of the supplied NDF to be used. Only acessed if MODE is "Bounds".

USEAXIS = GROUP (Read)

USEAXIS is only accessed if the current co-ordinate Frame of the NDF has more than two axes. A group of two strings should be supplied specifying the two axes which are to be used when supplying positions for Parameters CENTRE, LBOUND, and UBOUND. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if you supply an illegal value. If a null (!) value is supplied, the axes with the same indices as the two used pixel axes within the NDF are selected. [!]

Results Parameters:

VALUE = _DOUBLE (Write)

The data value at the top-right pixel in the displayed rectangle.

Examples:

```
look ngc6872 "1:27:23 -22:41:12" logfile=log
```

Lists a 7×7 block of pixel values centred on RA/DEC 1:27:23, −22:41:12 (this assumes that the current co-ordinate Frame in the NDF is an RA/DEC Frame). The listing is written to the text file log.

```
look m57 mode=bo lbound="18 20" ubound="203 241"
```

Lists the pixel values in an NDF called m57, within a rectangular region from pixel (18,20) to (203,241) (this assumes that the current co-ordinate Frame in the NDF is pixel co-ordinates). The listing is displayed on the screen only.

```
look ngc6872 "10 11" 1
```

Stores the value of pixel (10, 11) in output Parameter VALUE, but does not store it in a log file. This assumes that the current co-ordinate Frame in the NDF is pixel co-ordinates.

```
look ngc6872 mode=ard arddesc="circle(1:27:23,-22:41:12,0:0:10)"
```

Lists the pixel values within a circle of radius 10 ard-seconds, centred on RA=1:27:23 DEC=−22:41:12. This assumes that the current co-ordinate Frame in the NDF is an RA/DEC Frame.

```
look ngc6872 mode=ardfile ardfile=central.ard
```

Lists the pixel values specified by the ARD description stored in the text file `central.ard`.

Notes:

- ARD files may be created by ARDGEN or written manually. In the latter case consult SUN/183 for full details of the ARD descriptors and syntax; however, much may be learnt from looking at the ARD files created by ARDGEN and the ARDGEN documentation. There is also a in Section 15.1.1.
- The co-ordinate system in which positions are given within ARD descriptions can be indicated by including suitable COFRAME or WCS statements within the description (see SUN/183). For instance, starting the description with the text `"COFRAME(PIXEL)"` will indicate that positions are specified in pixel co-ordinates. The statement `"COFRAME(SKY,System=FK5)"` would indicate that positions are specified in RA/DEC (FK5,J2000). If no such statements are included, then a default co-ordinate system is used as specified in the parameter description above.
- Output messages are not displayed on the screen when the message filter environment variable MSG_FILTER is set to `QUIET`. The creation of output parameters and the log file is unaffected by MSG_FILTER.

Related Applications :

KAPPA: ARDGEN, ARDMASK, ARDPLOT, TRANDAT.

Implementation Status:

- This routine correctly processes the DATA, QUALITY and VARIANCE components of the input NDF.
- Processing of bad pixels and automatic quality masking are supported.

LUCY
Performs a Richardson-Lucy deconvolution of a one- or two-dimensional array

Description:

This application deconvolves the supplied one- or two-dimensional array using the Richardson-Lucy (R-L) algorithm. It takes an array holding observed data and another holding a Point-Spread Function (PSF) as input and produces an output array with higher resolution. The algorithm is iterative, each iteration producing a new estimate of the restored array which (usually) fits the observed data more closely than the previous estimate (in the sense that simulated data generated from the restored array is closer to the observed data). The closeness of the fit is indicated after each iteration by a normalised χ^2 value (*i.e.* the χ^2 per pixel). The algorithm terminates when the normalised χ^2 given by Parameter AIM is reached, or the maximum number of iterations given by Parameter NITER have been performed. The current estimate of the restored array is then written to the output NDF .

Before the first iteration, the restored array is initialised either to the array given by Parameter START, or, if no array is given, to the difference between the mean value in the input data array and the mean value in the background (specified by Parameters BACK and BACKVAL). Simulated data are then created from this trial array by smoothing it with the supplied PSF, and then adding the background on. The χ^2 value describing the deviation of this simulated data from the observed data are then found and displayed. If the required χ^2 is not reached by this simulated data, the first iteration commences, which consists of creating a new version of the restored array and then creating new simulated data from this new restored array (the corresponding χ^2 value is displayed). Repeated iterations are performed until the required χ^2 is reached, or the iteration limit is reached. The new version of the restored array is created as follows.

(1) A correction factor is found for each data value. This is the ratio of the observed data value to the simulated data value. An option exists to use the Snyder modification as used by the LUCY program in the STSDAS package within IRAF. With this option selected, the variance of the observed data value is added to both the numerator and the denominator when finding the correction factors.

(2) These correction factors are mapped into an array by smoothing the array of correction factors with the transposed PSF.

(3) The current version of the restored array is multiplied by this correction factor array to produce the new version of the restored array.

For further background to the algorithm, see L.B. Lucy, *Astron.J.* 1974, Vol 79, No. 6.

Usage:

```
lucy in psf out [aim]
```

Parameters:

AIM = _REAL (Read)
The χ^2 value at which the algorithm should terminate. Smaller values of AIM will result in higher apparent resolution in the output array but will also cause noise in the observed data to be interpreted as real structure. Small values will require larger number of iterations, so NITER may need to be given a larger value. Very-small values may be completely un-achievable, indicated by χ^2 not decreasing (or sometimes increasing) between iterations. Larger values will result in smoother output arrays with less noise. [1.0]

BACK = NDF (Read)
An NDF holding the background value for each observed data value. If a null value is supplied, a constant background value given by Parameter BACKVAL is used. [!]

BACKVAL = _REAL (Read)
The constant background value to use if BACK is given a null value. [0.0]

CHIFAC = _REAL (Read)
The normalised χ^2 value which is used to determine if the algorithm should terminate is defined as follows:

$$\chi^2 = \frac{1}{N} \cdot \sum \frac{(d-s)^2}{(CHIFAC.s-\sigma^2)}$$

where the sum is taken over the entire input array (excluding the margins used to pad the input array), n is the number of values summed, d is the observed data value, s is the simulated data value based on the current version of the restored array, σ^2 is the variance of the error associated with d, and $CHIFAC$ is the value of Parameter CHIFAC. Using 0 for CHIFAC results in the standard expression for χ^2. However, the algorithm sometimes has difficulty fitting bright features and so may not reach the required normalised χ^2 value. Setting CHIFAC to 1 (as is done by the LUCY program in the STSDAS package within IRAF) causes larger data values to be given less weight in the χ^2 calculation, and so encourages lower χ^2 values. [1.0]

IN= NDF (Read)
The input NDF containing the observed data.

NITER = _INTEGER (Read)
The maximum number of iterations to perform. [50]

OUT = NDF (Write)
The restored output array. The background specified by Parameters BACK and BACKVAL will have been removed from this array. The output is the same size as the input. There is no VARIANCE component in the output, but any QUALITY values are propagated from the input to the output.

PSF = NDF (Read)
An NDF holding an estimate of the Point-Spread Function (PSF) of the input array. This could, for instance, be produced using the KAPPA application PSF. There should be no bad pixels in the PSF otherwise an error will be reported. The PSF can be centred anywhere within the array, but the location of the centre must be specified using Parameters XCENTRE and YCENTRE. The PSF is assumed to have the value zero outside the supplied NDF.

SIGMA = _REAL (Read)
The standard deviation of the noise in the observed data. This is only used if Parameter VARIANCE is given the value FALSE. If a null (!) value is supplied, the value used is an estimate of the noise based on the difference between adjacent pixel values in the observed data. [!]

START = NDF (Read)
An NDF containing an initial guess at the restored array. This could, for instance, be the output from a previous run of LUCY, in which case the deconvolution would continue from the point it had previously reached. If a null value is given, then the restored array is initialised to a constant value equal to the difference between the mean observed data value and the mean background value. [!]

SNYDER = _LOGICAL (Read)
If TRUE then the variance of the observed data sample is added to both the numerator and denominator when evaluating the correction factor for each data sample. This is the modified form of the R-L algorithm used by the LUCY program in the STSDAS package within IRAF. [TRUE]

THRESH = _REAL (Read)
The fraction of the PSF peak amplitude at which the extents of the PSF are determined. These extents are used to determine the size of the margins used to pad the supplied input array. Lower values of THRESH will result in larger margins being used. THRESH must be positive and less than 0.5. [0.0625]

TITLE = LITERAL (Read)
A title for the output NDF. A null (!) value means using the title of the input NDF. [!]

VARIANCE = _LOGICAL (Read)
If TRUE, then the variance of each input data sample will be obtained from the VARIANCE component of the input NDF. An error is reported if this option is selected and the NDF has no VARIANCE component. If FALSE, then a constant variance equal to the square of the value given for Parameter SIGMA is used for all data samples. If a null (!) value is supplied, the value used is TRUE if the input NDF has a VARIANCE component, and FALSE otherwise. [!]

WLIM = _REAL (Read)
If the input array contains bad pixels, then this parameter may be used to determine the number of good data values which must contribute to an output pixel before a valid value is stored in the restored array. It can be used, for example, to prevent output pixels from being generated in regions where there are relatively few good data values to contribute to the restored result. It can also be used to 'fill in' small areas (*i.e.* smaller than the PSF) of bad pixels.

The numerical value given for WLIM specifies the minimum total weight associated with the good pixels in a smoothing box required to generate a good output pixel (weights for each pixel are defined by the normalised PSF). If this specified minimum weight is not present, then a bad output pixel will result, otherwise a smoothed output value will be calculated. The value of this parameter should lie between 0.0 and 1.0. WLIM=0 causes a good output value to be created even if there is only one good input value, whereas WLIM=1 causes a good output value to be created only if all input values are good. Values less than 0.5 will tend to reduce the number of bad pixels, whereas values larger than 0.5 will tend to increase the number of bad pixels.

This threshold is applied each time a smoothing operation is performed. Many smoothing operations are typically performed in a run of LUCY, and if WLIM is larger than 0.5 the effects of bad pixels will propagate further through the array at each iteration. After several iterations this could result in there being no good data left. An error is reported if this happens. [0.001]

XCENTRE = _INTEGER (Read)
The *x* pixel index of the centre of the PSF within the supplied PSF array. If a null (!) value is supplied, the value used is the middle pixel (rounded down if there are an even number of pixels per line). [!]

YCENTRE = _INTEGER (Read)
The *y* pixel index of the centre of the PSF within the supplied PSF array. If a null (!) value is supplied, the value used is the middle line (rounded down if there are an even number of lines). [!]

Examples:

```
lucy m51 star m51_hires
```

This example deconvolves the array in the NDF called m51, putting the resulting array in the NDF called m51_hires. The PSF is defined by the array in NDF star (the centre of the PSF is assumed to be at the central pixel). The deconvolution terminates when a normalised chi-squared value of 1.0 is reached.

```
lucy m51 star m51_hires 0.5 niter=60
```

This example performs the same function as the previous example, except that the deconvolution terminates when a normalised chi-squared value of 0.5 is reached, giving higher apparent resolution at the expense of extra spurious noise-based structure. The maximum number of iterations is increased to 60 to give the algorithm greater opportunity to reach the reduced chi-squared value.

```
lucy m51 star m51_hires2 0.1 start=m51_hires
```

This example continues the deconvolution started by the previous example in order to achieve a normalised chi-squared of 0.1. The output array from the previous example is used to initialise the restored array.

Notes:

- The convolutions required by the R-L algorithm are performed by the multiplication of Fourier transforms. The supplied input array is extended by a margin along each edge to avoid problems of wrap-around between opposite edges of the array. The width of this margin is about equal to the width of the significant part of the PSF (as determined by Parameter THRESH). The application displays the width of these margins. The margins are filled by replicating the edge pixels from the supplied input NDFs.
- The R-L algorithm works best for arrays which have zero background. Non-zero backgrounds cause dark rings to appear around bright, compact sources. To avoid

this a background array should be created before running LUCY and assigned to the Parameter BACK. The SEGMENT and SURFIT applications within KAPPA can be used to create such a background array.

Related Applications :

KAPPA: FOURIER, MEM2D, WIENER.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single-precision floating point.

LUTABLE
Manipulates an graphics device colour table

Description:

This application allows manipulation of the colour table of an graphics device provided some data are, according to the graphics database, already displayed upon the device. A two-dimensional data array, stored in the input NDF structure, may be nominated to assist in defining the colour table via an histogram equalisation. There are two stages to the running of this application.

(1) The way in which the lookup table (LUT) is to distributed amongst the pens (colour indices) of the colour table is required. Some pens are reserved by KAPPA as a palette, particularly for annotation. This application only modifies the unreserved portion of the colour table.

(2) The lookup table is now chosen from a programmed selection or read from an NDF.

The two stages may be repeated cyclically if desired. To exit the loop give the null response, `!`, to a prompt. Looping will not occur if the lookup table and the distribution method are supplied on the command line.

Usage:

```
lutable mapping coltab lut [device] ndf percentiles shade
```

Parameters:

DEVICE = DEVICE (Read)
Name of the graphics device to be used. [Current graphics device]

COLTAB = LITERAL (Read)
The lookup table required. The options are listed below.

`"Negative"` — This is negative grey scale with black assigned to the highest pen, and white assigned to the lowest available pen.

`"Colour"` — This consists of eighteen standard colour blocks.

`"Grey"` — This a standard grey scale.

`"External"` — Obtain a lookup table stored in an NDF's data array. If the table cannot be found in the specified NDF or if it is not a LUT then a grey scale is used.

FULL = _LOGICAL (Read)
If `TRUE` the whole colour-table for the device is stored including the reserved pens. This is necessary to save a colour table written by another package that does not reserve colour indices. For colour tables produced by KAPPA this should be `FALSE`. `[FALSE]`

LUT = NDF (Read)
Name of the NDF containing the lookup table as its data array. The LUT must be two-dimensional, the first dimension being 3, and the second being arbitrary. The method used to compress or expand the colour table if the second dimension is

different from the number of unreserved colour indices is controlled by Parameter NN. Also the LUT's values must lie in the range 0.0–1.0.

MAPPING = LITERAL (Read)
The way in which the colours are to be distributed among the pens. If NINTS is the number of unreserved colour indices the mapping options are described below.

`"Histogram"` — The colours are fitted to the pens using histogram equalisation of an NDF, given by Parameter IN, so that the colours approximately have an even distribution. In other words each pen is used approximately an equal number of times to display the two-dimensional NDF array. There must be an existing graphics deviceed. This is determined by looking for a DATA picture in the database. This is not foolproof as this may be a line plot rather an image.

`"Linear"` — The colours are fitted directly to the pens.

`"Logarithmic"` — The colours are fitted logarithmically to the pens, with colour 1 given to the first available pen and colour NINTS given to the last pen.

NDF = NDF (Read)
The input NDF structure containing the two-dimensional data array to be used for the histogram-equalisation mapping of the pens. The the data object referenced by the last DATA picture in the graphics database is reported. This assumes that the displayed data picture was derived from the nominated NDF data array.

NN = _LOGICAL (Read)
If `TRUE` the input lookup table is mapped to the colour table by using the nearest-neighbour method. This preserves sharp edges and is better for lookup tables with blocks of colour. If NN is `FALSE` linear interpolation is used, and this is suitable for smoothly varying colour tables. `[FALSE]`

PERCENTILES(2) = _REAL (Read)
The percentiles that define the range of the histogram to be equalised. For example, `[25,75]` would scale between the quartile values. It is advisable not to choose the limits less than 3 per cent and greater than 97. The percentiles are only required for histogram mapping. All values in the NDF's data array less than the value corresponding to the lower percentile will have the colour of the first unreserved pen. All values greater than the value corresponding to the upper percentile will have the colour of the last unreserved pen.

SHADE = _REAL (Read)
The type of shading. This only required for the histogram mapping. A value of -1 emphasises low values; $+1$ emphasises high values; `0` is neutral, all values have equal weight. The shade must lie in the range -1 to $+1$.

Examples:

```
lutable lo co
```

Changes the colour table on the current graphics device to a series of coloured blocks whose size increase logarithmically with the table index number.

```
lutable li ex rococo
```

This maps the lookup table stored in the NDF called rococo linearly to the colour table on the current graphics device device.

```
lutable li ex rococo full
```

This maps the lookup table stored in the NDF called rococo linearly to the full colour table on the current graphics device device, *i.e.* ignoring the reserved pens.

```
lutable hi gr ndf=nebula shade=0 percentiles=[5,90]
```

This maps the grey-scale lookup table via histogram equalisation between the 5 and 90 percentiles of an NDF called nebula to the colour table on the current graphics device device. There is no bias or shading to white or black.

Notes:

- The effects of this command will only be immediately apparent when run on X windows which have 256 colours (or other similar pseudocolour devices). On other devices (for instance, X windows with more than 256 colours) the effects will only become apparent when subsequent graphics applications are run.

Related Applications :

KAPPA: LUTEDIT, LUTREAD, LUTSAVE, LUTVIEW; FIGARO: COLOUR.

Implementation Status:

- Processing of bad pixels and automatic quality masking are supported for the image NDF
- All non-complex numeric data types can be handled. Processing is performed using single- or double-precision floating point, as appropriate.

LUTBGYRW
Loads the *BGYRW* lookup table

Description:

This procedure loads the *BGYRW* lookup table with linear scaling into the current graphics device. It is a continuous LUT starting with blue, followed by green, yellow, red and a splash of white.

Usage:

```
lutbgyrw
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTCOL
Loads the standard colour lookup table

Description:

Procedure for loading the standard colour lookup table into the current graphics device with linear scaling.

Usage:

```
lutcol
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTCOLD
Loads the *cold* lookup table

Description:

This procedure loads the *cold* lookup table with linear scaling into the current graphics device. It is a continuous LUT going from black to white, passing through cold shades of pale blue and grey.

Usage:

```
lutcold
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTCONT
Loads a lookup table to give the display the appearance of a contour plot

Description:

This procedure loads a lookup table that gives a contour-plot appearance into the current graphics device. The lookup table is mainly black with a set of white stripes and it is loaded with linear scaling.

Usage:

```
lutcont
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTEDIT
Creates or edits an graphics device colour lookup table

Description:

This application allows a lookup table to be created or edited interactively. The process is controlled through a graphical user interface which presents curves of intensity against pen number, and allows the user to change them in various ways. A specified image is displayed as part of the interface in order to see the effects of the changes. A histogram of pen values is also included. The colour of each pen can be displayed either as red, green and blue intensity, or as hue, saturation and value. More information on the use of the GUI is available through the Help menu within the GUI.

Usage:

```
lutedit lut image device
```

Parameters:

DEVICE = DEVICE (Read)

The name of an graphics device. If a null (`!`) value is supplied for Parameter LUT, then the current LUT associated with the specified device will be loaded into the editor initially. On exit, the final contents of the editor (if saved) are established as the current LUT for the specified device. [Current graphics device]

LUT = NDF (Read)

Name of an exiting colour table to be edited. This should be an NDF containing an array of red, green and blue intensities. The NDF must be two-dimensional, the first dimension being 3, and the second being arbitrary. The method used to compress or expand the colour table if the second dimension is different from the number of unreserved colour indices is controlled by the `"Interpolation"` option in the GUI. If LUT is null (`!`) the current KAPPA colour table for the xwindows graphics display is used. `[!]`

IMAGE = NDF (Read)

Input NDF data structure containing the image to be displayed to show the effect of the created colour table. If a null value is supplied a default image is used.

Related Applications :

KAPPA: LUTABLE, LUTREAD, LUTSAVE, LUTVIEW, PALREAD, PALSAVE; FIGARO: COLOUR.

LUTFC
Loads the standard false-colour lookup table

Description:

This procedure loads the standard false-colour lookup table with linear scaling into the current graphics device.

Usage:

```
lutfc
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTGREY
Loads the standard grey-scale lookup table

Description:

Procedure for loading the standard grey-scale lookup table into the current graphics device with linear scaling.

Usage:

```
lutgrey
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTHEAT
Loads the *heat* lookup table

Description:

This procedure loads the *heat* lookup table with linear scaling into the current graphics device.

Usage:

```
lutheat
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTIKON
Loads the default *Ikon* lookup table

Description:

This procedure loads the default *Ikon* lookup table with linear scaling into the current graphics device.

Usage:

```
lutikon
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

- This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.
- The device need not be an Ikon.

LUTNEG
Loads the standard negative grey-scale lookup table

Description:

Procedure for loading the standard grey-scale lookup table into the current graphics device with negative linear scaling.

Usage:

```
lutneg
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTRAMPS
Loads the coloured-ramps lookup table

Description:

This procedure loads the coloured-ramps lookup table with linear scaling into the current graphics device.

Usage:

`lutramps`

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTREAD
Loads an graphics device lookup table from an NDF

Description:

This application reads a lookup table stored in an NDF with the standard format, and loads it into the current graphics device. device.

Usage:

```
lutread lut
```

Arguments:

LUT = LITERAL (Read)

The file containing the lookup table. It is passed to the Parameter LUT but not validated.

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

LUT = NDF (Read)

Name of the NDF containing the lookup table as its data array. The LUT must be two-dimensional, the first dimension being 3, and the second being arbitrary. Linear interpolation is used to compress or expand the colour table if the second dimension is different from the number of unreserved colour indices. Also the LUT's values must lie in the range 0.0–1.0.

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameters cannot be specified on the command line. You will only be prompted for the parameters if the device is not suitable or not available, and/or the lookup table file could not be accessed.

LUTSAVE
Saves the current colour table of an graphics device in an NDF

Description:

This routine saves the colour table of a nominated graphics device to an NDF LUT file and/or a text file.

Usage:

```
lutsave lut [device]
```

Parameters:

DEVICE = DEVICE (Read)
The name of the graphics device whose colour table is to be saved. [Current graphics device]

FULL = _LOGICAL (Read)
If TRUE the whole colour-table for the device is stored including the reserved pens. This is necessary to save a colour table written by another package that does not reserve colour indices. For colour tables produced by KAPPA this should be FALSE. [FALSE]

LOGFILE = FILENAME (Write)
The name of a text file to receive the formatted values in the colour table. Each line i the file contains the red, green and blue intensities for a single pen, separated by spaces. A null string (!) means that no file is created. [!]

LUT = NDF (Write)
The output NDF into which the colour table is to be stored. Its second dimension equals the number of colour-table entries that are stored. This will be fewer than the total number of colour indices on the device if FULL is FALSE. No NDF is created if a null (!) value is given.

TITLE = LITERAL (Read)
The title for the output NDF. ["KAPPA - Lutsave"]

Examples:

```
lutsave pizza
```

This saves the current colour table on the current graphics device to an NDF called pizza.

```
lutsave redshift full
```

This saves in full the current colour table on the current graphics device to an NDF called redshift.

Related Applications :

KAPPA: LUTEDIT, LUTABLE, LUTREAD.

LUTSPEC
Loads a spectrum-like lookup table

Description:

This procedure loads an optical-spectrum-like lookup table with linear scaling into the current graphics device.

Usage:

`lutspec`

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTVIEW
Draws a colour-table key

Description:

This application displays a key to the current colour table on the specified graphics device using the whole of the current colour table (excluding the low 16 pens which are reserved for axis annotation, *etc.*). The key can either be a simple rectangular block of colour which ramps through the colour table, a histogram-style key in which the width of the block reflects the number of pixels allocated to each colour index, or a set of RGB intensity curves. The choice is made using the STYLE parameter.

By default, numerical data values are displayed along the long edge of the key. The values corresponding to the maximum and minimum colour index are supplied using Parameters HIGH and LOW. Intermediate colour indices are labelled with values which are linearly interpolated between these two extreme values.

The rectangular area in which the key (plus annotations) is drawn may be specified either using a graphics cursor, or by specifying the co-ordinates of two corners using Parameters LBOUND and UBOUND. Additionally, there is an option to make the key fill the current picture. See Parameter MODE. The key may be constrained to the current picture using Parameter CURPIC.

The appearance of the annotation my be controlled in detail using the STYLE parameter.

Usage:

```
lutview [mode] [low] [high] [curpic] [device] lbound=?  ubound=?
```

Parameters:

COMP = LITERAL (Read)
The component (within the NDF given by Parameter NDF) which is currently displayed. It may be `"Data"`, `"Quality"`, `"Variance"`, or `"Error"` (where `"Error"` is an alternative to `"Variance"` and causes the square root of the variance values to be used). If `"Quality"` is specified, then the quality values are treated as numerical values (in the range 0 to 255). The dynamic default is obtained from global Parameter COMP which is set by applications such as DISPLAY. `[]`

CURPIC = _LOGICAL (Read)
If CURPIC is `TRUE`, the colour table key is to lie within the current picture, otherwise the new picture can lie anywhere within the BASE picture. This parameter ignored if the current-picture mode is selected. `[FALSE]`

DEVICE = DEVICE (Read)
The graphics device on which the colour table is to be drawn. [Current graphics device]

FRAME = LITERAL (Read)
Specifies the co-ordinate Frame of the positions supplied using Parameters LBOUND and UBOUND. The following Frames will always be available.

- "GRAPHICS" — gives positions in millimetres from the bottom-left corner of the plotting surface.
- "BASEPIC" — gives positions in a normalised system in which the bottom-left corner of the plotting surface is (0, 0) and the shortest dimension of the plotting surface has length 1.0. The scales on the two axes are equal.
- "CURPIC" — gives positions in a normalised system in which the bottom-left corner of the underlying DATA picture is (0, 0) and the shortest dimension of the picture has length 1.0. The scales on the two axes are equal.
- "NDC" — gives positions in a normalised system in which the bottom-left corner of the plotting surface is (0, 0) and the top-right corner is (1, 1).
- "CURNDC" — gives positions in a normalised system in which the bottom-left corner of the current picture is (0, 0) and the top-right corner is (1, 1).

There may be additional Frames available, describing previously displayed data. If a null value is supplied, the current Frame associated with the displayed data (if any) is used. This parameter is only accessed if Parameter MODE is set to "XY". ["BASEPIC"]

HIGH = _REAL (Read)

The value corresponding to the maximum colour index. It is used to calculate the annotation scale for the key. If it is null (!) the maximum colour index is used, and histogram style keys are not available. [Current display linear-scaling maximum]

LBOUND = LITERAL (Read)

Co-ordinates of the lower-left corner of the rectangular region containing the colour ramp and annotation, in the co-ordinate Frame specified by Parameter FRAME (supplying a colon ":" will display details of the selected co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas. A null (!) value causes the lower-left corner of the BASE or (if CURPIC is TRUE) current picture to be used.

LOW = _REAL (Read)

The value corresponding to the minimum colour index. It is used to calculate the annotation scale for the key. If it is null (!) the minimum colour index is used, and histogram style keys are not available. [Current display linear-scaling minimum]

LUT = NDF (Read)

Name of the NDF containing a lookup table as its data array; the lookup table is written to the graphics device's colour table. The purpose of this parameter is to provide a means of controlling the appearance of the image on certain devices, such as colour printers, that do not have a dynamic colour table, *i.e.* the colour table is reset when the device is opened. If used with dynamic devices, such as windows or Ikons, the new colour table remains after this application has completed. A null, !, means that the existing colour table will be used.

The LUT must be two-dimensional, the first dimension being 3, and the second being arbitrary. The method used to compress or expand the colour table if the second dimension is different from the number of unreserved colour indices is controlled by Parameter NN. Also the LUT's values must lie in the range 0.0–1.0. [!]

MODE = LITERAL (Read)

Method for defining the position, size and shape of the rectangular region containing the colour ramp and annotation. The options are:

- `"Cursor"` — The graphics cursor is used to supply two diametrically opposite corners or the region.
- `"XY"` — The Parameters LBOUND and UBOUND are used to get the limits.
- `"Picture"` — The whole of the current picture is used. Additional positioning options are available by using other KAPPA applications to create new pictures and then specifying the picture mode.

`["Cursor"]`

NDF = NDF (Read)
The NDF defining the image values to be used if a histogram-style key is requested. This should normally be the NDF currently displayed in the most recently created DATA picture. If a value is supplied on the command line for this parameter it will be used. Otherwise, the NDF to used is found by interrogating the graphics database (which contains references to displayed images). If no reference NDF can be obtained from the graphics database, the user will be prompted for a value.

NN = _LOGICAL (Read)
If NN is `TRUE`, the input lookup table is mapped to the colour table by using the nearest-neighbour method. This preserves sharp edges and is better for lookup tables with blocks of colour. If NN is `FALSE`, linear interpolation is used, and this is suitable for smoothly varying colour tables. NN is ignored unless LUT is not null. `[FALSE]`

STYLE = GROUP (Read)
A group of attribute settings describing the plotting style to use for the annotation.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

Axis 1 is always the *data value* axis, whether it is displayed horizontally or vertically. So for instance, to set the label for the data value axis, assign a value to `Label(1)` in the supplied style.

To get a ramp key (the default), specify `"form=ramp"`. To get a histogram key (a coloured histogram of pen indices), specify `"form=histogram"`. To get a graph key (three curves of RGB intensities), specify `"form=graph"`. If a histogram key is produced, the population axis can be either logarithmic or linear. To get a logarithmic population axis, specify `"logpop=1"`. To get a linear population axis, specify

"logpop=0" (the default). To annotate the long axis with pen numbers instead of pixel value, specify "pennums=1" (the default, "pennums=0", shows pixel values). [current value]

UBOUND = LITERAL (Read)

Co-ordinates of the upper-right corner of the rectangular region containing the colour ramp and annotation, in the co-ordinate Frame specified by Parameter FRAME (supplying a colon ":" will display details of the selected co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas. A null (!) value causes the lower-left corner of the BASE or (if CURPIC is TRUE) the current picture to be used.

Examples:

```
lutview
```

Draws an annotated colour table at a position selected via the cursor on the current graphics device.

```
lutview style="form=hist,logpop=1"
```

As above, but the key has the form of a coloured histogram of the pen numbers in the most recently displayed image. The second axis displays the logarithm (base 10) of the bin population.

```
lutview style="form=graph,pennums=1"
```

The key is drawn as a set of three (or one if a monochrome colour table is in use) curves indicating the red, green and blue intensity for each pen. The first axis is annotated with pen numbers instead of data values.

```
lutview style="edge(1)=right,label(1)=Data value in m31"
```

As above, but the data values are labelled on the right edge of the box, and the values are labelled with the string "Data value in m31".

```
lutview style="textlab(1)=0,width(border)=3,colour(border)=white"
```

No textual label is drawn for the data values, and a thicker than usual white box is drawn around the colour ramp.

```
lutview style="textlab(1)=0,numlab(1)=0,majticklen(1)=0"
```

Only the border is drawn around the colour ramp.

```
lutview style="textlab(1)=0,numlab(1)=0,majticklen(1)=0,border=0"
```

No annotation at all is drawn.

```
lutview p
```

Draws a colour table that fills the current picture on the current graphics device.

```
lutview curpic
```

Draws a colour table within the current picture positioned via the cursor.

```
lutview xy lut=my_lut device=ps_p lbound="0.92,0.2" ubound="0.98,0.8"
```

Draws the colour table in the NDF called my_lut with an outline within the BASE picture on the device ps_p, defined by the *x*-*y* bounds (0.92, 0.2) and (0.98, 0.8). In other words the plot is to the right-hand side with increasing colour index with increasing *y* position.

Related Applications :

KAPPA: DISPLAY, LUTABLE, LUTEDIT ; FIGARO: COLOUR.

LUTWARM
Loads the *warm* lookup table

Description:

This procedure loads the *warm* lookup table with linear scaling into the current graphics device. It is a continuous LUT going from black to white, passing through warm shades of yellow and brown.

Usage:

```
lutwarm
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

LUTZEBRA
Loads a pseudo-contour lookup table

Description:

This procedure loads a pseudo-contour lookup table with linear scaling into the current graphics device. The lookup table is mainly black with a set of white stripes.

Usage:

```
lutzebra
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device whose colour table is to be changed. [Current graphics device]

Notes:

This is a procedure that calls LUTABLE. Therefore, the parameter cannot be specified on the command line. You will only be prompted for the DEVICE parameter if the current graphics device is not suitable or not available.

MAKESNR
Creates a signal-to-noise array from an NDF with defined variances

Description:

This application creates a new NDF from an existing NDF by dividing the DATA component of the input NDF by the square root of its VARIANCE component. The DATA array in the output NDF thus measures the signal to noise ratio in the input NDF.

Anomalously small variance values in the input can cause very large spurious values in the output signal to noise array. To avoid this, pixels that have a variance value below a given threshold are set bad in the output NDF.

Usage:

```
makesnr in out [minvar]
```

Parameters:

IN = NDF (Read)
The input NDF. An error is reported if this NDF does not have a VARIANCE component.

MINVAR = _REAL (Read)
The minimum variance value to be used. Input pixels that have variance values smaller than this value will be set bad in the output. The suggested default is determined by first forming a histogram of the logarithm of the input variance values. The highest peak is then found in this histogram. The algorithm then moves down from this peak towards lower variance values until the histogram has dropped to a value equal to the square root of the peak value, or a significant minimum is encountered in the histogram. The corresponding variance value is used as the suggested default. []

OUT = NDF (Write)
The output signal to noise NDF. The VARIANCE component of this NDF will be filled with the value 1.0 (except that bad DATA values will also have bad VARIANCE values).

Examples:

```
makesnr m51 m51_snr
```

This example divides the DATA component of the NDF called m51, by the square root of its own VARIANCE component, rejecting pixels below the default MINVAR value, and writes the resulting signal-to-noise values to an NDF called m51_hires.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.

- The DATA values in the output NDF represent dimensionless ratios, and therefore the UNITS component is not propagated.

MAKESURFACE
Creates a two-dimensional NDF from the coefficients of a polynomial surface

Description:

The coefficients describing a two-dimensional polynomial surface are read from a SURFACEFIT extension in an NDF (written by FITSURFACE), and are used to create a two-dimensional surface of specified size and extent. The surface is written to a new NDF.

The size and extent of the surface may be obtained from a template NDF or given explicitly.

Elements in the new NDF outside the defined range of the polynomial or spline will be set to bad values.

Usage:

```
makesurface in out [like] type=?  lbound=?  ubound=?  xlimit=?  ylimit=?
```

Parameters:

IN = NDF (Read)
The NDF containing the SURFACEFIT extension.

LBOUND(2) = _INTEGER (Read)
Lower bounds of new NDF (if LIKE=!). The suggested defaults are the lower bounds of the IN NDF.

LIKE = NDF (Read)
An optional template NDF which, if specified, will be used to define the labels, size, shape, data type and axis range of the new NDF. If a null response (!) is given, the label, units, axis labels, and axis units are taken from the IN NDF. The task prompts for the data type and bounds, using those of the IN NDF as defaults, and the axis ranges. [!]

OUT = NDF (Write)
The new NDF to contain the surface fit.

TITLE = LITERAL (Read)
A title for the new NDF. If a null response (!) is given, the title will be propagated either from LIKE, or from IN if LIKE=!. [!]

TYPE = LITERAL (Read)
Data type for the new NDF (if LIKE=!). It must be one of the following: "_DOUBLE", "_REAL", "_INTEGER", "_WORD", "_BYTE", "_UBYTE". The suggested default is the data type of the data array in the IN NDF.

UBOUND(2) = _INTEGER (Read)
Upper bounds of new NDF (if LIKE=!). The suggested defaults are the upper bounds of the IN NDF.

VARIANCE = _LOGICAL (Read)
If TRUE, a uniform variance array equated to the mean squared residual of the fit is created in the output NDF, provided the SURFACEFIT structure contains the RMS component. [FALSE]

XLIMIT(2) = _DOUBLE (Read)
Co-ordinates of the left then right edges of the *x* axis (if LIKE=!). The suggested defaults are respectively the minimum and maximum *x* co-ordinates of the IN NDF.

YLIMIT(2) = _DOUBLE (Read)
Co-ordinates of the bottom then top edges of the *y* axis (if LIKE=!). The suggested defaults are respectively the minimum and maximum *y* co-ordinates of the IN NDF.

Examples:

```
makesurface flatin flatout \
```

This generates a two-dimensional image in the NDF called flatout using the surface fit stored in the two-dimensional NDF flatin. The created image has the same data type, bounds, and co-ordinate limits as the data array of flatin.

```
makesurface flatin flatout type=_wo lbound=[1,1] ubound=[320,512]
```

As the previous example, except that the data array in flatout has data type _WORD, and the bounds of flatout are 1:320, 1:512.

```
makesurface flatin flatout like=flatin
```

This has the same effect as the first example, except it has an advantage. If the current co-ordinate system is "Data" and either or both of the axes are inverted (values decrease with increasing pixel index), the output image will be correctly oriented.

```
makesurface flatin flatout like=template title="Surface fit"
```

This generates a two-dimensional image in the NDF called flatout using the surface fit stored in the two-dimensional NDF flatin. The created image inherits the attributes of the NDF called template. The title of flatout is "Surface fit".

Notes:

- The polynomial surface fit is stored in SURFACEFIT extension, component FIT of type POLYNOMIAL, variant CHEBYSHEV or BSPLINE. This extension is created by FITSURFACE. Also read from the SURFACEFIT extension is the co-ordinate system (component COSYS), and the fit RMS (component RMS).
- When LIKE=!, COSYS="Data" or "Axis" and the original NDF had an axis that decreased with increasing pixel index, you may want to flip the co-ordinate limits (via Parameters XLIMIT or YLIMIT) to match the original sense of the axis, otherwise the created surface will be flipped with respect to the image from which it was fitted.

Related Applications :

KAPPA: FITSURFACE, SURFIT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions. However, neither QUALITY nor a SURFACEFIT extension is propagated when LIKE is not null.
- All non-complex numeric data types can be handled. Processing is performed in single- or double-precision floating point, as appropriate.

MANIC
Change the dimensionality of all or part of an NDF

Description:

This application manipulates the dimensionality of an NDF . The input NDF can be projected on to any *n*-dimensional surface (line, plane, *etc.*) by averaging or taking the median the pixels in perpendicular directions, or grown into new dimensions by duplicating an existing *n*-dimensional surface. The order of the axes can also be changed at the same time. Any combination of these operations is also possible.

The shape of the output NDF is specified using Parameter AXES. This is a list of integers, each element of which identifies the source of the corresponding axis of the output—either the index of one of the pixel axes of the input, or a zero indicating that the input should be expanded with copies of itself along that axis. If any axis of the input NDF is not referenced in the AXES list, the missing dimensions will be collapsed to form the resulting data. Dimensions are collapsed by averaging all the non-bad pixels along the relevant pixel axis (or axes).

Usage:

```
manic in out axes
```

Parameters:

AXES( ) = _INTEGER (Read)
An array of integers which define the pixel axes of the output NDF. The array should contain one value for each pixel axis in the output NDF. Each value can be either a positive integer or zero. If positive, it is taken to be the index of a pixel axis within the input NDF which is to be used as the output axis. If zero, the output axis will be formed by replicating the entire output NDF a specified number of times (see Parameters LBOUND and UBOUND). At least one non-zero value must appear in the list, and no input axis may be used more than once.

ESTIMATOR = LITERAL (Read)
The method by which data values in collapsed axes are combined. The permittted options are `"Mean"` to form the average, or `"Median"` to use the median. `["Mean"]`

IN = NDF (Read)
The input NDF.

LBOUND( ) = _INTEGER (Read)
An array holding the lower pixel bounds of any new axes in the output NDF (that is, output axes which have a zero value in the corresponding element of the AXES parameter). One element must be given for each zero-valued element within AXES, in order of appearance within AXES. The dynamic default is to use 1 for every element. `[]`

OUT = NDF (Write)
The output NDF.

TITLE = LITERAL (Read)
Title for the output NDF. A null (`!`) means use the title from the input NDF. `[!]`

UBOUND( ) = _INTEGER (Read)
An array holding the upper pixel bounds of any new axes in the output NDF (that is, output axes which have a zero value in the corresponding element of the AXES parameter). One element must be given for each zero-valued element within AXES, in order of appearance within AXES. The dynamic default is to use 1 for every element. []

Examples:

```
manic image transim [2,1]
```

This transposes the two-dimensional NDF image so that its x pixel co-ordinates are in the y direction and vice versa. The ordering of the axes within the current WCS Frame will only be changed if the Domain of the current Frame is PIXEL or AXES. For instance, if the current Frame has Domain "SKY", with Axis 1 being RA and Axis 2 being DEC, then these will be unchanged in the output NDF. However, the Mapping which is used to relate (RA,DEC) positions to pixel positions will be modified to take the permutation of the pixel axes into account.

```
manic cube summ 3
```

This creates a one dimensional output NDF called summ, in which the single pixel axis corresponds to the z (third) axis in an input NDF called (cube). Each element in the output is equal to the average data value in the corresponding xy plane of the input.

```
manic in=cube out=summ axis=3 estimator=median
```

The same as the previous example, except each output value is equal to the median data value in the corresponding xy plane of the input cube.

```
manic line plane [0,1] lbound=1 ubound=25
```

This takes a one-dimensional NDF called line and expands it into a two-dimensional NDF called plane. The second pixel axis of the output NDF corresponds to the first (and only) pixel axis in the input NDF. The first pixel axes of the output is formed by replicating the the input NDF 25 times.

```
manic line plane [1,0] lbound=1 ubound=25
```

This does the same as the last example except that the output NDF is transposed. That is, the input NDF is copied into the output NDF so that it is parallel to pixel Axis 1 (x) in the output NDF, instead of pixel Axis 2 (y) as before.

```
manic cube hyper [1,0,0,0,0,0,3] ubound=[2,4,2,2,1] accept
```

This manic example projects the second dimension of an input three-dimensional NDF on to the plane formed by its first and third dimensions by averaging, and grows the resulting plane up through five new dimensions with a variety of extents.

Notes:

- This application permutes the NDF pixel axes, and any associated AXIS structures. It does not change the axes of the current WCS co-ordinate Frame, either by permuting, adding or deleting, unless that frame has Domain `"PIXEL"` or `"AXES"`. See the first example in the `"Examples"` section.

Related Applications :

KAPPA: COLLAPSE, PERMAXES.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions. QUALITY is also propagated if possible (*i.e.* if no input axes are collapsed).
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported, up to a maximum of 7.

MATHS
Evaluates mathematical expressions applied to NDF data structures

Description:

This application allows arithmetic and mathematical functions to be applied pixel-by-pixel to a number of NDF data structures and constants so as to produce a new NDF. The operations to be performed are specified using a Fortran-like mathematical expression. Up to 26 each input NDF data and variance arrays, 26 parameterised 'constants', and pixel and data co-ordinates along up to 7 dimensions may be combined in wide variety of ways using this application. The task can also calculate variance estimates for the result when there is at least one input NDF array.

Usage:

```
maths exp out ia-iz=?  va-vz=?  fa-fz=?  pa-pz=?  lbound=?  ubound=?
```

Parameters:

EXP = LITERAL (Read)

The mathematical expression to be evaluated for each NDF pixel, *e.g.* `"(IA-IB+2)*PX"`. In this expression, input NDFs are denoted by the variables IA, IB, ... IZ, while constants may either be given literally or represented by the variables PA, PB, ... PZ. Values for those NDFs and constants which appear in the expression will be requested via the application's parameter of the same name.

Fortran-77 syntax is used for specifying the expression, which may contain the usual intrinsic functions, plus a few extra ones. An appendix in SUN/61 gives a full description of the syntax used and an up to date list of the functions available. The expression may be up to 132 characters long and is case insensitive.

FA-FZ = LITERAL (Read)

These parameters supply the values of 'sub-expressions' used in the expression EXP. Any of the 26 (FA, FB, ... FZ) may appear; there is no restriction on order. These parameters should be used when repeated expressions are present in complex expressions, or to shorten the value of EXP to fit within the 132-character limit. Sub-expressions may contain references to other sub-expressions and constants (PA-PZ). An example of using sub-expressions is:

EXP > `PA*ASIND(FA/PA)*XA/FA`

FA > `SQRT(XA*XA+XB*XB)`

PA > `10.1`

where the parameter name is to the left of > and its value is to the right of the >.

IA-IZ = NDF (Read)

The set of 26 parameters named IA, IB, ... IZ is used to obtain the input NDF data structure(s) to which the mathematical expression is to be applied. Only those parameters which actually appear in the expression are used, and their values are obtained in alphabetical order. For instance, if the expression were `"SQRT(IB+IA)"`, then the Parameters IA and IB would be used (in this order) to obtain the two input NDF data structures.

LBOUND( ) = _INTEGER (Read)
Lower bounds of new NDF, if LIKE=! and there is no input NDF referenced in the expression. The number of values required is the number of pixel co-ordinate axes in the expression.

LIKE = NDF (Read)
An optional template NDF which, if specified, will be used to define bounds and data type of the new NDF, when the expression does not contain a reference to an NDF. If a null response (!) is given the bounds are obtained via Parameters LBOUND and UBOUND, and the data type through Parameter TYPE. [!]

OUT = NDF (Write)
Output NDF to contain the result of evaluating the expression at each pixel.

PA-PZ = _DOUBLE (Read)
The set of 26 parameters named PA, PB, ... PZ is used to obtain the numerical values of any parameterised 'constants' which appear in the expression being evaluated. Only those parameters which actually appear in the expression are used, and their values are obtained in alphabetical order. For instance, if the expression were "PT*SIN(IA/PS)", then the Parameters PS and PT (in this order) would be used to obtain numerical values for substitution into the expression at the appropriate points.

These parameters are particularly useful for supplying the values of constants when writing procedures, where the constant may be determined by a command-language variable, or when the constant is stored in a data structure such as a global parameter. In other cases, constants should normally be given literally as part of the expression, as in "IZ**2.77".

QUICK = _LOGICAL (Read)
Specifies the method by which values for the variance component of the output NDF are calculated. The algorithm used to determine these values involves perturbing each of the input NDF data arrays in turn by an appropriate amount, and then combining the resulting output perturbations. If QUICK is set to TRUE, then each input data array will be perturbed once, in the positive direction only. If QUICK is set to FALSE, then each will be perturbed twice, in the positive and negative directions, and the maximum resultant output perturbation will be used to calculate the output variance. The former approach (the normal default) executes more quickly, but the latter is likely to be more accurate in cases where the function being evaluated is highly non-linear, and/or the errors on the data are large. This parameter is ignored if the expression does not contain a token to at least one input NDF structure. [TRUE]

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the (alphabetically) first input NDF to be used instead. [!]

TYPE = LITERAL (Read)
Data type for the new NDF, if LIKE=! and no input NDFs are referenced in the expression. It must be one either "_DOUBLE" or "_REAL".

UBOUND( ) = _INTEGER (Read)
Upper bounds of new NDF, if LIKE=! and there is no input NDF referenced in the expression. These must not be smaller than the corresponding LBOUND. The number of values required is the number of pixel co-ordinate axes in the expression.

UNITS = _LOGICAL (Read)
Specifies whether the UNITS component of the (alphabetically) first input NDF or the template NDF will be propagated to the output NDF. By default this component is not propagated since, in most cases, the units of the output data will differ from those of any of the input data structures. In simple cases, however, the units may be unchanged, and this parameter then allows the UNITS component to be preserved. This parameter is ignored if the expression does not contain a token to at least one input NDF structure and LIKE=!. [FALSE]

VA-VZ = NDF (Read)
The set of 26 parameters named VA, VB, ...VZ is used to obtain the input NDF variance array(s) to which the mathematical expression is to be applied. The variance VA corresponds to the data array specified by Parameter IA, and so on. Only those parameters which actually appear in the expression, and do not have their corresponding data-array Parameter IA-IZ present, have their values obtained in alphabetical order. For instance, if the expression were "IB+SQRT(VB+VA)", then the Parameters VA and IB would be used (in this order) to obtain the two input NDF data structures. The first would use just the variance array, whilst the second would read both data and variance arrays.

VARIANCE = _LOGICAL (Read)
Specifies whether values for the VARIANCE component of the output NDF should be calculated. If this parameter is set to TRUE (the normal default), then output variance values will be calculated if any of the input NDFs contain variance information. Any which do not are regarded as having zero variance. Variance calculations will normally be omitted only if none of the input NDFs contain variance information. However, if VARIANCE is set to FALSE, then calculation of output variance values will be disabled under all circumstances, with a consequent saving in execution time. This parameter is ignored if the expression does not contain at least one token to an input NDF structure. [TRUE]

Examples:

```
maths "ia-1" dat2 ia=dat1
```

The expression "ia-1" is evaluated to subtract 1 from each pixel of the input NDF referred to as IA, whose values reside in the data structure dat1. The result is written to the NDF structure dat2.

```
maths "(ia-ib)/ic" ia=data ib=back ic=flat out=result units
```

The expression "(ia-ib)/ic" is evaluated to remove a background from an image and to divide it by a flat-field. All the images are held in NDF data structures, the input image being obtained from the data structure data, the background image from back and the flat-field from flat. The result is written to the NDF structure result. The data units are unchanged and are therefore propagated to the output NDF.

```
maths "-2.5*log10(ii)+25.7" ii=file1 out=file2
```

The expression "-2.5*log10(ii)+25.7" is evaluated to convert intensity measurements into magnitudes, including a zero point. Token II represents the input

measurements held in the NDF structure file1. The result is written to the NDF structure file2. If file1 contains variance values, then corresponding variance values will also be calculated for file2.

```
maths exp="pa*exp(ia+pb)" out=outfile pb=13.7 novariance
```

The expression `"pa*exp(ia+pb)"` is evaluated with a value of `13.7` for the constant PB, and output is written to the NDF structure outfile. The input NDF structure to be used for token IA and the value of the other numerical constant PA will be prompted for. `NOVARIANCE` has been specified so that output variance values will not be calculated.

```
maths exp="mod(XA,32)+mod(XB,64)" out=outfile like=comwest
```

The expression `"mod(XA,32)+mod(XB,64)"` is evaluated, and output is written to the NDF structure outfile. The output NDF inherits the shape, bounds, and other properties (except the variance) of the NDF called comwest. The data type of outfile is _REAL unless comwest has type _DOUBLE. XA and XB represent the pixel co-ordinates along the x and y axes respectively.

```
maths "xf*xf+0*xa" ord2 lbound=[-20,10] ubound=[20,50]
```

The expression `"xf*xf+0*xa"` is evaluated, and output is written to the NDF structure ord2. The output NDF has data type _REAL, is two-dimensional with bounds $-20{:}20$, $10{:}50$. The XA is needed to indicate that XF represents pixel co-ordinates along the y axis.

```
maths "xa/max(1,xb)+sqrt(va)" ord2 va=fuzz title="Fuzz correction"
```

The expression `"xa/max(1,xb)+sqrt(va)"` is evaluated, and output is written to the NDF structure ord2. Token VA represents the input variance array held in the NDF structure fuzz. The output NDF inherits the shape, bounds, and other properties of fuzz. The title of ord2 is `"Fuzz correction"`. The data type of ord2 is _REAL unless fuzz has type _DOUBLE. XA and XB represent the pixel co-ordinates along the x and y axes respectively.

Notes:

- The alphabetically first input NDF is regarded as the primary input dataset. NDF components whose values are not changed by this application will be propagated from this NDF to the output. The same propagation rules apply to the LIKE template NDF, except that the output NDF does have inherit any variance information.
- There are additional tokens which can appear in the expression.
  The set of seven tokens named CA, CB, ... CG is used to obtain the data co-ordinates from the primary input NDF data structure. Any of the seven parameters may appear in the expression. The order defines which axis is which, so for example,

`"2*CF+CB*CB"` means the first-axis data co-ordinates squared, plus twice the co-ordinates along the second axis. There must be at least one input NDF in the expression to use the CA-CG tokens, and it must have dimensionality of at least the number of CA-CG tokens given.

The set of seven tokens named XA, XB, ...XG is used to obtain the pixel co-ordinates from the primary input NDF data structure. Any of the seven parameters may appear in the expression. The order defines which axis is which, so for example, `"SQRT(XE)+XC"` means the first-axis pixel co-ordinates plus the square root of the co-ordinates along the second axis. Here no input NDF need be supplied. In this case the dimensionality of the output NDF is equal to the number of XA-XG tokens in the expression. However, if there is at least one NDF in the expression, there should not be more XA-XG tokens than the dimensionality of the output NDF (given as the intersection of the bounds of the input NDFs).

- If illegal arithmetic operations (*e.g.* division by zero, or square root of a negative number) are attempted, then a bad pixel will be generated as a result. (However, the infrastructure software that detects this currently does not work on OSF/1 systems, and therefore MATHS will crash in this circumstance.)
- All arithmetic performed by this application is floating point. Single-precision will normally be used, but double-precision will be employed if any of the input NDF arrays has a numeric type of _DOUBLE.

Calculating Variance :

The algorithm used to calculate output variance values is general-purpose and will give correct results for any reasonably well-behaved mathematical expression. However, this application as a whole, and the variance calculations in particular, are likely to be less efficient than a more specialised application written knowing the form of the mathematical expression in advance. For simple operations (addition, subtraction, *etc.*) the use of other applications (ADD, SUB, *etc.*) is therefore recommended, particularly if variance calculations are required.

The main value of the variance-estimation algorithm used here arises when the expression to be evaluated is too complicated, or too infrequently used, to justify the work of deriving a direct formula for the variance. It is also of value when the data errors are especially large, so that the linear approximation normally used in error analysis breaks down.

There is no variance processing when there are no tokens for input NDF structures.

Timing :

If variance calculations are not being performed, then the time taken is approximately proportional to the number of NDF pixels being processed. The execution time also increases with the complexity of the expression being evaluated, depending in the usual way on the nature of any arithmetic operations and intrinsic functions used. If certain parts of the expression will often give rise to illegal operations (resulting in bad pixels), then execution time may be minimised by placing these operations near the beginning of the expression, so that later parts may not need to be evaluated.

If output variance values are being calculated and the QUICK parameter is set to `TRUE`, then the execution time will be multiplied by an approximate factor (N+1), where n is the number of input NDFs which contain a VARIANCE component. If QUICK is set to `FALSE`, then the execution time will be multiplied by an approximate factor (2N+1).

Related Applications :

KAPPA: CREFRAME, SETAXIS, and numerous arithmetic tasks; FIGARO: numerous arithmetic tasks.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDFs. HISTORY and extensions are propagated from both the primary NDF and template NDF.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- NDFs with any number of dimensions can be processed. The NDFs supplied as input need not all be the same shape.

MEDIAN
Smooths a two-dimensional data array using a weighted median filter

Description:

This task filters the two-dimensional data array in the input NDF structure with a Weighted Median Filter (WMF) in a 3-by-3-pixel kernel to create a new NDF. There are a number of predefined weighting functions and parameters that permit other symmetric weighting functions. See Parameter MODE and the topic "User-defined Weighting Functions".

A threshold for replacement of a value by the median can be set. If the absolute value of the difference between the actual value and the median is less than the threshold, the replacement will not occur. The array boundary is dealt by either pixel replication or a reflection about the edge pixels of the array.

The WMF can be repeated iteratively a specified number of times, or it can be left to iterate continuously until convergence is achieved and no further changes are made to the data. In the latter case a damping algorithm is used if the number of iterations exceeds some critical value, which prevents the result oscillating between two solutions (which can sometimes happen). When damping is switched on data values are replaced not by the median value, but by a value midway between the original and the median.

Bad pixels are not included in the calculation of the median. There is a defined threshold which specifies minimum-allowable median position as a fraction of the median position when there are no bad pixels. For neighbourhoods with too many bad pixels, and so the median position is too small, the resulting output pixel is bad.

Usage:

```
median in out [mode] [diff] [bound] [numit] [corner] [side] [centre]
```

Parameters:

BOUND = LITERAL (Read)

Determines the type of padding required at the array edges before the filtering starts. The alternatives are described below.

`"Replication"` — The values at the edge of the data array are replicated into the padded area. For example, with STEP=2 one corner of the original and padded arrays would appear as follows:

corner of original array:

1	1	1	1	1
1	2	2	2	2
1	2	3	3	3
1	2	3	4	4
1	2	3	4	5

corresponding corner of padded array:

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	2	2	2	2
1	1	1	2	3	3	3
1	1	1	2	3	4	4
1	1	1	2	3	4	5

`"Reflection"` — The values near the edge of the data array are reflected about the array's edge pixels. For example, with STEP=2 one corner of the original and padded arrays would appear as follows:

corner of original array:

1	1	1	1	1
1	2	2	2	2
1	2	3	3	3
1	2	3	4	4
1	2	3	4	5

corresponding corner of padded array:

3	2	1	2	3	3	3
2	2	1	2	2	2	2
1	1	1	1	1	1	1
2	2	1	2	2	2	2
3	2	1	2	3	3	3
3	2	1	2	3	4	4
3	2	1	2	3	4	5

`["Replication"]`

CENTRE = _INTEGER (Read)
Central value for weighting function, required if MODE = −1. It must be an odd value in the range 1 to 21. `[1]`

CORNER = _INTEGER (Read)
Corner value for weighting function, required if MODE = −1. It must be in the range 0 to 10. `[1]`

DIFF = _DOUBLE (Read)
Replacement of a value by the median occurs if the absolute difference of the value and the median is greater than DIFF. `[0.0]`

IN = NDF (Read)
NDF structure containing the two-dimensional data array to be filtered.

ITERATE = LITERAL (Read)
Determines the type of iteration used. The alternatives are described below.

`"Specified"` — You specify the number of iterations at each step size in the Parameter NUMIT.

`"Continuous"` — The filter iterates continuously until convergence is achieved and the array is no longer changed by the filter. A damping algorithm comes into play after MAXIT iterations, and the filter will give up altogether after MAXIT × 1.5 iterations (rounded up to the next highest integer).

`"Continuous"` mode is recommended only for images which are substantially smooth to start with (such as a sky background frame from a measuring machine). Complex images may take many iterations, and a great deal of time, to converge. `["Specified"]`

MAXIT = _INTEGER (Read)
The maximum number of iterations of the filter before the damping algorithm comes into play, when ITERATE = `"Continuous"`. It must lie in the range 1 to 30. `[10]`

MEDTHR = _REAL (Read)
Minimum-allowable actual median position as a fraction of the median position when there are no bad pixels, for the computation of the median at a given pixel. `[0.8]`

MODE = _INTEGER (Read)
Determines type of weighting used, −1 allows you to define the weighting, and 0 to 7 the predefined filters. The predefined modes have the following weighting functions:

0:	1	1	1	**1**:	0	1	0	**2**:	1	0	1	**3**:	1	1	1
	1	1	1		1	1	1		0	1	0		1	3	1
	1	1	1		0	1	0		1	0	1		1	1	1

4:	0	1	0	**5**:	1	0	1	**6**:	1	2	1	**7**:	1	3	1
	1	3	1		0	3	0		2	3	2		3	3	3
	0	1	0		1	0	1		1	2	1		1	3	1

`[0]`

NUMIT = _INTEGER (Read)
The specified number of iterations of the filter, when ITERATE="Specified". `[1]`

OUT = NDF (Write)
NDF structure to contain the two-dimensional data array after filtering.

SIDE = _INTEGER (Read)
Side value for weighting function, required if MODE = −1. It must be in the range 0 to 10. `[1]`

STEP() = _INTEGER (Read)
The spacings between the median filter elements to be used. The data may be filtered at one particular spacing by specifying a single value, such as STEP=4, or may be filtered at a whole series of spacings in turn by specifying a list of values, such as STEP=[4,3,2,1]. There is a limit of 32 values. `[1]`

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
median a100 a100med
```

This applies an equally weighted median filter to the NDF called a100 and writes the result to the NDF a100med. It uses the default settings, which are a single step size of one pixel, and a difference threshold of 0.0. The task pads the array by replication to deals with the edge pixels, and runs the filter once only.

```
median a100 a100med bound=ref
```

As in the previous example except that it uses reflection rather than replication when padding the array.

```
median abc sabc mode=3 step=4 diff=1.0 numit=2
```

This applies a median filter to the NDF called abc with a $\begin{matrix}1&1&1\\1&3&1\\1&1&1\end{matrix}$ weighting mask (MODE=3), a step size of 4 pixels (STEP=4) and a difference threshold of 1.0 (DIFF=1.0). It runs the filter twice (NUMIT=2) and writes the result to the NDF called sabc.

```
median abc sabc mode=3 step=[4,3,2,1] diff=1.0 numit=2
```

This applies a median filter as in the previous example, only this time run the filter at step sizes of 4, 3, 2, and 1 pixels, in that order (STEP=[4,3,2,1]). It runs the filter twice at each step size (NUMIT=2). Note that the filter will be run a total of *eight* times (number of step sizes times the number of iterations).

```
median in=spotty step=[4,3,2,1] iterate=cont maxit=6 out=clean
```

This applies a median filter to the NDF called spotty with the default settings for the mode and difference threshold. It runs the filter at step sizes of 4, 3, 2 and 1 pixels, operating continuously at each step size until the result converges (ITERATE=CONT). Damping will begin after 6 iterations (MAXIT=6), and the filtering will stop regardless after 10 iterations ($1 + \mathrm{INT}(1.5 * \mathrm{MAXIT})$). Note that the filter will run an indeterminate number of times, up to a maximum of 40 (number of step sizes $\times$ maximum number of iterations), and may take a long time. The resultant data array are written to the NDF called clean.

User-defined Weighting Functions :

Parameters CORNER, SIDE, and CENTRE allow other symmetric functions in addition to those offered by MODE=0 to 7. A step size has to be specified too; this determines the spacing of the elements of the weighting function. The data can be filtered at one step size only, or using a whole series of step sizes in sequence. The weighting function has the form:

```
%CORNER  .   %SIDE   .  %CORNER

   .          .          .

 %SIDE   .  %CENTRE  .   %SIDE

   .          .          .

%CORNER  .   %SIDE   .  %CORNER
```

The . Indicates that the weights are separated by the stepsize-minus-one zeros.

Related Applications :

KAPPA: BLOCK, CONVOLVE, FFCLEAN, GAUSMOOTH; ESP: FASTMED; FIGARO: ICONV3, ISMOOTH, IXSMOOTH, MEDFILT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions. VARIANCE is not used to weight the median filter and is not propagated. QUALITY is also lost.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

MEM2D
Performs a Maximum-Entropy deconvolution of a two-dimensional NDF

Description:

MEM2D is based on the Gull and Skilling Maximum Entropy package MEMSYS3. It takes an image and a Point-Spread Function as input and produces an equal-sized image as output with higher resolution. Facilities are provided to 'analyse' the resulting deconvolved image, *i.e.* to calculate an integrated flux in some area of the deconvolved image and also an estimate of the uncertainty in the integrated flux. This allows the significance of structure visible in the deconvolution to be checked.

For a detailed description of the algorithm, and further references, see the MEMSYS users manual, and SUN/117.

Usage:

```
mem2d in out mask=? { fwhmpsf=?
                    { psf=?
                  psftype
```

Parameters:

ANALYSE = _LOGICAL (Read)

ANALYSE should be given a TRUE value if an analysis of a previously generated deconvolution is to be performed, instead of a whole new deconvolution being started. An analysis returns the integrated flux in some area of the deconvolved image you specify, together with the standard deviation on the integrated flux value. The area to be integrated over is specified by an image associated with Parameter MASK. This facility can, for instance, be used to assess the significance of structure seen in the deconvolution. An analysis can only be performed if the input NDF (see Parameter IN) contains a MEM2D extension (see Parameter EXTEND). If the input does contain such an extension, and if the extension shows that the deconvolution was completed, then ANALYSE is defaulted to TRUE, otherwise it is defaulted to FALSE. []

DEF = _REAL (Read)

This is the value to which the output image will default in areas for which there is no valid data in the input. The 'zero entropy' image is defined to be a flat surface with value given by Parameter DEF. Any deviation of the output image away from this image will cause its entropy to become negative. Thus a maximum-entropy criterion causes the output image to be as similar as possible to a flat surface with value DEF (within the constraints of the data). DEF is defaulted to the mean data value in the input image and must always be strictly positive. []

EXTEND = _LOGICAL (Read)

If EXTEND has a TRUE value, then the output NDF will contain an extension called MEM2D which will contain all the information required to either restart or analyse

the deconvolution. Note, including this extension makes the output file much bigger (by about a factor of seven). [TRUE]

FWHMICF = _REAL (Read)
This is the Full Width at Half Maximum (in pixels) of a Gaussian Intrinsic Correlation Function (ICF) to be used in the deconvolution. The ICF can be used to encode prior knowledge of pixel-to-pixel correlations in the output image. A value of 0 for FWHMICF causes no ICF to be used, and so no correlations are expected in the output. Larger values encourage smoothness in the output on the scale of the ICF. If a non-zero ICF is used, the image entropy which is maximised is not the output image, but a 'hidden' image. This hidden image is the deconvolution of the output image with the ICF, and is assumed to have no pixel-to-pixel correlations. [2]

FWHMPSF = _REAL (Read)
This is the Full Width at Half Maximum (in pixels) of a Gaussian Point Spread Function (PSF). This PSF is used to deconvolve the input only if Parameter PSFTYPE has the value "Gaussian".

ILEVEL = _INTEGER (Read)
ILEVEL controls the amount of information displayed as MEM2D runs. If set to 0 then no information is displayed. Larger values up to a maximum of 3, give larger amounts of information. A value of 3 gives full MEMSYS3 diagnostics after each iteration. [1]

IN = NDF (Read)
The input NDF . This can either contain an image to be deconvolved, or the output from a previous run of MEM2D. The NDF is considered to be an output from MEM2D if it contains an extension called MEM2D (see Parameter EXTEND). If such an extension is found, a check is made to see if the NDF contains a completed deconvolution or a partial deconvolution. If the deconvolution is complete, the ANALYSE parameter is defaulted to TRUE, and unless you override this default, an analysis of the deconvolution contained in the input NDF is performed. If the input deconvolution is not complete, then the deconvolution process is restarted from where it left off. If no MEM2D extension is found, then a new deconvolution is started from scratch.

MASK = NDF (Read)
An image to use as a mask to define the areas to be integrated when performing an analysis (see Parameter ANALYSE). The integrated-flux value calculated by the analysis is actually the total data sum in the product of the mask and the deconvolved image. Mask pixel values can be positive or negative (or zero) and so, for instance, masks can be arranged which subtract off a background brightness from a source before returning the integrated source flux.

MODEL = NDF (Read)
An image to use as the default model for the reconstruction. If a null value is given, then a constant value given by the Parameter DEF is used to define a flat default model. The section of the given image which matches the bounds of the input image is used. Any bad pixels in the image cause the corresponding pixels in the input image to be ignored. Such pixels are set bad in the output. The model image should contain no pixels with a value of zero or less. The default model is defined to have zero entropy. The hidden image will tend to the default model in the absence of data. It should be noted that this model applies to the 'hidden' image, not the actually

required reconstructed image. The reconstructed image is obtained from the hidden image by blurring the hidden image with the ICF. [!]

MODELOUT = NDF (Write)
An image which can be used for the default model in a further run of MEM2D. Each pixel value in the created image is a linear combination of the model value at the corresponding pixel in the current reconstruction, and the hidden image pixel value. Pixels for which the hidden image is well away from the current model, tend towards the value of the hidden image; pixels for which the hidden image is close to the current model tend towards the model. Running MEM2D several times, using the new model created on the previous run as the model for the current run, can reduce the 'mottling' often seen in MEM2D reconstructions. [!]

NITER = _INTEGER (Read)
The maximum number of maximum-entropy iterations to perform. MEM2D continues the deconvolution until either MEMSYS3 indicates that the termination criterion ($\Omega = 1.0$) has been reached, or the maximum number of iterations is reached. If a deconvolution requires more iterations than was allowed by NITER, then you can choose to continue the deconvolution by giving the prematurely terminated output from MEM2D as the input to another run of MEM2D, specifying a larger value for NITER. [50]

NOISE = LITERAL (Read)
NOISE defines the noise statistics within the input image. It can take the value "Gaussian" or "Poisson". If Gaussian noise is selected, the data variances are set initially to the values stored in the VARIANCE component of the input NDF. If no such component exists, then the data variances are set to a constant value equal to the RMS difference between adjacent pixels in the x direction. MEMSYS3 scales these initial noise estimates to maximise the data 'evidence'. The evidence is displayed as "LOG(PROB)" and the noise scaling factor as "SIGMA", if Parameter ILEVEL is set to 2 or more. If Poisson statistics are selected the uncertainty in each data value is, as usual, of the order of the square root of the data value. When using Poisson statistics, there is no equivalent to the noise scaling performed when using Gaussian statistics. Any input VARIANCE component is ignored. ["Gaussian"]

OUT = NDF (Write)
The output image in a 'primitive' NDF. The output is the same size as the input. Any pixels which were flagged as bad in the input will also be bad in the output. If Parameter EXTEND is TRUE, then the output NDF contains an extension called MEM2D containing information which allows the deconvolution to be either continued or analysed. There is no VARIANCE component in the output, but any QUALITY values are propagated from the input to the output. If Parameter UPDATE is TRUE, then the output NDF is created after the first iteration and is updated after each subsequent iteration.

PSF = NDF (Read)
An NDF holding an estimate of the Point Spread Function (PSF) of the input image. This PSF is used to deconvolve the input only if Parameter PSFTYPE has the value "NDF". The PSF can be centred anywhere within the image, the location of the centre is specified using Parameters XCENTRE and YCENTRE. The extent of the PSF actually used is controlled by Parameter THRESH.

PSFTYPE = LITERAL (Read)
PSFTYPE determines if the Point Spread Function used in the deconvolution is to be Gaussian (if PSFTYPE="Gaussian"), or is to be defined by an image you supply (if PSFTYPE="NDF"). ["NDF"]

RATE = _REAL (Read)
This is the value to use for the MEMSYS3 RATE parameter. It determines the rate at which the convergence is allowed to proceed. If RATE is high, each maximum-entropy iteration is allowed to make a big change to the current reconstruction. This can cause numeric problems within MEMSYS3 resulting in MEM2D crashing with a "floating overflow" error. If this happens, try reducing RATE. Useful values will normally be of the order of unity, and must lie in the interval 0.0001 to 100. [0.5]

THRESH = _REAL (Read)
The fraction of the PSF peak amplitude at which the extents of the NDF PSF are determined. It must be positive and less than 0.5. This parameter is only used when PSFTYPE="NDF". An error will result if the input PSF is truncated above this threshold. [0.0625]

TITLE = LITERAL (Read)
A title for the output NDF. A null (!) value means using the title of the input NDF. [!]

UPDATE = _LOGICAL (Read)
If UPDATE is given a TRUE value, then the output NDF will be created after the first iteration, and will then be updated after each subsequent iteration. This means that the current reconstruction can be examined without aborting the application. Also, if Parameter EXTEND is TRUE, then if the job aborts for any reason, it can be restarted from the last completed iteration (see Parameter IN). [TRUE]

XCENTRE = _INTEGER (Read)
The *x* pixel index of the centre of the PSF within the supplied PSF image. This is only required if PSFTYPE is "NDF". XCENTRE is defaulted to the middle pixel (rounded down if there are an even number of pixels per line). []

YCENTRE = _INTEGER (Read)
The *y* pixel index (line number) of the centre of the PSF within the supplied PSF image. This is only required if PSFTYPE is "NDF". YCENTRE is defaulted to the middle line (rounded down if there are an even number of lines). []

Results Parameters:

DSUM = _REAL (Write)
The standard deviation of the integrated-flux value calculated if an analysis is performed (see Parameter ANALYSE).

SUM = _REAL (Write)
The integrated-flux value calculated if an analysis is performed (see Parameter ANALYSE).

Examples:
```
mem2d m51 m51_hires psftype=gaussian fwhmpsf=3
```

This example deconvolves the data array in the NDF called m51, putting the resulting image in the data array of the NDF called m51_hires. A circular Gaussian Point-Spread Function is used with a Full Width at Half Maximum of 3 pixels.

```
mem2d m51 m51_hires psf=star xcentre=20 ycentre=20
```

This example performs the same function as the previous example, but the PSF is defined by the data array of the NDF called star, instead of being defined to be Gaussian. This allows the PSF to be any arbitrary two-dimensional function. NDF star could be produced for example, by the KAPPAapplication called PSF. Parameters XCENTRE and YCENTRE give the pixel indices of the centre of the beam defined by the PSF in star. The PSF is truncated to one sixteenth of its peak amplitude.

```
mem2d m51_hires m51_hires niter=70 psf=star
```

If the previous example failed to converge within the default 50 iterations, the deconvolution can be started again from its current state, rather than having to start again from scratch. Here NITER gives the upper limit on the total number of iterations which can be performed (including those performed in the previous run of MEM2D), **not** just the number performed in this single run of MEM2D. This facility can also be used if a MEM2D run is interrupted for any reason, such as the host computer going down, or a batch-queue CPU limit being reached. To use this facility the Parameters EXTEND and UPDATE should have the default values of TRUE.

```
mem2d m51_hires mask=nucleus
```

Once a deconvolved image has been produced, the significance of features seen in the deconvolution can be assessed. This example takes in the NDF m51_hires produced by a previous run of MEM2D. If this is a completed deconvolution then the Parameter ANALYSE will be defaulted to TRUE, and an analysis will be performed. This effectively results in the deconvolution being multiplied by the data array of the NDF called nucleus, and the total data sum in the resulting image being displayed, together with the standard deviation on the total data sum. The image in m51_hires is the most probable deconvolution, but there may be other deconvolutions only slightly less probable than m51_hires. The standard deviation produced by an analysis takes account of the spread between such deconvolutions. If the total data sum is not significantly greater than the standard deviation, then the feature selected by the mask image (called nucleus in this case) may well be spurious. The mask image itself may for instance consist of an area of uniform value $+1$ covering some feature of interest, and the bad value (or equivalently the value zero) everywhere else. The analysis would then give the integrated flux in the feature, assuming that the background is known to be zero. If the background is not zero, then the mask may contain a background region containing the value -1, of equal area to the region containing the value $+1$. The resulting integrated flux would then be the total flux in the source minus the flux in a background region of equal area.

Notes:

- MEM2D requires a large quantity of memory—almost as much as the rest of KAPPA. In order for the KAPPA monolith to load without you having to increase your memory or datasize resources, and because MEM2D is batch oriented (see Timing) it is only available as a separate application.
- Memory is required to store several intermediate images while the deconvolution is in progress. If the input image is small enough, these images are stored in a statically declared, internal array. Otherwise, they are stored in dynamically mapped external arrays. There is no limit on the size of image which can be processed by MEM2D (other than those imposed by limited resources on the host computer).
- It is sometimes desirable for the pixels in the output image to be smaller than those in the input image. For instance, if the input data are critically sampled (two samples per PSF), the output image may not be a very good deconvolution. In such cases sub-dividing the output pixels would give better results. At the moment MEM2D cannot do this. Be warned that sub-dividing the input pixels and then running the current version of MEM2D will not have the same effect, since the noise in the input image will then have pixel-to-pixel correlations, and be interpreted as real structure.

Timing :

MEM deconvolution is extremely CPU intensive. The total CPU time taken depends partly on the size of the image, and partly on the complexity of the structure within the image. As a typical example, a 100×100 image containing 20 Gaussians on a flat background took about 34 minutes of elapsed time on an unloaded DEC Alpha 2000. Deconvolution jobs should therefore always be done in batch. To perform an analysis on a deconvolution takes about the same length of time as a single deconvolution iteration.

Related Applications :

KAPPA: FOURIER, LUCY, WIENER.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported, though only to remove them by the DEF value.
- All non-complex numeric data types can be handled. Arithmetic is performed using single-precision floating point.

MFITTREND
Fits independent trends to data lines that are parallel to an axis

Description:

This routine fits trends to all lines of data in an NDF that lie parallel to a chosen axis. The trends are characterised by polynomials of order up to 15, or by cubic splines. The fits can be restricted to use data that only lies within a series of co-ordinate ranges along the selected axis.

The ranges may be determined automatically. There is a choice of tunable approaches to mask regions to be excluded from the fitting to cater for a variety of data sets. The actual ranges used are reported in the current co-ordinate Frame and pixels, provided they apply to all lines being fitted.

Once the trends have been determined they can either be stored directly or subtracted from the input data. If stored directly they can be subtracted later. The advantage of that approach is the subtraction can be undone, but at some cost in efficiency.

Fits may be rejected if their root-mean squared (rms) residuals are more than a specified number of standard deviations from the the mean rms residuals of the fits. Rejected fits appear as bad pixels in the output data.

Fitting independent trends can be useful when you need to remove the continuum from a spectral cube, where each spectrum is independent of the others (that is you need an independent continuum determination for each position on the sky). It can also be used to de-trend individual spectra and perform functions like debiassing a CCD which has bias strips.

Usage:

```
mfittrend in axis ranges out { order
                             { knots=?
                           fittype
```

Parameters:

AUTO = _LOGICAL (Read)

If TRUE, the ranges that define the trends are determined automatically, and Parameter RANGES is ignored. [FALSE]

AXIS = LITERAL (Read)

The axis of the current co-ordinate system that defines the direction of the trends. This is specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If the axes of the current Frame are not parallel to the NDF pixel axes, then the pixel axis which is most nearly parallel to the specified current Frame axis will be used. AXIS defaults to the last axis. [!]

CLIP() = _REAL (Read)

Array of standard-deviation limits for progressive clipping of outlying binned (see NUMBIN) pixel values while determining the fitting ranges automatically. It is therefore only applicable when AUTO=TRUE. Its purpose is to exclude features that are not part of the trends.

Pixels are rejected at the *i*th clipping cycle if they lie beyond plus or minus CLIP(*i*) times the dispersion about the median of the remaining good pixels. Thus lower values of CLIP will reject more pixels. The normal approach is to start low and progressivley increase the clipping factors, as the dispersion decreases after the exclusion of features. The source of the dispersion depends on the value the METHOD parameter. Between one and five values may be supplied. Supplying the null value (!), results in 2, 2.5, and 3 clipping factors. [2,2,2.5,3]

FITTYPE = LITERAL (Read)

The type of fit. It must be either "Polynomial" for a polynomial or "Spline" for a bi-cubic B-spline. ["Polynomial"]

FOREST = _LOGICAL (Read)

Set this TRUE if the data may contain spectral data with many lines—a line forest—when using the automatic range mode (AUTO=TRUE). A different approach using the histogram determines the baseline mode and noise better in the presence of multiple lines. This leads to improved masking of the spectral lines and affords a better determination of the baseline. In a lineforest the ratio of baseline to line regions is much reduced and hence normal sigma clipping, when FOREST=FALSE, is biased. [FALSE]

KNOTS = _INTEGER (Read)

The number of interior knots used for the cubic-spline fit along the trend axis. Increasing this parameter value increases the flexibility of the surface. KNOTS is only accessed when FITTYPE="Spline". See INTERPOL for how the knots are arranged. The default is the current value.

For INTERPOL=TRUE, the value must be in the range 1 to 11, and 4 is a reasonable value for flatish trends. The initial default is 4.

For INTERPOL=FALSE the allowed range is 1 to 60 with an initial default of 8. In this mode, KNOTS is the maximum number of interior knots.

The upper limit of acceptable values for a trend axis is no more than half of the axis dimension. []

IN = NDF (Read & Write)

The input NDF. On successful completion this may have the trends subtracted, but only if SUBTRACT and MODIFYIN are both set TRUE.

INTERPOL = _LOGICAL (Read)

The type of spline fit to use when FITTYPE="Spline".

If set TRUE, an interpolating spline is fitted by least squares that ensures the fit is exact at the knots. Therefore the knot locations may be set by the POSKNOT parameter.

If set FALSE, a smoothing spline is fitted. A smoothing factor controls the degree of smoothing. The factor is determined iteratively between limits, hence it is the slower option of the two, but generally gives better fits, especially for curvy trends. The location of of the knots is decided automatically by Dierckx's algorithm, governed where they are most needed. Knots are added when the weighted sum of the squared residuals exceeds the smoothing factor. A final fit is made with the chosen smoothing, but finding the knots afresh.

The few iterations commence from the upper limit and progress more slowly at each iteration towards the lower limit. The iterations continue until the residuals stabilise or the maximum number of interior knots is reached or the lower limit is reached. The upper limit is the weighted sum of the squares of the residuals of the least-squares cubic polynomial fit. The lower limit is the estimation of the overall noise obtained from a clipped mean the standard deviation in short segments that diminish bias arising from the shape of the trend. The lower limit prevents too many knots being created and fitting to the noise or fine features.

The iteration to a smooth fit makes a smoothing spline somewhat slower. [FALSE]

MASK = NDF (Write)

The name of the NDF to contain the feature mask. It is only accessed for automatic mode and METHOD="Single" or "Global". It has the same bounds as the input NDF and the data array is type _BYTE. No mask NDF is created if null (!) is supplied. [!]

METHOD = LITERAL (Given)

The method used to define the masked regions in automatic mode. Allowed values are as follows.

- "Region" — This averages trend lines from a selected representative region given by Parameter SECTION and bins neighbouring elements within this average line. Then it performs a linear fit upon the binned line, and rejects the outliers, iteratively with standard-deviation clipping (Parameter CLIP). The standard deviation is that of the average line within the region. The ranges are the intervals between the rejected points, rescaled to the original pixels. They are returned in Parameter ARANGES.
  This is best suited to a low dispersion along the trend axis and a single concentrated region containing the bulk of the signal to be excluded from the trend fitting.
- "Single" — This is like "Region" except there is neither averaging of lines nor a single set of ranges. Each line is masked independently. The dispersion for the clipping of outliers within a line is the standard deviation within that line.
  This is more appropriate when the features being masked vary widely across the image, and significantly between adjacent lines. Some prior smoothing or background tracing (CUPID: FINDBACK) will usually prove beneficial.
- "Global" — This is a variant of "Single". The only difference is that the dispersion used to reject features using the standard deviation of the whole data array. This is more robust than "Single", however it does not perform rejections well for lines with anomalous noise.

["Single"]

MODIFYIN = _LOGICAL (Read)
Whether or not to subtract the trends from the input NDF. It is only used when SUBTRACT is TRUE. If MODIFYIN is FALSE, then an NDF name must be supplied by the OUT parameter. [FALSE]

NUMBIN = _INTEGER (Read)
The number of bins in which to compress the trend line for the automatic range-determination mode. A single line or even the average over a region will often be noisy; this compression creates a better signal-to-noise ratio from which to detect features to be excluded from the trend fitting. If NUMBIN is made too large, weaker features will be lost or stronger features will be enlarged and background elements excluded from the fitting. The minimum value is 16, and the maximum is such that there will be a factor of two compression. NUMBIN is ignored when there are fewer than 32 elements in each line to be de-trended. [32]

ORDER = _INTEGER (Read)
The order of the polynomials to be used when trend fitting. A polynomial of order 0 is a constant and 1 a line, 2 a quadratic *etc.* The maximum value is 15. ORDER is only accessed when FITTYPE="Polynomial". [3]

OUT = NDF (Read)
The output NDF containing either the difference between the input NDF and the various trends, or the values of the trends themselves. This will not be used if SUBTRACT and MODIFYIN are TRUE (in that case the input NDF will be modified).

POSKNOT( ) = LITERAL (Read)
The co-ordinates of the interior knots for all trends. KNOTS values should be supplied, or just the null (!) value to request equally spaced knots. The units of these co-ordinates is determined by the axis of the current world co-ordinate system of the input NDF that corresponds to the trend axis. Supplying a colon ":" will display details of the current co-ordinate Frame. [!]

PROPBAD = _LOGICAL (Read)
Only used if SUBTRACT is FALSE. If PROPBAD is TRUE, the returned fitted values are set bad if the corresponding input value is bad. If PROPBAD is FALSE, the fitted values are retained. [TRUE]

RANGES() = LITERAL (Read)
Pairs of co-ordinates that define ranges along the trend axis. When given these ranges are used to select the values that are used in the fits. The null value (!), causes all the values along each data line to be used. The units of these ranges is determined by the axis of the current world co-ordinate system of the input NDF that corresponds to the trend axis. Supplying a colon ":" will display details of the current co-ordinate Frame. Up to ten pairs of values are allowed. This parameter is not accessed when AUTO=TRUE. [!]

RMSCLIP = _REAL (Read)
The number of standard deviations exceeding the mean of the root-mean-squared residuals of the fits at which a fit is rejected. A null value (!) means perform no rejections. Allowed values are between 2 and 15. [!]

SECTION = LITERAL (Read)
The region from which representative lines are averaged in automatic mode to determine the regions to fit trends. It is therefore only accessed when AUTO=TRUE,

METHOD= "Region", and the dimensionality of the input NDF is more than 1. The value is defined as an NDF section, so that ranges can be defined along any axis, and be given as pixel indices or axis (data) co-ordinates. The pixel axis corresponding to Parameter AXIS is ignored. So for example, if the pixel axis were three in a cube, the value "3:5,4," would average all the lines in elements in Columns 3 to 5 and Row 4. See Section 9 for details.

A null value (!) requests that a representative region around the centre be used. [!]

SUBTRACT = _LOGICAL (Read)

Whether not to subtract the trends from the input NDF or not. If not, then the trends will be evaluated and written to a new NDF (see also Parameter PROPBAD). [FALSE]

TITLE = LITERAL (Read)

Value for the title of the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

VARIANCE = _LOGICAL (Read)

If TRUE and the input NDF contains variances, then the polynomial or spline fits will be weighted by the variances.

Results Parameters:

ARANGES() = _INTEGER (Write)

This parameter is only written when AUTO=TRUE, recording the trend-axis fitting regions determined automatically. They comprise pairs of pixel co-ordinates.

Examples:

```
mfittrend in=cube axis=3 ranges="1000,2000,3000,4000" order=4 out=detrend
```

This example fits cubic polynomials to the spectral axis of a data cube. The fits only use the data lying within the ranges 1000 to 2000 and 3000 to 4000 Ångstroms (assuming the spectral axis is calibrated in Ångstroms and that is the current co-ordinate system). The fit is evaluated and written to the data cube called detrend.

```
mfittrend in=cube axis=3 auto clip=[2,3] order=4 out=detrend
```

As above except the fitting ranges are determined automatically with 2- then 3-sigma clipping.

```
mfittrend in=cube axis=3 auto clip=[2,3] fittype=spline out=detrend
interpol
```

As the previous example except that interpolation cubic-spline fits with four equally spaced interior knots are used to characterise the trends.

```
mfittrend m51 3 out=m51_bsl mask=m51_msk auto fittype=spl
```

This example fits to trends along the third axis of NDF m51 and writes the evaluated fits to NDF m51_bsl. The fits use a smoothing cubic spline with the placement and number of interior knots determined automatically. Features are determined automatically, and a mask of excluded features is written to NDF m51_msk.

```
mfittrend cube axis=3 auto method=single order=1 subtract out=cube_dt
mask=cube_mask
```

This fits linear trends to the spectral axis of a data cube called cube, masking spectral features along each line independently. The mask pixels are recorded in NDF cube_mask. The fitted trend are subtracted and stored in NDF cube_dt.

Notes:

- This application attempts to solve the problem of fitting numerous polynomials in a least-squares sense and that do not follow the natural ordering of the NDF data, in the most CPU-time-efficient way possible.

  To do this requires the use of additional memory (of order one fewer than the dimensionality of the NDF itself, times the polynomial order squared). To minimise the use of memory and get the fastest possible determinations you should not use weighting and assert that the input data do not have any BAD values (use the application SETBAD to set the appropriate flag).
- If you choose to use the automatic range determination. You may need to determine empirically what are the best clipping limits, binning factor, and for METHOD="Region" the region to average.
- You are advised to inspect the fits, especially the spline fits or high-order polynomials. A given set of trends may require more than one pass through this task, if they exhibit varied morphologies. Use masking or NDF sections to select different regions that are fit with different parameters. The various trend maps are then integrated with PASTE to form the final composite set of trends that you can subtract.

Related Applications :

FIGARO: FITCONT, FITPOLY; CCDPACK: DEBIAS; KAPPA: SETBAD.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Handles data of up to 7 dimensions.
- Huge NDFs are supported.

MLINPLOT
Draws a multi-line plot of the data values in a two-dimensional NDF

Description:

This application plots a set of curves giving array value against position in a two-dimensional NDF . All the curves are drawn within a single set of annotated axes. Each curve is displaced vertically by a specified offset to minimise overlap between the curves. These offsets may be chosen automatically or specified by the user (see Parameter SPACE). The curves may be drawn in several different ways such as a `"join-the-dots"` plot, a `"staircase"` plot, a `"chain"` plot (see Parameter MODE).

The data represented by each curve can be either a row or column (chosen using Parameter ABSAXS) of any array component within the supplied NDF (see Parameter COMP). Vertical error bars may be drawn if the NDF contains a VARIANCE component (see Parameter ERRBAR). The vertical axis of the plot represents array value (or the logarithm of the array value—see Parameter YLOG). The horizontal axis represents position, and may be annotated using an axis selected from the Current Frame of the NDF (see Parameter USEAXIS).

Each curve may be labelled using its pixel index or a label specified by the user (see Parameters LINLAB and LABELS). The appearance of these labels (size, colour, fount, horizontal position, *etc.*) can be controlled using Parameter STYLE. A key may be produced to the left of the main plot listing the vertical offsets of the curves (see Parameter KEY). The appearance of the key may be controlled using Parameter KEYSTYLE. Its position may be controlled using Parameter KEYOFF. Markers indicating the zero point for each curve may also be drawn within the main plot (see Parameter ZMARK).

The bounds of the plot on both axes can be specified using Parameters XLEFT, XRIGHT, YBOT, and YTOP. If not specified they take default values which encompass the entire supplied data set. The current picture is usually cleared before plotting the new picture, but Parameter CLEAR can be used to prevent this, allowing several plots to be 'stacked' together. If a new plot is drawn over an existing plot, then the bounds of the new plot are set automatically to the bounds of the existing plot (XLEFT, XRIGHT, YBOT, and YTOP are then ignored).

Usage:

```
mlinplot ndf [comp] lnindx [mode] [xleft] [xright] [ybot] [ytop] [device]
```

Parameters:

ABSAXS = _INTEGER (Read)

This selects whether to plot rows or columns within the NDF. If ABSAXS is `1`, each curve will represent the array values within a single row of pixels within the NDF. If it is `2`, each curve will represent the array values within a single column of pixels within the NDF. `[1]`

AXES = _LOGICAL (Read)

`TRUE` if labelled and annotated axes are to be drawn around the plot. If a null (`!`) value is supplied, `FALSE` is used if the plot is being aligned with an existing plot

(see Parameter CLEAR), and TRUE is used otherwise. Parameters USEAXIS and YLOG determine the quantities used to annotated the horizontal and vertical axes respectively. The width of the margins left for the annotation may be controlled using Parameter MARGIN. The appearance of the axes (colours, founts, *etc.*) can be controlled using the Parameter STYLE. [!]

CLEAR = _LOGICAL (Read)

If TRUE the current picture is cleared before the plot is drawn. If CLEAR is FALSE not only is the existing plot retained, but also the previous plot is used to specify the axis limits. [TRUE]

COMP = LITERAL (Read)

The NDF component to be plotted. It may be "Data", "Quality", "Variance", or "Error" (where "Error" is an alternative to "Variance" and causes the square root of the variance values to be displayed). If "Quality" is specified, then the quality values are treated as numerical values (in the range 0 to 255). ["Data"]

DEVICE = DEVICE (Read)

The plotting device. [current graphics device]

ERRBAR = _LOGICAL (Read)

TRUE if vertical error bars are to be drawn. This is only possible if the NDF contains a VARIANCE component, and Parameter COMP is set to "Data". The length of the error bars (in terms of standard deviations) is set by Parameter SIGMA. The appearance of the error bars (width, colour, *etc.*) can be controlled using Parameter STYLE. See also Parameter FREQ. [FALSE]

FREQ = _INTEGER (Read)

The frequency at which error bars are to be plotted. For instance, a value of 2 would mean that alternate points have error bars plotted. This lets some plots be less cluttered. FREQ must lie in the range 1 to half of the number of points to be plotted. FREQ is only accessed when Parameter ERRBAR is TRUE. [1]

KEY = _LOGICAL (Read)

TRUE if a key giving the offset of each curve is to be produced. The appearance of this key can be controlled using Parameter KEYSTYLE, and its position can be controlled using Parameter KEYPOS. [TRUE]

KEYPOS() = _REAL (Read)

Two values giving the position of the key. The first value gives the gap between the right-hand edge of the multiple-line plot and the left-hand edge of the key (0.0 for no gap, 1.0 for the largest gap). The second value gives the vertical position of the top of the key (1.0 for the highest position, 0.0 for the lowest). If the second value is not given, the top of the key is placed level with the top of the multiple-line plot. Both values should be in the range 0.0 to 1.0. If a key is produced, then the right hand margin specified by Parameter MARGIN is ignored. [current value]

KEYSTYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the key (see Parameter KEY).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which

they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The heading in the key can be changed by setting a value for the Title attribute (the supplied heading is split into lines of no more than 17 characters). The appearance of the heading is controlled by attributes Colour(Title), Font(Title), *etc*. The appearance of the curve labels is controlled by attributes Colour(TextLab), Font(TextLab), *etc.* (the synonym Labels can be used in place of TextLab). The appearance of the offset values is controlled by attributes Colour(NumLab), Font(NumLab), *etc.* (the synonym Offset can be used in place of NumLab). Offset values are formatted using attributes Format(2), *etc.* (the synonym Offset can be used in place of the value 2). [current value]

LABELS = LITERAL (Read)

A group of strings with which to label the plotted curves. A comma-separated list of strings should be given, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. The first string obtained is used as the label for the first curve requested using Parameter LNINDX, the second string is used as the label for the second curve, *etc*. If the number of supplied strings is less than the number of curves requested using LNINDX, then extra default labels are used. These are equal to the NDF pixel index of the row or column, preceded by a hash character ("#"). If a null (!) value is supplied for LABELS, then default labels are used for all curves. [!]

LINLAB = _LOGICAL (Read)

If TRUE, the curves in the plot will be labelled using the labels specified by Parameter LABELS. A single label is placed in-line with the curve. The horizontal position and appearance of these labels can be controlled using Parameter STYLE. [TRUE]

LNINDX = LITERAL (Read)

Specifies the NDF pixel indices of the rows or columns to be displayed (see Parameter ABSAXS). A maximum of 100 lines may be selected. It can take any of the following values.

- "ALL" or "*" — All lines (rows or columns).
- "xx,yy,zz" — A list of line indices.
- "xx:yy" — Line indices between *xx* and *yy* inclusively. When *xx* is omitted the range begins from the lower bound of the line dimension; when *yy* is omitted the

range ends with the maximum value it can take, that is the upper bound of the line dimension or the maximum number of lines this routine can plot.

- Any reasonable combination of above values separated by commas.

MARGIN(4) = _REAL (Read)
The widths of the margins to leave around the multiple-line plot for axis annotation. The widths should be 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. See also Parameter KEYPOS. [current value]

MARKER = _INTEGER (Read)
This parameter is only accessed if Parameter MODE is set to `"Chain"` or `"Mark"`. It specifies the symbol with which each position should be marked, and should be given as an integer PGPLOT marker type. For instance, `0` gives a box, `1` gives a dot, `2` gives a cross, `3` gives an asterisk, `7` gives a triangle. The value must be larger than or equal to -31. [current value]

MODE = LITERAL (Read)
Specifies the way in which each curve is drawn. MODE can take the following values.

- `"Histogram"` — An histogram of the points is plotted in the style of a 'staircase' (with vertical lines only joining the y values and not extending to the base of the plot). The vertical lines are placed midway between adjacent x positions.
- `"Line"` — The points are joined by straight lines.
- `"Point"` — A dot is plotted at each point.
- `"Mark"` — Each point is marker with a symbol specified by Parameter MARKER.
- `"Chain"` — A combination of `"Line"` and `"Mark"`.

[current value]

NDF = NDF (Read)
NDF structure containing the array to be plotted.

OFFSET() = _DOUBLE (Read)
This parameter is used to obtain the vertical offsets for the data curve when Parameter SPACE is given the value `"Free"`. The number of values supplied should equal the number of curves being drawn.

PENS = GROUP (Read)
A group of strings, separated by semicolons, each of which specifies the appearance of a pen to be used to draw a curve. The first string in the group describes the pen to use for the first curve, the second string describes the pen for the second curve, *etc*. If there are fewer strings than curves, then the supplied pens are cycled through again, starting at the beginning. Each string should be a comma-separated list of plotting attributes to be used when drawing the curve. For instance, the string `"width=0.02,colour=red,style=2"` produces a thick, red, dashed curve. Attributes which are unspecified in a string default to the values implied by Parameter STYLE. I f a null value (`!`) is given for PENS, then the pen attributes implied by Parameter STYLE are used. `[!]`

SIGMA = LITERAL (Read)
If vertical error bars are produced (see Parameter ERRBAR), then SIGMA gives the number of standard deviations that the error bars are to represent. [current value]

SPACE = LITERAL (Read)
The value of this parameter specifies how the vertical offset for each data curve is determined. It should be given one of the following values:

- `"Average"` — The offsets are chosen automatically so that the average data values of the curves are evenly spaced between the upper and lower limits of the plotting area. Any line- to-line striping is thus hidden and the amount of overlap of adjacent traces is minimised.
- `"Constant"` — The offsets are chosen automatically so that the zero points of the curves are evenly spaced between the upper and lower limits of the plotting area. The width of any line- to-line strip is constant, which could result in the curves becoming confused if the bias of a curve from its zero point is so large that it overlaps another curve.
- `"Free"` — The offsets to use are obtained explicitly using Parameter OFFSET.
- `"None"` — No vertical offsets are used. All curves are displayed with the same zero point.

The input can be abbreviated to an unambiguous length and is case insensitive. `["Average"]`

STYLE = GROUP (Read)
A group of attribute settings describing the plotting style to use when drawing the annotated axes, data curves, error bars, zero markers, and curve labels.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (`!`)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the data curves is controlled by the attributes Colour(Curves), Width(Curves), *etc.* (the synonym Lines may be used in place of Curves). The appearance of markers used if Parameter MODE is set to `"Point"`, `"Mark"` or `"Chain"` is controlled by Colour(Markers), Width(Markers), *etc.* (the synonym Symbols may be used in place of Markers). The appearance of the error bars is controlled using Colour(ErrBars),

Width(ErrBars), *etc.* (see Parameter ERRBAR). The appearance of the zero-point markers is controlled using Colour(ZeroMark), Size(ZeroMark), *etc*. The appearance of the curve labels is controlled using Colour(Labels), Size(Labels), *etc*. LabPos(Left) controls the horizontal position of the in-line curve label (see Parameter LINLAB), and LabPos(Right) controls the horizontal position of the curve label associated with the right-hand zero-point marker (see Parameter ZMARK). LabPos without any qualifier is equivalent to LabPos(Left). LabPos values are floating point, with `0.0` meaning the left edge of the plotting area, and `1.0` the right edge. Values outside the range 0 to 1 may be used. [current value]

USEAXIS = LITERAL (Read)

The quantity to be used to annotate the horizontal axis of the plot specified by using one of the following options.

- An integer index of an axis within the current co-ordinate Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- An axis Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

The quantity used to annotate the horizontal axis must have a defined value at all points in the array, and must increase or decrease monotonically along the array. For instance, if RA is used to annotate the horizontal axis, then an error will be reported if the profile passes through RA=0 because it will introduce a non-monotonic jump in axis value (from 0h to 24h, or 24h to 0h). If a null (`!`) value is supplied, the value of Parameter ABSAXS is used. `[!]`

XLEFT = LITERAL (Read)

The axis value to place at the left-hand end of the horizontal axis. If a null (`!`) value is supplied, the value used is the first element in the data being displayed. The value supplied may be greater than or less than the value supplied for XRIGHT. A formatted value for the quantity specified by Parameter USEAXIS should be supplied. `[!]`

XRIGHT = LITERAL (Read)

The axis value to place at the right-hand end of the horizontal axis. If a null (`!`) value is supplied, the value used is the last element in the data being displayed. The value supplied may be greater than or less than the value supplied for XLEFT. A formatted value for the quantity specified by Parameter USEAXIS should be supplied. `[!]`

YBOT = _DOUBLE (Read)

The data value to place at the bottom end of the vertical axis. If a null (`!`) value is supplied, the value used is the lowest data value to be displayed, after addition of the vertical offsets. The value supplied may be greater than or less than the value supplied for YTOP. `[!]`

YLOG = _LOGICAL (Read)

`TRUE` if the value displayed on the vertical axis is to be the logarithm of the supplied data values. If `TRUE`, then the values supplied for Parameters YTOP and YBOT should be values for the logarithm of the data value, not the data value itself. `[FALSE]`

YTOP = _DOUBLE (Read)
The data value to place at the top end of the vertical axis. If a null (!) value is supplied, the value used is the highest data value to be displayed, after addition of the vertical offsets. The value supplied may be greater than or less than the value supplied for YBOT. [!]

ZMARK = _LOGICAL (Read)
If TRUE, then a pair of short horizontal lines are drawn at the left and right edges of the main plot for each curve. The vertical position of these lines corresponds to the zero point for the corresponding curve. The right-hand marker is annotated with the curve label (see Parameter LABELS). The appearance of these markers can be controlled using the Parameter STYLE. [TRUE]

Examples:

```
mlinplot rcw3_b1 reset \
```

Plot the first five rows of the two-dimensional NDF file, rcw3_b1 on the current graphics device. The lines are offset such that the averages of the rows are evenly separated in the direction of the vertical axis.

```
mlinplot rcw3_b1 lnindx="1,3,5,7:10" \
```

Plot the rows 1, 3, 5, 7, 8, 9 and 10 of the two-dimensional NDF file, rcw3_b1, on the current graphics device.

```
mlinplot rcw3_b1 lnindx=* \
```

Plot all rows of the two-dimensional NDF file, rcw3_b1, on the current graphics device.

```
mlinplot rcw3_b1 lnindx=* style="colour(curve)=red+width(curve)=4"
```

As the previous example, but the rows are drawn in red at four times normal thickness. The change of line coluor persists to the next invocation, but not the temporary widening of the lines.

```
mlinplot rcw3_b1 lnindx=* style="+width(curve)=4"
```

As the previous example, but now the rows are drawn in the current line colour.

```
mlinplot rcw3_b1 absaxs=2 lnindx="20:25,30,31" \
```

Plot columns 20, 21, 22, 23, 24, 25, 30 and 31 of the two-dimensional NDF file, rcw3_b1, on the current graphics device.

```
mlinplot rcw3_b1 style="Title=CRDD rcw3_b1" \
```

Plot the currently selected rows of the two-dimensional NDF file, rcw3_b1, on the current graphics device. The plot has a title of "CRDD rcw3_b1".

```
mlinplot rcw3_b1(100:500,) ybot=0.0 ytop=1.0E-3 \
```

Plot the currently selected rows of the two-dimensional NDF, rcw3_b1, between column 100 and column 500. The vertical display range is from 0.0 to 1.0E-3.

```
mlinplot rcw3_b1 space=constant device=ps_p \
```

Plot the currently selected rows of the two-dimensional NDF file, rcw3_b1, on the ps_p device. The base lines are evenly distributed over the range of the vertical axis.

```
mlinplot rcw3_b1 space=free offset=[0.,2.0E-4,4.0E-4,6.0E-4,0.1] \
```

Plot the currently selected rows of the two-dimensional NDF file, rcw3_b1. The base lines are set at 0.0 for the first row, 2.0E-4 for the second, 4.0E-4 for the third, 6.0E-4 for the fourth, and 0.1 for the fifth.

Notes:

- If no Title is specified via the STYLE parameter, then the TITLE component in the NDF is used as the default title for the annotated axes. Should the NDF not have a TITLE component, then the default title is instead taken from current co-ordinate Frame in the NDF, unless this attribute has not been set explicitly, whereupon the name of the NDF is used as the default title.
- The application stores a number of pictures in the graphics database in the following order: a FRAME picture containing the annotated axes, data plot, and optional key; a KEY picture to store the key if present; and a DATA picture containing just the data plot. Note, the FRAME picture is only created if annotated axes or a key has been drawn, or if non-zero margins were specified using Parameter MARGIN.

Related Applications :

KAPPA: CLINPLOT, LINPLOT; FIGARO: ESPLOT, IPLOTS, MSPLOT, SPLOT, SPECGRID; SPLAT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, QUALITY, LABEL, TITLE, WCS, and UNITS components of the NDF.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Only double-precision floating-point data can be processed directly. Other non-complex data types will undergo a type conversion before the plot is drawn.

MOCGEN
Creates a Multi-Order Coverage map describing regions of an image

Description:

This application creates a Multi-Order Coverage (MOC) map describing selected regions of the sky, using the scheme described in Version 1.1 of the MOC recommendation published by the International Virtual Observatory Alliance (IVOA).

The regions of sky to be included in the MOC may be specified in several ways (see Parameter MODE).

Usage:

```
mocgen in out mode
```

Parameters:

COMP = LITERAL (Read)

The NDF component to be used if Parameter MODE is `"Good"` or `"Bad"`. It may be `"Data"` or `"Variance"`. `["Data"]`

FORMAT = LITERAL (Read)

The format to use when generating the output MOC specified by Parameter OUT.

- `"FITS"` — The output MOC is stored as a binary-table extension in a FITS file, using the conventions described in Version 1.1 of the IVOA's MOC recommendation.
- `"AST"` — The output MOC is stored as a text file using native AST encoding.
- `"String"` — The output MOC is stored as a text file using the `"string"` encoding described in Version 1.1 of the IVOA's MOC recommendation.
- `"JSON"` — The output MOC is stored as a text file using the JSON encoding described in Version 1.1 of the IVOA's MOC recommendation.

`["FITS"]`

IN = NDF (Read)

The input NDF. Must be two-dimensional.

MAXRES = _REAL (Read)

The size of the smallest cells in the returned MOC, in arcseconds. The nearest of the valid values defined in the MOC recommendation is used. The default value is the largest legal value that results in the cells in the MOC being no larger than the size of the pixels in the centre of the supplied NDF. [!]

MINRES = _REAL (Read)

The size of the largest feature that may be missed in the supplied NDF, in arcseconds. It gives the resolution of the initial grid used to identify areas that are inside the MOC. Bounded 'holes' or 'islands' in the NDF that are smaller than one cell of this initial grid may be missed (i.e. such holes may be 'filled in' and islands omitted in the resulting MOC). The default value is 16 times Parameter MaxRes. [!]

MODE = LITERAL (Read)
The mode used to specify the sky regions to include in the output MOC.

- `"Good"` — The output MOC contains the good pixels in the input NDF specified by Parameter IN.
- `"Bad"` — The output MOC contains the bad pixels in the input NDF specified by Parameter IN.
- `"Qual"` — The output MOC contains the pixels that have the quality specified by Parameter QEXP within the input NDF specified by Parameter IN.

`["Good"]`

OUT = FILENAME (Write)
Name of the file in which to store the MOC description of the selected regions. The format to use is specified by Parameter FORMAT. If Parameter FORMAT is `"FITS"`, `".fits"` is appended to the supplied file name, provided no other file type is included in the supplied string.

QEXP = LITERAL (Read)
The quality expression. Only used if Parameter MOD is `"QUAL"`.

USEAXIS = GROUP (Read)
USEAXIS is only accessed if the current co-ordinate Frame of the NDF has too many axes. A group of strings should be supplied specifying the axes which are to be used. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (`!`) value is supplied, the axes with the same indices as the two used pixel axes within the NDF are used. `[!]`

Examples:

```
mocgen m31 m31.fits
```

Generates a a MOC description of the good pixels in NDF `"m31"`, storing the MOC as a binary table in FITS file `"m31.fits"`.

Related Applications :

KAPPA: REGIONMASK

MSTATS
Calculate statistics over a group of data arrays or points

Description:

This application calculates cumulative statistics over a group of NDFs . It can either generate the statistics of each corresponding pixel in the input array components and output a new NDF with array components containing the result, or calculate statistics at a single point specified in the current co-ordinate Frame of the input NDFs.

In array mode (SINGLE=FALSE), statistics are calculated for each pixel in one of the array components (DATA, VARIANCE or QUALITY) accumulated over all the input NDFs and written to an output NDF; each pixel of the output NDF is a result of combination of pixels with the same Pixel co-ordinates in all the input NDFs. There is a selection of statistics available to form the output values.

The input NDFs must all have the same number of dimensions, but need not all be the same shape. The shape of the output NDF can be set to either the intersection or the union of the shapes of the input NDFs using the TRIM parameter.

In single pixel mode (SINGLE=TRUE) a position in the current co-ordinate Frame of all the NDFs is given, and the value at the pixel covering this point in each of the input NDFs is accumulated to form the results that comprise the mean, variance, and median. These statistics, and if environment variable MSG_FILTER is set to VERBOSE, the value of each contributing pixel, is reported directly to you.

Usage:

```
mstats in out [estimator]
```

Parameters:

CLIP = _REAL (Read)

The number of standard deviations about the mean at which to clip outliers for the "Mode", "Cmean" and "Csigma" statistics (see Parameter ESTIMATOR). The application first computes statistics using all the available pixels. It then rejects all those pixels whose values lie beyond CLIP standard deviations from the mean and will then re-evaluate the statistics. For "Cmean" and "Csigma" there is currently only one iteration, but up to seven for "Mode".

The value must be positive. [3.0]

COMP = LITERAL (Read)

The NDF array component to be analysed. It may be "Data", "Quality", "Variance", or "Error" (where "Error" is an alternative to "Variance" and causes the square root of the variance values to be used). If "Quality" is specified, then the quality values are treated as numerical values (in the range 0 to 255). In cases other than "Data", which is always present, a missing component will be treated as having all pixels set to the 'bad' value. ["Data"]

ESTIMATOR = LITERAL (Read)

The method to use for estimating the output pixel values. It can be one of the

following options. The first four are more for general collapsing, and the remainder are for cube analysis.

- "Mean" — Mean value
- "WMean" — Weighted mean in which each data value is weighted by the reciprocal of the associated variance. (2)
- "Mode" — Modal value. (4)
- "Median" — Median value. Note that this is extremely memory and CPU intensive for large datasets; use with care! If strange things happen, use "Mean". (3)

- "Absdev" — Mean absolute deviation from the unweighted mean. (2)
- "Cmean" — Sigma-clipped mean. (4)
- "Csigma" — Sigma-clipped standard deviation. (4)
- "Comax" — Co-ordinate of the maximum value.
- "Comin" — Co-ordinate of the minimum value.
- "FBad" — Fraction of bad pixel values.
- "FGood" — Fraction of good pixel values.
- "Integ" — Integrated value, being the sum of the products of the value and pixel width in world co-ordinates. Note that for sky co-ordinates the width is measured in radians.
- "Iwc" — Intensity-weighted co-ordinate, being the sum of each value times its co-ordinate, all divided by the integrated value (see the "Integ" option).
- "Iwd" — Intensity-weighted dispersion of the co-ordinate, normalised like "Iwc" by the integrated value. (4)
- "Max" — Maximum value.
- "Min" — Minimum value.
- "FBad" — Fraction of bad pixel values.
- "FGood" — Fraction of good pixel values.
- "NBad" — Count of bad pixel values.
- "NGood" — Count of good pixel values.
- "Rms" — Root-mean-square value. (4)
- "Sigma" — Standard deviation about the unweighted mean. (4)
- "Sum" — The total value.

Where needed, the co-ordinates are the indices of the input NDFs in the supplied order. Thus the calculations behave like the NDFs were stacked one upon another to form an extra axis, and that axis had GRID co-ordinates. Care using wildcards is necessary, to achieve a specific order, say for a time series, and hence assign the desired co-ordinate for a each NDF. Indirection through a text file is recommended.

The selection is restricted if there are only a few input NDFs. For instance, measures of dispersion like "Sigma" and "Iwd" are meaningless for combining only two NDFs. The minimum number of input NDFs for each estimator is given in parentheses in the list above. Where there is no number, there is no restriction. If you supply an unavailable option, you will be informed, and presented with the available options.
["Mean"]

IN = GROUP (Read)
A group of input NDFs. They may have different shapes, but must all have the same number of dimensions. This should be given as a comma-separated list, in which each list element can be one of the following.

- An NDF name, optionally containing wild-cards and/or regular expressions ("*", "?", "[a-z]" *etc.*);
- the name of a text file, preceded by an up-arrow character "^". Each line in the text file should contain a comma-separated list of elements, each of which can in turn be an NDF name (with optional wild-cards, *etc*), or another file specification (preceded by an up-arrow). Comments can be included in the file by commencing lines with a hash character "#".

If the value supplied for this parameter ends with a minus sign "-", then the user is re-prompted for further input until a value is given which does not end with a minus sign. All the images given in this way are concatenated into a single group.

OUT = NDF (Read)
The name of an NDF to receive the results. Each pixel of the DATA (and perhaps VARIANCE) component represents the statistics of the corresponding pixels of the input NDFs. Only used if SINGLE=FALSE.

POS = LITERAL (Read)
In Single pixel mode (SINGLE=TRUE), this parameter gives the position in the current co-ordinate Frame at which the statistics should be calculated (supplying a colon ":" will display details of the required co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas. The pixel covering this point in each input array, if any, will be used.

SINGLE = _LOGICAL (Read)
Whether the statistics should be calculated in Single pixel mode or Array mode. If SINGLE=TRUE, then the POS parameter will be used to get the point to which the statistics refer, but if SINGLE=FALSE an output NDF will be generated containing the results for all the pixels. [FALSE]

TITLE = LITERAL (Read)
Title for the output NDF. ["KAPPA - Mstats"]

TRIM = _LOGICAL (Read)
This parameter controls the shape of the output NDF. If TRIM=TRUE, then the output NDF is the shape of the intersection of all the input NDFs, *i.e.* only pixels which appear in all the input arrays will be represented in the output. If TRIM=FALSE, the output is the shape of the union of the inputs, *i.e.* every pixel which appears in the input arrays will be represented in the output. [TRUE]

VARIANCE = _LOGICAL (Read)
A flag indicating whether a variance array present in the NDF is used to weight the array values while forming the estimator's statistic, and to derive output variance. If VARIANCE is TRUE and all the input NDFs contain a variance array, this array will be used to define the weights, otherwise all the weights will be set equal. [TRUE]

WLIM = _REAL (Read)
If the input NDFs contain bad pixels, then this parameter may be used to determine

at a given pixel location the number of good pixels which must be present within the input NDFs before a valid output pixel is generated. It can be used, for example, to prevent output pixels from being generated in regions where there are relatively few good pixels to contribute to the result of combining the input NDFs.

Results Parameters:

MEAN = _DOUBLE (Write)
The mean pixel value, if SINGLE=TRUE.

MEDIAN = _DOUBLE (Write)
The median pixel value, if SINGLE=TRUE.

VAR = _DOUBLE (Write)
The variance of the pixel values, if SINGLE=TRUE.

Examples:

```
mstats idat* ostats
```

This calculates the mean of each pixel in the Data arrays of all the NDFs in the current directory with names which start "idat", and writes the result in a new NDF called ostats. The shape of ostats will be the intersection of the volumes of all the indat* NDFs.

```
mstats idat* ostats trim=false
```

This does the same as the previous example, except that the output NDF will be the 'union' of the volumes of the input NDFs, that is a cuboid with lower bounds as low as the lowest pixel bound of the input NDFs in each dimension and with upper bounds as high as the highest pixel bound in each dimension.

```
mstats idat* ostats variance
```

This is like the first example except variance information present is used to weight the data values.

```
mstats idat* ostats comp=variance variance
```

This does the same as the first example except that statistics are calculated on the VARIANCE components of all the input NDFs. Thus the pixels of the VARIANCE component of ostats will be the variances of the variances of the input data.

```
mstats m31* single=true pos="0:42:38,40:52:20"
```

This example is analysing the pixel brightness at the indicated sky position in a number of NDFs whose name start with "m31", which all have SKY as their current co-ordinate Frame. The mean and variance of the pixels at that position in all the NDFs are printed to the screen. If the reporting level is verbose, the command also prints the

value of the sampled pixel in each of the NDFs. For those in which the pixel at the selected position is bad or falls outside the NDF, this is also indicated.

```
mstats in="arr1,arr2,arr3" out=middle estimator=median wlim=1.0
```

This example calculates the medians of the DATA components of the three named NDFs and writes them into a new NDF called middle. All input values must be good to form a non-bad output value.

Notes:

- A warning is issued (at the normal reporting level) whenever any output values are set bad because there are too few contributing data values. This reports the fraction of flagged output data generated by the WLIM parameter's threshold.

  No warning is given when Parameter WLIM=0. Input data containing only bad values are not counted in the flagged fraction, since no potential good output value has been lost.
- For SINGLE=TRUE the value of the MSG_FILTER environment variable is used to output messages. If it is QUIET, nothing is reported on the screen. If it is undefined, NORMAL or VERBOSE, the statistics are reported. If it is VERBOSE, the individual pixel values are also reported.

Related Applications :

CCDPACK: MAKEMOS, MAKECAL, MAKEFLAT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, QUALITY, LABEL, TITLE, UNITS, WCS, and HISTORY components of the first input NDF, and propagates all its extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Calculations are performed using the most appropriate of the data types integer, real or double precision. If the input NDFs' structures contain values with other data types, then conversion will be performed as necessary.
- Up to six NDF dimensions are supported.
- Huge NDFs are supported.

MULT
Multiplies two NDF data structures

Description:

The routine multiplies two NDF data structures pixel-by-pixel to produce a new NDF.

Usage:

```
mult in1 in2 out
```

Parameters:

IN1 = NDF (Read)
First NDF to be multiplied.

IN2 = NDF (Read)
Second NDF to be multiplied.

OUT = NDF (Write)
Output NDF to contain the product of the two input NDFs.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN1 to be used instead. `[!]`

Examples:

```
mult a b c
```

This multiplies the NDF called a by the NDF called b, to make the NDF called c. NDF c inherits its title from a.

```
mult out=c in1=a in2=b title="Normalised spectrum"
```

This multiplies the NDF called a by the NDF called b, to make the NDF called c. NDF c has the title `"Normalised spectrum"`.

Notes:

If the two input NDFs have different pixel-index bounds, then they will be trimmed to match before being multiplied. An error will result if they have no pixels in common.

Related Applications :

KAPPA: ADD, CADD, CDIV, CMULT, CSUB, DIV, MATHS, SUB.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.

- All non-complex numeric data types can be handled. Calculations are performed using the most appropriate of the data types integer, real or double precision. If the input NDF structures contain values with other data types, then conversion will be performed as necessary.
- Huge NDFs are supported.

NATIVE
Converts an HDS object to native machine data representation

Description:

This application converts an HDS object (or structure) so that all primitive data values within it are represented using the appropriate native data representation for the machine in use (this includes the appropriate number format and byte ordering). This may typically be required after moving HDS files from another machine which uses a different number format and/or byte order, and will minimise the subsequent access time on the new machine. Conversion is performed by modifying the data *in situ*. No separate output file is produced.

This application can also be used to replace any IEEE floating-point NaN or Inf values in an HDS object with the appropriate Starlink bad value. This conversion is performed even if the data values within the object are already represented using the appropriate native data representation for the machine in use.

Usage:

```
native object
```

Parameters:

OBJECT = UNIVERSAL (Read and Write)

The HDS structure to be converted; either an entire container file or a particular object or structure within the file may be specified. If a structure is given, all components (and sub-components, *etc.*) within it will also be converted.

Examples:

```
native myfile
```

Converts all the primitive data in the HDS container file myfile to be held using the appropriate native machine representation for faster subsequent access.

```
native yourfile.data_array
```

Converts just the DATA_ARRAY component (and its contents, if a structure) in the container file yourfile to the appropriate native machine data representation. Other file contents remain unchanged.

NDFCOMPARE
Compares a pair of NDFs for equivalence

Description:

This application compares two supplied NDFs, and sets the Parameter SIMILAR to `"FALSE"` if they are significantly different in any way, and to `"TRUE"` if they are not significantly different.

If they are not similar, a textual description of the differences is written to standard output, and to any file specified by Parameter REPORT.

The two NDFS are compared in the following ways. Each test has an integer identifier, and the list of tests to be used can be controlled by Parameters DOTESTS and SKIPTESTS. Tests that are not included by default are indicated by the test number being in square brackets. Some tests have parameters that control the exact nature of the test. These are listed in parentheses at the end of the description test listed below.

- 1 — The number of pixel axes are compared.
- 2 — The pixel bounds are compared.
- 3 — The list of co-ordinate systems in the WCS FrameSet are compared.
- 4 — The presence or absence of NDF components are compared (COMP).
- 5 — The sky positions of a grid of pixels are compared (ACCPOS).
- 6 — The data units strings are compared (WHITE).
- 7 — The label strings are compared (CASE,WHITE).
- 8 — The title strings are compared (CASE,WHITE).
- 9 — The data types are compared.
- 10 — The lists of NDF extensions are compared.
- 11 — The number of bad DATA values are compared (NBAD).
- 12 — The number of bad VARIANCE values are compared (NBAD).
- 13 — The pixel DATA values are compared (ACCDAT).
- 14 — The pixel VARIANCE values (if any) are compared (ACCVAR).
- 15 — The pixel QUALITY values (if any) are compared (NBAD).
- 16 — The QUALITY names (if any) are compared.
- [17] — The lists of root ancestor NDFs that were used to create each NDF are compared.

Usage:

```
ndfcompare in1 in2 [report]
```

Parameters:

ACCDAT = LITERAL (Read)
The maximum difference allowed between two pixel data values for them to be considered equivalent. The supplied string should contain a numerical value followed by a single character (case insensitive) from the list below indicating how the numerical value is to be used.

- "V" — The numerical value is a signal-to-noise value. The absolute difference in pixel data value is divided by the square root of the smaller of the two variances associated with the pixels (one from each input NDF). If the resulting ratio is smaller than the ACCDAT value, then the two pixel data values are considered to be equivalent. An error is reported if either NDF does not have a VARIANCE component.
- "R" — The numerical value is a relative error. The absolute difference between the two pixel data values is divided by the absolute mean of the two data values. If the resulting ratio is smaller than the ACCDAT value, then the two pixel data values are considered to be equivalent. To avoid problems with pixels where the mean is close to zero, a lower limit equal to the RMS of the data values is placed on the mean value used in the above ratio.
- "A" — The numerical value is an absolute error. If the absolute difference in pixel data value is smaller than the ACCDAT value, then the two pixel data values are considered to be equivalent.

If no character is included in the ACCDAT string, "R" is assumed. ["1E-6 R"]

ACCPOS = _DOUBLE (Read)
The maximum difference allowed between two axis values for them to be considered equivalent, in units of pixels on the corresponding pixel axes. [0.2]

ACCVAR = LITERAL (Read)
The maximum difference allowed between two pixel variance values for them to be considered equivalent. The supplied string should contain a numerical value followed by a single character (case insensitive) from the list below indicating how the numerical value is to be used.

- "R" — The numerical value is a relative error. The absolute difference in variance value is divided by the absolute mean of the two variance values. If the resulting ratio is smaller than the ACCVAR value, then the two pixel variances are considered to be equivalent.
- "A" — The numerical value is an absolute error. If the absolute difference in variance values is smaller than the ACCVAR value, then the two pixel variances are considered to be equivalent.

If no character is included in the ACCVAR string, "R" is assumed. ["1E-6 R"]

CASE = _LOGICAL (Read)
If TRUE, then string comparisons are case sensitive. Otherwise they are case insensitive. [TRUE]

COMP = _LITERAL (Read)
A comma separated list of the NDF components to include in the test. If a null (!) value is supplied, all NDF components are included. [!]

DOTESTS() = _INTEGER (Read)
An initial list of indices for the tests to be performed, or null (!) if all tests are to be included in the initial list. This initial list is modified by excluding any tests specified by Parameter SKIPTESTS. [!]

IN1 = NDF (Read)
The first NDF.

IN2 = NDF (Read)
The second NDF.

NBAD = LITERAL (Read)
The maximum difference allowed between the number of bad values in each NDF. The same value is used for both DATA and VARIANCE arrays. It is also used as the maximum number of pixel that can have different QUALITY values. The supplied string should contain a numerical value followed by a single character (case insensitive) from the list below indicating how the numerical value is to be used.

- "R" — The numerical value is a relative error. The absolute difference in the number of bad values is divided by the mean number of bad values in both NDFs (for the QUALITY array, the total number of pixels in the NDF is used as the denominator in this ratio). If the resulting ratio is smaller than the NBAD value, then the two NDFs are considered to be equivalent for the purposes of this test.
- "A" — The numerical value is an absolute error. If the absolute difference in the number of bad values is smaller than the NBAD value, then the two NDFs are considered to be equivalent for the purposes of this test.

If no character is included in the NBAD string, "R" is assumed. ["0.001 R"]

REPORT = LITERAL (Read)
The name of a text file to create in which details of the differences found between the two NDFs will be store. [!]

SKIPTESTS() = _INTEGER (Read)
A list of indices for tests that are to removed from the initial list of tests specified by Parameter DOTESTS. If a null (!) value is supplied, the initial list is left unchanged. [15]

SIMILAR = _LOGICAL (Write)
Set to FALSE on exit if any of the used tests indicate that the two NDFs differ.

WHITE = _LOGICAL (Read)
If TRUE, then trailing or leading white space is ignored when comparing strings. [FALSE]

Related Applications :

KAPPA: NDFTRACE, NORMALIZE.

NDFCOMPRESS
Compresses an NDF so that it occupies less disk space

Description:

This application creates a copy of an NDF that occupies less disk space. This compression does not affect the data values seen by subsequent application, since all applications will automatically uncompress the data.

Two compression methods are available: SCALE or DELTA (see Parameter METHOD).

Usage:

```
ndfcompress in out method
```

Parameters:

DSCALE = _DOUBLE (Read)

The scale factor to use for the Data component, when compressing with METHOD set to SCALE. If a null (`!`) value is supplied for DSCALE or DZERO, default values will be used for both that cause the scaled data values to occupy 96% of the available range of the data type selected using Parameter SCALEDTYPE. `[!]`

DZERO = _DOUBLE (Read)

The zero offset to use for the Data component, when compressing with METHOD set to SCALE. If a null (`!`) value is supplied for DSCALE or DZERO, default values will be used for both that cause the scaled data values to occupy 96% of the available range of the data type selected using Parameter SCALEDTYPE. `[!]`

IN = NDF (Read)

The input NDF.

METHOD = LITERAL (Read)

The compression method to use. The options are as follows.

- `"BOTH"` — A lossy compression scheme for all data types. It first creates an intermediate NDF from the supplied NDF using `"SCALED"` compression and then creates the final ouput NDF by applying `"DELTA"` compression to the intermediate NDF. The intermediate NDF is then deleted.
- `"SCALED"` — A lossy compression scheme for all data types. See "Scaled Compression" below, and Parameters DSCALE, DZERO, VSCALE, VZERO, and SCALEDTYPE.
- `"DELTA"` — A lossless compression scheme for integer data types. See "Delta Compression" below, and Parameters ZAXIS, ZMINRATIO, and ZTYPE.

The current value is the default, which is initially `"DELTA"`. `[]`

OUT = NDF (Write)

The output NDF.

SCALEDTYPE = LITERAL (Read)

The data type to use for the scaled data values. This is only used if METHOD is `"SCALED"`. It can be one of the following options.

- "_INTEGER" — four-byte signed integers
- "_WORD" — two-byte signed integers
- "_UWORD" — two-byte unsigned integers
- "_BYTE" — one-byte signed integers
- "_UBYTE" — one-byte unsigned integers

The same data type is used for both DATA and (if required) VARIANCE components of the output NDF. The initial default value is "_WORD". [current value]

VSCALE = _DOUBLE (Read)
The scale factor to use for the VARIANCE component, when compressing with METHOD set to SCALE. If a null (!) value is supplied for VSCALE or VZERO, default values will be used for both that cause the scaled variance values to occupy 96% of the available range of the data type selected using Parameter SCALEDTYPE. [!]

VZERO = _DOUBLE (Read)
The zero factor to use for the VARIANCE component, when compressing with METHOD set to SCALE. If a null (!) value is supplied for VSCALE or VZERO, default values will be used for both that cause the scaled variance values to occupy 96% of the available range of the data type selected using Parameter SCALEDTYPE. [!]

ZAXIS = _INTEGER (Read)
The index of the pixel axis along which differences are to be taken, when compressing with METHOD set to "DELTA". If this is zero, a default value will be selected that gives the greatest compression. [0]

ZMINRATIO = _REAL (Read)
The minimum allowed compression ratio for an array (the ratio of the supplied array size to the compressed array size), when compressing with METHOD set to "DELTA". If compressing an array results in a compression ratio smaller than or equal to ZMINRATIO, then the array is left uncompressed in the new NDF. If the supplied value is zero or negative, then each array will be compressed regardless of the compression ratio. [1.0]

ZTYPE = LITERAL (Read)
The data type to use for storing differences between adjacent uncompressed data values, when compressing with METHOD set to "DELTA". Must be one of _INTEGER, _WORD, _BYTE or blank. If a null (!) value or blank value is supplied, the data type that gives the best compression is determined and used. [!]

Examples:

```
ndfcompress infile outfile scale scaledtype=_uword
```

Copies the contents of the NDF structure infile to the new structure outfile, scaling the values so that they fit into unsigned two-byte integers. The scale and zero values used are chosen automatically.

Scaled Compression :

The SCALE compression method scales the supplied data values using a linear transformation so that they fit into a smaller (integer) data type. A description of the scaling

uses is stored with the output NDF so that later application can reconstruct the original unscaled values. This method is not lossless, due to the truncation involved in converting floating-point values to integers.

Delta Compression :

DELTA compression is lossless, but can only be used on integer values. It assumes that adjacent integer values in the input tend to be close in value, and so differences between adjacent values can be represented in fewer bits than the absolute values themselves. The differences are taken along a nominated pixel axis within the supplied array (specified by Parameter ZAXIS). Any input value that differs from its earlier neighbour by more than the data range of the selected data type is stored explicitly using the data type of the input array.

Further compression is achieved by replacing runs of equal input values by a single occurrence of the value with a corresponding repetition count.

It should be noted that the degree of compression achieved is dependent on the nature of the data, and it is possible for a compressed array to occupy more space than the uncompressed array. The mean compression factor actually achieved is displayed (the ratio of the supplied NDF size to the compressed NDF size).

It is possible to delta compress an NDF that has already been scale compressed. This provides a means of further compressing floating-point arrays. However, note that the default values supplied for DSCALE, DZERO, VSCALE, and VZERO may not be appropriate as they are chosen to maximise the spread of the scaled integer values in order to minimise the integer truncation error, but delta compression works best on arrays of integers in which the spread of values is small.

If the input NDF is already DELTA compressed, it will be uncompressed and then recompressed using the supplied parameter values.

More details of delta compression can be found in SUN/11 (*ARY - A Subroutine Library for Accessing ARRAY Data Structures*), subsection *Delta Compressed Array Form*.

Related Applications :

KAPPA: NDFCOPY.

Implementation Status:

The TITLE, LABEL, UNITS, DATA, VARIANCE, QUALITY, AXIS, WCS, and HISTORY components are copied by this routine, together with all extensions.

NDFCOPY
Copies an NDF (or NDF section) to a new location

Description:

This application copies an NDF to a new location. By supplying an NDF section as input it may be used to extract a subset, or to change the size or dimensionality of an NDF. A second NDF may also be supplied to act as a shape template, and hence to define the region of the first NDF which is to be copied.

Any unused space will be eliminated by the copying operation performed by this routine, so it may be used as a way of compressing NDF structures from which components have been deleted. This ability also makes NDFCOPY a useful alternative to SETBOUND in cases where an NDF's size is to be reduced.

Usage:

```
ndfcopy in out
```

Parameters:

COMP = LITERAL (Read)

The name of an array component in the input NDF (specified by Parameter IN) that will become the DATA_ARRAY in the output NDF (specified by Parameter OUT). It has the following options.

- `"Data"` — Each array component present is propagated to its counterpart.
- `"Variance"` — The VARIANCE component in the input NDF becomes the DATA_ARRAY in the output NDF and retains its data type. The original DATA_ARRAY is not copied.
- `"Error"` — The square root of the VARIANCE component in the input NDF becomes the DATA_ARRAY in the output NDF and retains the VARIANCE's data type. The original DATA_ARRAY and VARIANCE components are not copied.
- `"Quality"` — The QUALITY component in the input NDF becomes the DATA_ARRAY in the output NDF and will be data type _UBYTE. The original DATA_ARRAY and VARIANCE components are not copied.

`["Data"]`

EXTEN = _LOGICAL (Read)

If set to `FALSE` (the default), any NDFs contained within extensions of the input NDF are copied to equivalent places within the output NDF without change. If set `TRUE`, then any extension NDFs which have the same bounds as the base input NDF are padded or trimmed as necessary in order to ensure that they have the same bounds as the output NDF. `[FALSE]`

IN = NDF (Read)

The input NDF (or section) which is to be copied.

LIKE = NDF (Read)
This parameter may be used to supply an NDF to be used as a shape template during the copying operation. If such a template is supplied, then its shape will be used to select a matching section from the input NDF before copying takes place. By default, no template will be used and the shape of the output NDF will therefore match that of the input NDF (or NDF section). The shape of the template in either pixel indices or the current WCS Frame may be used, as selected by Parameter LIKEWCS. [!]

LIKEWCS = _LOGICAL (Read)
If TRUE, then the WCS bounds of the template supplied via Parameter LIKE are used to decide on the bounds of the output NDF. Otherwise, the pixel bounds of the template are used. [FALSE]

OUT = NDF (Write)
The output NDF data structure.

TITLE = LITERAL (Read)
A title for the output NDF. A null value (the default) will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

TRIM = _LOGICAL (Read)
If TRUE, then the number of pixel axes in the output NDF will be reduced if necessary to remove any pixel axes which span only a single pixel. For instance if stokes is a three-dimensional data cube with pixel bounds (1:100,-50:40,1:3), and the Parameter IN is given the value "stokes(,,2)", then the dimensionality of the output depends on the setting of TRIM: if TRIM=FALSE the output is three-dimensional with pixel bounds (1:100,-50:40,2:2) and if TRIM=TRUE the output is two-dimensional with pixel bounds (1:100,-50:40). In this example, the third pixel axis spans only a single pixel and is consequently removed if TRIM=TRUE. [FALSE]

TRIMBAD = _LOGICAL (Read)
If TRUE, then the pixel bounds of the output NDF are trimmed to exclude any border of bad pixels within the input NDF. That is, the output NDF will be the smallest NDF that encloses all good data values in the input NDF. [FALSE]

TRIMWCS = _LOGICAL (Read)
This parameter is only accessed if Parameter TRIM is TRUE. It controls the number of axes in the current WCS co-ordinate Frame of the output NDF. If TRIMWCS=YES, then the current Frame in the output NDF will have the same number of axes as there are pixel axes in the output NDF. If this involves removing axes, then the axes to retain are specified by Parameter USEAXIS. If TRIMWCS=NO then all axes are retained in the current WCS Frame of the output NDF. Using the example in the description of the TRIM parameter, if the input NDF stokes has a three-dimensional current WCS Frame with axes (RA,Dec,Stokes) and TRIMWCS=YES, then an axis will be removed from the current Frame to make it two-dimensional (that is, to match the number of pixel axes remaining after the removal of insignificant pixel axes). The choice of which two axes to retain is controlled by Parameter USEAXIS. If, on the other hand, TRIMWCS was set to FALSE, then the output NDF would still have two pixel axes, but the current WCS Frame would retain all three axes from the input NDF. If one or more current-Frame axes are removed, the transformation from the current Frame to pixel Frame may become undefined resulting in some WCS operations being unusable. The inverse of this transformation (from pixel Frame to current Frame) is unchanged however. [TRUE]

USEAXIS = LITERAL (Read)

This parameter is only accessed if TRIM and TRIMWCS are both `TRUE` and some axes need to be removed from the current WCS Frame of the output NDF. It gives the axes which are to be retained in the current WCS Frame of the output NDF. Each axis can be specified using one of the following options.

- An integer index of an axis within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- An axis Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

The dynamic default selects the axes with the same indices as the pixel axes being copied. The value should be given as a comma-separated list. `[]`

Examples:

```
ndfcopy infile outfile
```

Copies the contents of the NDF structure infile to the new structure outfile. Any unused space will be eliminated during the copying operation.

```
ndfcopy infile outfile comp=var
```

As the previous example except that the VARIANCE component of NDF infile becomes the DATA_ARRAY of NDF outfile.

```
ndfcopy in=data1(3:40,-3:17) out=data2 title="Extracted section"
```

Copies the section (3:40,-3:17) of the NDF called data1 to a new NDF called data2. The output NDF is assigned the new title `"Extracted section"`, which replaces the title derived from the input NDF.

```
ndfcopy galaxy newgalaxy like=oldgalaxy
```

Copies a section of the NDF called galaxy to form a new NDF called newgalaxy. The section which is copied will correspond in shape with the template oldgalaxy. Thus, after the copying operation, both newgalaxy and oldgalaxy will have the same pixel-index bounds.

```
ndfcopy aa(20~11,20~11) bb like=aa
```

Copies from the NDF section consisting of an 11×11 pixel region of aa centred on pixel (20,20), into a new NDF called bb. The shape of the region copied is made to match the original shape of aa. The effect is to extract the selected square region of pixels into a new NDF of the same shape as the original, setting the surrounding region to the bad-pixel value.

```
ndfcopy survey(12h23m:12h39m,11d:13d50m,) virgo trimwcs trim
```

Copies a section specified by equatorial co-ordinate ranges from the three-dimensional NDF called survey, whose third pixel axis has only one element, to a two-dimensional NDF called virgo. Information on the third WCS axis is removed too.

Related Applications :

KAPPA: SETBOUND; FIGARO: ISUBSET.

Implementation Status:

- If present, an NDF's TITLE, LABEL, UNITS, DATA, VARIANCE, QUALITY, AXIS WCS, and HISTORY components are copied by this routine, together with all extensions. The output NDF's title may be modified, if required, by specifying a new value via the TITLE parameter.
- Huge NDFs are supported.

NDFECHO
Displays a group of NDF names

Description:

This application lists the names of the supplied NDFs to the screen, optionally filtering them using a regular expression. Its primary use is within scripts that need to process groups of NDFs. Instead of the full name, a required component of the name may be displayed instead (see Parameter SHOW).

Two modes are available.

- If the NDFs are specified via the NDF parameter, then the NDFs must exist and be accessible (an error is reported otherwise). The NDF names obtained can then be modified by supplying a suitable GRP modification expression such as `"*_A"` for Parameter MOD.
- To list NDFs that may not exist, supply a null (`!`) value for Parameter NDF and the main group expression to Parameter MOD.

Usage:

```
ndfecho ndf [mod] [first] [last] [show]
```

Parameters:

ABSPATH = _LOGICAL (Read)
If `TRUE`, any relative NDF paths are converted to absolute, using the current working directory. `[FALSE]`

EXISTS = _LOGICAL (Read)
If `TRUE`, then only display paths for NDFs specified by Parameter MOD that actually exist and are accessible. `[FALSE]`

FIRST = _INTEGER (Read)
The index of the first NDF to be tested. A null (`!`) value causes the first NDF to be used (Index 1). `[!]`

LAST = _INTEGER (Read)
The index of the last NDF to be tested. If a non-null value is supplied for FIRST, then the run-time default for LAST is equal to the supplied FIRST value (so that only a single NDF will be tested). If a null value is supplied for FIRST, then the run-time default for LAST is the last NDF in the supplied group. `[]`

LOGFILE = FILENAME (Write)
The name of a text file in which to store the listed NDF names. If a null (!) value is supplied, no log file is created. `[!]`

MOD = LITERAL (Read)
An optional GRP modification expression that will be used to modify any names obtained via the NDF parameter. For instance, if MOD is `"*_A"` then the supplied NDF names will be modified by appending `"_A"` to them. No modification occurs if a null (`!`) value is supplied.

If a null value is supplied for Parameter NDF then the value supplied for Parameter MOD should not include an asterisk, since there are no names to be modified. Instead, the MOD value should specify an explicit group of NDF names do not need to exist.
The list can be filtered to remove any NDFs that do not exist (see Parameter EXISTS). `[!]`

NDF = NDF (Read)
A group of existing NDFs. This should be given as a comma-separated list, in which each list element can be one of the following options.

- An NDF name, optionally containing wild-cards and/or regular expressions (`"*"`, `"?"`, `"[a-z]"` *etc.*).
- The name of a text file, preceded by an up-arrow character `"^"`. Each line in the text file should contain a comma-separated list of elements, each of which can in turn be an NDF name (with optional wild-cards, *etc.*), or another file specification (preceded by an up-arrow). Comments can be included in the file by commencing lines with a hash character `"#"`.

If the value supplied for this parameter ends with a hyphen, then you are re-prompted for further input until a value is given which does not end with a hyphen. All the NDFs given in this way are concatenated into a single group.
If a null (`!`) value is supplied, then the displayed list of NDFs is determined by the value supplied for the MOD parameter.

PATTERN = LITERAL (Read)
Specifies a pattern matching template using the syntax described below in "Pattern Matching Syntax". Each NDF is displayed only if a match is found between this pattern and the item specified by Parameter SHOW. A null (`!`) value causes all NDFs to be displayed. `[!]`

SHOW = LITERAL (Read)
Specifies the information to be displayed about each NDF. The options are as follows.

- `"Base"` — The base file name.
- `"Dir"` — The directory path (if any).
- `"Fspec"` — The directory, base name and file type concatenated to form a full file specification.
- `"Ftype"` — The file type (usually `.sdf` but may not be if any foreign NDFs are supplied).
- `"HDSpath"` — The HDS path within the container file (if any).
- `"Path"` — The full name of the NDF as supplied by the user.
- `"Slice"` — The NDF slice specification (if any).

Note, the fields are extracted from the NDF names as supplied by the user. No missing fields (except for `"Ftype"`) are filled in. `["Path"]`

Results Parameters:

NMATCH = _INTEGER (Write)
An output parameter to which is written the number of NDFs between FIRST and LAST that match the pattern supplied by Parameter PATTERN.

SIZE = _INTEGER (Write)
An output parameter to which is written the total number of NDFs in the specified group.

VALUE = LITERAL (Write)
An output parameter to which is written information about the NDF specified by Parameter FIRST, or the first NDF in the group if FIRST is not specified. The information to write is specified by the SHOW parameter.

Examples:

```
ndfecho mycont
```

Report the full path of all the NDFs within the HDS container file `mycont.sdf`. The NDFs must all exist.

```
ndfecho ^files.lis first=4 show=base
```

This reports the file base name for just the fourth NDF in the list specified within the text file `files.lis`. The NDFs must all exist.

```
ndfecho ^files.lis *_a logfile=log.lis
```

This reports the names of the NDFs listed in text file `files.lis`, but appending `"_a"` to the end of each name. The NDFs must all exist. The listed NDF names are written to a new text file called `log.lis`.

```
ndfecho in=!  mod={^base}|_a|_b|
```

This reports the names of the NDFs listed in text file `base`, but replacing `"_a"` with `"_b"` in their names. The NDFs need not exist since they are completely specified by Parameter MOD and not by Parameter NDF.

Pattern Matching Syntax :

The syntax for the PATTERN parameter value is a minimal form of regular expression. The following atoms are allowed.

- `"[chars]"` — Matches any of the characters within the brackets.
- `"[^chars]"` — Matches any character that is not within the brackets (ignoring the initial `"^"` character).
- `"."` — Matches any single character.
- `"\d"` — Matches a single digit.
- `"\D"` — Matches anything but a single digit.
- `"\w"` — Matches any alphanumeric character, and `"_"`.
- `"\W"` — Matches anything but alphanumeric characters, and `"_"`.
- `"\s"` — Matches white space.
- `"\S"` — Matches anything but white space.

Any other character that has no special significance within a regular expression matches itself. Characters that have special significance can be matched by preceding them with a backslash (\) in which case their special significance is ignored (note, this does not apply to the characters in the set dDsSwW).

Note, minus signs ("-") within brackets have no special significance, so ranges of characters must be specified explicitly.

The following quantifiers are allowed.

- "*" — Matches zero or more of the preceding atom, choosing the largest possible number that gives a match.
- "*?"— Matches zero or more of the preceding atom, choosing the smallest possible number that gives a match.
- "+" — Matches one or more of the preceding atom, choosing the largest possible number that gives a match.
- "+?"— Matches one or more of the preceding atom, choosing the smallest possible number that gives a match.
- "?" — Matches zero or one of the preceding atom.
- "{n}" — Matches exactly *n* occurrences of the preceding atom.

The following constraints are allowed.

- "^" — Matches the start of the test string.
- "$" — Matches the end of the test string.

Multiple templates can be concatenated, using the "|" character to separate them. The test string is compared against each one in turn until a match is found.

NDFTRACE
Displays the attributes of an NDF data structure

Description:

This routine displays the attributes of an NDF data structure including:

- its name;
- the values of its character components (title, label, and units);
- its shape (pixel bounds, dimension sizes, number of dimensions and total number of pixels);
- axis co-ordinate information (axis labels, units and extents);
- optionally, axis array attributes (type and storage form) and the values of the axis normalisation flags;
- attributes of the main data array and any other array components present (including the type and storage form and an indication of whether 'bad' pixels may be present);
- attributes of the current co-ordinate Frame in the WCS component (title, domain, and, optionally, axis labels and axis units, plus the system epoch and projection for sky co-ordinate Frames). In addition the bounding box of the NDF within the Frame is displayed.
- optionally, attributes of all other co-ordinate Frames in the WCS component.
- a list of any NDF extensions present, together with their data types; and
- history information (creation and last-updated dates, the update mode and the number of history records).

Most of this information is output to parameters.

Usage:

```
ndftrace ndf
```

Parameters:

FULLAXIS = _LOGICAL (Read)

If the NDF being examined has an axis co-ordinate system defined, then by default only the label, units and extent of each axis will be displayed. However, if a `TRUE` value is given for this parameter, full details of the attributes of all the axis arrays will also be given. `[FALSE]`

FULLFRAME = _LOGICAL (Read)

If a `FALSE` value is given for this parameter then only the Title and Domain attributes are displayed for a co-ordinate Frame. Otherwise, a more complete description is given. `[FALSE]`

FULLWCS = _LOGICAL (Read)

If a `TRUE` value is given for this parameter then all co-ordinate Frames in the WCS component of the NDF are displayed. Otherwise, only the current co-ordinate Frame is displayed. `[FALSE]`

NDF = NDF (Read)
The NDF data structure whose attributes are to be displayed.

Results Parameters:

AEND( ) = _DOUBLE (Write)
The axis upper extents of the NDF. For non-monotonic axes, zero is used. See Parameter AMONO. This is not assigned if AXIS is FALSE.

AFORM( ) = LITERAL (Write)
The storage forms of the axis centres of the NDF. This is only written when FULLAXIS is TRUE and AXIS is TRUE.

ALABEL( ) = LITERAL (Write)
The axis labels of the NDF. This is not assigned if AXIS is FALSE.

AMONO( ) = _LOGICAL (Write)
These are TRUE when the axis centres are monotonic, and FALSE otherwise. This is not assigned if AXIS is FALSE.

ANORM( ) = _LOGICAL (Write)
The axis normalisation flags of the NDF. This is only written when FULLAXIS is TRUE and AXIS is TRUE.

ASTART( ) = _DOUBLE (Write)
The axis lower extents of the NDF. For non-monotonic axes, zero is used. See Parameter AMONO. This is not assigned if AXIS is FALSE.

ATYPE( ) = LITERAL (Write)
The data types of the axis centres of the NDF. This is only written when FULLAXIS is TRUE and AXIS is TRUE.

AUNITS( ) = LITERAL (Write)
The axis units of the NDF. This is not assigned if AXIS is FALSE.

AVARIANCE( ) = _LOGICAL (Write)
Whether or not there are axis variance arrays present in the NDF. This is only written when FULLAXIS is TRUE and AXIS is TRUE.

AXIS = _LOGICAL (Write)
Whether or not the NDF has an axis system.

BAD = _LOGICAL (Write)
If TRUE, the NDF's data array may contain bad values.

BADBITS = LITERAL (Write)
The BADBITS mask. This is only valid when QUALITY is TRUE.

CURRENT = _INTEGER (Write)
The integer Frame index of the current co-ordinate Frame in the WCS component.

DIMS( ) = _INT64 (Write)
The dimensions of the NDF.

EXTNAME( ) = LITERAL (Write)
The names of the extensions in the NDF. It is only written when NEXTN is positive.

EXTTYPE( ) = LITERAL (Write)
The types of the extensions in the NDF. Their order corresponds to the names in EXTNAME. It is only written when NEXTN is positive.

FDIM( ) = _INTEGER (Write)
The numbers of axes in each co-ordinate Frame stored in the WCS component of the NDF. The elements in this parameter correspond to those in the FDOMAIN and FTITLE parameters. The number of elements in each of these parameters is given by NFRAME.

FDOMAIN( ) = LITERAL (Write)
The domain of each co-ordinate Frame stored in the WCS component of the NDF. The elements in this parameter correspond to those in the FDIM and FTITLE parameters. The number of elements in each of these parameters is given by NFRAME.

FLABEL( ) = LITERAL (Write)
The axis labels from the current WCS Frame of the NDF.

FLBND( ) = _DOUBLE (Write)
The lower bounds of the bounding box enclosing the NDF in the current WCS Frame. The number of elements in this parameter is equal to the number of axes in the current WCS Frame (see FDIM). Celestial axis values will be in units of radians.

FORM = LITERAL (Write)
The storage form of the NDF's data array. This will be `"SIMPLE"`, `"PRIMITIVE"`, or `"SCALED"`.

FPIXSCALE( ) = LITERAL (Write)
The nominal WCS pixel scale for each axis in the current WCS Frame. For celestial axes, the value stored will be in arcseconds. For other axes, the value stored will be in the units given by the corresponding element of FUNIT.

FTITLE( ) = LITERAL (Write)
The title of each co-ordinate Frame stored in the WCS component of the NDF. The elements in this parameter correspond to those in the FDOMAIN and FDIM parameters. The number of elements in each of these parameters is given by NFRAME.

FUBND( ) = _DOUBLE (Write)
The upper bounds of the bounding box enclosing the NDF in the current WCS Frame. The number of elements in this parameter is equal to the number of axes in the current WCS Frame (see FDIM). Celestial axis values will be in units of radians.

FUNIT( ) = LITERAL (Write)
The axis units from the current WCS Frame of the NDF.

HISTORY = _LOGICAL (Write)
Whether or not the NDF contains HISTORY records.

LABEL = LITERAL (Write)
The label of the NDF.

LBOUND( ) = _INT64 (Write)
The lower bounds of the NDF.

NDIM = _INTEGER (Write)
The number of dimensions of the NDF.

NEXTN = _INTEGER (Write)
The number of extensions in the NDF.

NFRAME = _INTEGER (Write)
The number of WCS domains described by Parameters FDIM, FDOMAIN, and FTITLE. Set to zero if WCS is FALSE.

QUALITY = _LOGICAL (Write)
Whether or not the NDF contains a QUALITY array.

TITLE = LITERAL (Write)
The title of the NDF.

TYPE = LITERAL (Write)
The data type of the NDF's data array.

UBOUND( ) = _INT64 (Write)
The upper bounds of the NDF.

UNITS = LITERAL (Write)
The units of the NDF.

VARIANCE = _LOGICAL (Write)
Whether or not the NDF contains a variance array.

WCS = _LOGICAL (Write)
Whether or not the NDF has any WCS co-ordinate Frames, over and above the default GRID, PIXEL and AXIS Frames.

WIDTH( ) = _LOGICAL (Write)
Whether or not there are axis width arrays present in the NDF. This is only written when FULLAXIS is TRUE and AXIS is TRUE.

Examples:

```
ndftrace mydata
```

Displays information about the attributes of the NDF structure called mydata.

```
ndftrace ndf=r106 fullaxis
```

Displays information about the NDF structure r106, including full details of any axis arrays present.

```
ndftrace mydata ndim=(mdim)
```

Passes the number of dimensions of the NDF called mydata into the ICL variable mdim.

Notes:

- If the WCS component of the NDF is undefined, then an attempt is made to find WCS information from two other sources: first, an IRAS90 astrometry structure, and secondly, the FITS extension. If either of these sources yield usable WCS information, then it is displayed in the same way as the NDF WCS component. Other KAPPA applications will use this WCS information as if it were stored in the WCS component.
- The reporting of NDF attributes is suppressed when the message filter environment variable MSG_FILTER is set to `QUIET`. It benefits procedures and scripts where only the output parameters are needed. The creation of output parameters is unaffected by MSG_FILTER.

Related Applications :

KAPPA: FITSLIST, WCSFRAME; HDSTRACE.

Implementation Status:

Huge NDFs are supported.

NOGLOBALS
Resets the KAPPA global parameters

Description:

This application resets the KAPPA global parameters, and so makes their values undefined.

Usage:

```
noglobals
```

NOMAGIC
Replaces all occurrences of magic value pixels in an NDF array with a new value

Description:

This function replaces the standard 'magic value' assigned to bad pixels in an NDF with an alternative value, or with random samples taken from a Normal distribution. Input pixels which do not have the magic value are left unchanged. The number of replacements is reported. NOMAGIC's applications include the export of data to software that has different magic values or does not support bad values.

If a constant value is used to replace magic values (which will be the case if Parameter SIGMA is given the value zero), then the same replacement value is used for both the data and variance arrays when COMP="All". If the variance is being processed, the replacement value is constrained to be non-negative.

Magic values are replaced by random values if the Parameter SIGMA is given a non-zero value. If both DATA and VARIANCE components are being processed, then the random values are only stored in the DATA component; a constant value equal to SIGMA squared is used to replace all magic values in the VARIANCE component. If only a single component is being processed (whether it be DATA, VARIANCE, or Error), then the random values are used to replace the magic values. If random values are generated which will not fit into the allowed numeric range of the output NDF, then they are discarded and new random values are obtained instead. This continues until a usable value is obtained. This could introduce some statistical bias if many such re-tries are performed. For this reason SIGMA is restricted so that there are at least 4 standard deviations between the mean (given by REPVAL) and the nearest limit. NOMAGIC notifies of any re-tries that are required.

Usage:

```
nomagic in out repval [sigma] [comp]
```

Parameters:

COMP = LITERAL (Read)

The components whose flagged values are to be substituted. It may be:

- "Data"
- "Error"
- "Variance"
- "All"

The last of the options forces substitution of bad pixels in both the data and variance arrays. This parameter is ignored if the data array is the only array component within the NDF. ["Data"]

IN = NDF (Read)

Input NDF structure containing the data and/or variance array to have its elements flagged with the magic value replaced by another value.

OUT = NDF (Write)
Output NDF structure containing the data and/or variance array without any elements flagged with the magic value.

REPVAL = _DOUBLE (Read)
The constant value to substitute for the magic values, or (if Parameter SIGMA is given a non-zero value) the mean of the distribution from which replacement values are obtained. It must lie within the minimum and maximum values of the data type of the array with higher precision, except when variance is being processed, in which case the minimum is constrained to be non-negative. The replacement value is converted to the data type of the array being converted. The suggested default is the current value.

SIGMA = _DOUBLE (Read)
The standard deviation of the random values used to replace magic values in the input NDF. If this is zero (or if a null value is given), then a constant replacement value is used. The supplied value must be positive and must be small enough to allow at least 4 standard deviations between the mean value (given by REPVAL) and the closest limit. [!]

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (!) propagates the title from the input NDF to the output NDF. [!]

Examples:

```
nomagic aitoff irasmap repval=−2000000
```

This copies the NDF called aitoff to the NDF irasmap, except that any bad values in the data array are replaced with the IPAC blank value, −2000000, in the NDF called irasmap.

```
nomagic saturnb saturn 9999.0 comp=all
```

This copies the NDF called saturnb to the NDF saturn, except that any bad values in the data and variance arrays are replaced with 9999 in the NDF called saturn.

```
nomagic in=cleaned out=filled repval=0 sigma=10 comp=all
```

This copies the NDF called cleaned to the NDF filled, except that any bad values in the data array are replaced by random samples taken from a Normal distribution of mean zero and standard deviation 10. Bad values in the variance array are replaced by the constant value 100.

Notes:

- If the NDF arrays have no bad pixels the application will abort.
- Use GLITCH if a neighbourhood context is required to remove the bad values.

Related Applications :

KAPPA: CHPIX, FILLBAD, GLITCH, SEGMENT, SETMAGIC, SUBSTITUTE, ZAPLIN; FIGARO: GOODVAR.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

NORMALIZE
Normalises one NDF to a similar NDF by calculating a scale factor and zero-point difference

Description:

This application compares the data values in one NDF against the corresponding values in the other NDF. A least-squares straight-line is then fitted to the relationship between the two sets of data values in order to determine the relative scale factor and any zero-level offset between the NDFs (the offset may optionally be fixed at zero—see Parameter ZEROFF). To reduce computation time, the data points are binned according to the data value in the second NDF. The mean data value within each bin is used to find the fit and weights are applied according to the number of pixels which contribute to each bin.

To guard against erroneous data values, which can corrupt the fit obtained, the application then performs a number of iterations. It calculates a noise estimate for each bin according to the rms deviation of data values in the bin from the straight-line fit obtained previously. It then re-bins the data, omitting values which lie more than a specified number of standard deviations from the expected value in each bin. The straight-line fit is then re-calculated. You can specify the number of standard deviations and the number of iterations used.

A plot is produced after the final iteration showing the bin centres, with error bars representing the spread of values in each bin. The best fitting straight line is overlayed on this plot.

Optionally, an output NDF can be created containing a normalised version of the data array from the first input NDF.

For the special case of two-dimensional images, if IN2 (or IN1) spans only a single row or column, it can be used to normalize each row or column of IN1 (or IN2) in turn. See Parameter LOOP.

Usage:

```
normalize in1 in2 out
```

Parameters:

AXES = _LOGICAL (Read)

TRUE if labelled and annotated axes are to be drawn around the plot. The width of the margins left for the annotation may be controlled using Parameter MARGIN. The appearance of the axes (colours, founts, *etc.*) can be controlled using the Parameter STYLE. The dynamic default is TRUE if CLEAR is TRUE, and FALSE otherwise. []

CLEAR = _LOGICAL (Read)

If TRUE the current picture is cleared before the plot is drawn. If CLEAR is FALSE not only is the existing plot retained, but also an attempt is made to align the new picture with the existing picture. Thus you can generate a composite plot within a single set of axes, say using different colours or modes to distinguish data from different datasets. [TRUE]

DATARANGE(2) = _REAL (Read)
This parameter may be used to override the auto-scaling feature. If given, two real numbers should be supplied specifying the lower and upper data values in IN2, between which data will be used. If a null (`!`) value is supplied, the values used are the auto-scaled values, calculated according to the value of PCRANGE. Note, this parameter controls the range of data used in the fitting algorithm. The range of data displayed in the plot can be specified separately using Parameters XLEFT, XRIGHT, YBOT, and YTOP. `[!]`

DEVICE = DEVICE (Read)
The graphics workstation on which to produce the plot. If a null value (`!`) is supplied no plot will be made. [Current graphics device]

DRAWMARK = _LOGICAL (Read)
The central markers for each bin are not included in the plot if this parameter is set to `FALSE`. `[TRUE]`

DRAWWIDTH = _LOGICAL (Read)
The "error bars" marking the width of each bin are not included in the plot if this parameter is set to `FALSE`. `[TRUE]`

IN1 = NDF (Read)
The NDF to be normalised.

IN2 = NDF (Read)
The NDF to which IN1 will be normalised.

LOOP = _LOGICAL (Read)
If both IN1 and IN2 are two-dimensional, but one of them spans only a single row or column, then setting LOOP to `TRUE` will cause every row or column in to be normalised independently of each other. Specifically, if IN2 spans only a single row or column, then it will be used to normalise each row or column of IN1 in turn. Any output NDF (see Parameter OUT) will have the shape and size of IN1. If IN1 spans only a single row or column, then it will be normalised in turn by each row or column of IN2. Any output NDF (see Parameter OUT) will then have the shape and size of IN2. In either case, the details of the fit for each row or column will be displayed separately. Also see Parameters OUTSLOPE, OUTOFFSET, and OUTCORR. `[FALSE]`

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. If a null (`!`) value is supplied, the value used is `0.15` (for all edges) if annotated axes are produced, and zero otherwise. [current value]

MARKER = _INTEGER (Read)
Specifies the symbol with which each position should be marked in the plot. It should be given as an integer PGPLOT marker type. For instance, `0` gives a box, `1` gives a dot, `2` gives a cross, `3` gives an asterisk, `7` gives a triangle. The value must be larger than or equal to -31. [current value]

MINPIX = _INTEGER (Read)
The minimum number of good pixels required in a bin before it contributes to the fitted line. It must be in the range 1 to the number of pixels per bin. `[2]`

NBIN = _INTEGER (Read)
The number of bins to use when binning the scatter plot prior to fitting a straight line, in the range 2 to 10000. [50]

NITER = _INTEGER (Read)
The number of iterations performed to reject bad data values in the range 0 to 100. [2]

NSIGMA = _REAL (Read)
The number of standard deviations at which bad data are rejected. It must lie in the range 0.1 to 1.0E6. [3.0]

OUT = NDF (Write)
An optional output NDF to hold a version of IN1 which is normalised to IN2. A null (!) value indicates that an output NDF is not required. See also Parameter LOOP.

OUTCORR = NDF (Write)
An optional 1-dimensonal output NDF to hold the correlation coefficient for each row or column when LOOP=YES. See Parameter CORR. Ignored if LOOP=NO. [!]

OUTOFFSET = NDF (Write)
An optional 1-dimensonal output NDF to hold the offset used for each row or column when LOOP=YES. See Parameter OFFSET. Ignored if LOOP=NO. [!]

OUTSLOPE = NDF (Write)
An optional 1-dimensonal output NDF to hold the slope used for each row or column when LOOP=YES. See Parameter SLOPE. Ignored if LOOP=NO. [!]

PCRANGE(2) = _REAL (Read)
This parameter takes two real values in the range 0 to 100 and is used to modify the action of the auto-scaling algorithm which selects the data to use in the fitting algorithm. The two values correspond to the percentage points in the histogram of IN2 at which the lower and upper cuts on data value are placed. With the default value, the plots will omit those pixels that lie in the lower and upper two-percent intensity range of IN2. Note, this parameter controls the range of data used in the fitting algorithm. The range of data displayed in the plot can be specified separately using Parameters XLEFT, XRIGHT, YBOT, and YTOP. [2,98]

STYLE = GROUP (Read)
A group of attribute settings describing the plotting style to use when drawing the annotated axes, data values, error bars, and best-fitting line.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with

a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the best-fitting straight line is controlled by the attributes Colour(Curves), Width(Curves), *etc.* (the synonym Line may be used in place of Curves). The appearance of markers is controlled by Colour(Markers), Width(Markers), *etc.* (the synonym Symbols may be used in place of Markers). The appearance of the error bars is controlled using Colour(ErrBars), Width(ErrBars), *etc*. Note, Size(ErrBars) controls the length of the serifs (*i.e.* the cross pieces at each end of the error bar), and defaults to 1.0. [current value]

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN1 to be used instead. [!]

XLEFT = _DOUBLE (Read)
The axis value to place at the left hand end of the horizontal axis of the plot. If a null (!) value is supplied, the value used is the minimum data value used by the fitting algorithm from IN2 (with a small margin). 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 of the plot. If a null (!) value is supplied, the value used is the maximum data value used by the fitting algorithm from IN2 (with a small margin). 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 of the plot. If a null (!) value is supplied, the value used is the minimum data value used by the fitting algorithm from IN1 (with a small margin). 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 of the plot. If a null (!) value is supplied, the value used is the maximum data value used by the fitting algorithm from IN1 (with a small margin). The value supplied may be greater than or less than the value supplied for YBOT. [!]

ZEROFF = _LOGICAL (Read)
If TRUE, the offset of the linear fit is constrained to be zero. [FALSE]

Results Parameters:

CORR = _REAL (Write)
Pearson's coefficient of linear correlation for the data included in the last fit.

OFFSET = _REAL (Write)
The offset in the linear normalisation expression: IN1 = SLOPE $*$ IN2 + OFFSET.

SLOPE = _REAL (Write)
The slope of the linear normalisation expression: IN1 = SLOPE $*$ IN2 + OFFSET.

Examples:

```
normalize cl123a cl123b cl123c
```

This normalises NDF cl123a to the NDF cl123b. A plot of the fit is made on the current graphics device, and the resulting normalisation scale and offset are written only to the `normalize.sdf` parameter file (as in the all the examples below except where noted). The NDF cl123c is the normalised version of the input cl123a.

```
normalize cl123a cl123b style="'size(errba)=0,title=Gain calibration'"
```

This normalises NDF cl123a to the NDF cl123b. A plot of the fit is made on the current graphics device with the title `"Gain calibration"`. The error bars are drawn with no serifs.

```
normalize cl123a cl123b cl123c offset=(shift) slope=(scale)
```

This normalises NDF cl123a to the NDF cl123b. A plot of the fit is made on the current graphics device. The resulting normalisation scale and offset are written to the ICL variables SCALE and SHIFT respectively, where they could be passed to another application via an ICL procedure. The NDF cl123c is the normalised version of the input cl123a.

```
normalize in2=old in1=new out=!  device=xwindows style=^normstyle
```

This normalises NDF new to the NDF old. A plot of the fit is made on the xwindows device, using the plotting style defined in text file normstyle. No output NDF is produced.

```
normalize in1=new in2=old out=young niter=5 pcrange=[3,98.5]
```

This normalises NDF new to the NDF old. It has five iterations to reject outliers from the linear regression, and forms the regression using pixels in old whose data values lie between the 3 and 98.5 percentiles, comparing with the corresponding pixels in new. A plot of the fit is made on the current graphics device. The NDF young is the normalised version of the input new.

Notes:

- The application stores two pictures in the graphics database in the following order: a FRAME picture containing the annotated axes and data plot, and a DATA picture containing just the data plot. Note, the FRAME picture is only created if annotated axes have been drawn, or if non-zero margins were specified using Parameter MARGIN. The world co-ordinates in the DATA picture will correspond to data value in the two NDFs.

Related Applications :

CCDPACK: MAKEMOS.

Implementation Status:

- The routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF, and propagates all extensions to the output NDF. All propagated components come from the NDF to be normalised.
- At the moment, variance values are not used in the fitting algorithm but are modified in the output NDF to take account of the scaling introduced by the normalisation. (A later version may take account of variances in the fitting algorithm.)
- Processing of bad pixels and automatic quality masking are supported.
- Only _REAL data can be processed directly. Other non-complex numeric data types will undergo a type conversion before processing occurs. _DOUBLE data cannot be processed due to a loss of precision.
- The pixel bounds of the two input NDFs are matched by trimming before calculating the normalisation constants, and are mapped as vectors to allow processing of NDFs of any dimensionality. An output NDF may optionally be produced which is based on the first input NDF (IN1) by applying the calculated normalisation constants to IN1.

NUMB
Counts the number of elements of an NDF with values or absolute values above or below a threshold

Description:

This routine counts and reports the number of elements of an array within an input NDF structure that have a value or absolute value greater or less than a specified threshold. This statistic is also shown as a percentage of the total number of array elements.

Usage:

```
numb in value [comp]
```

Parameters:

ABS = _LOGICAL (Read)

If ABS=TRUE, the criterion is a comparison of the absolute value with the threshold; if FALSE, the criterion is a comparison of the actual value with the threshold. The current value is the suggested default. [FALSE]

ABOVE = _LOGICAL (Read)

If ABOVE=TRUE the criterion tests whether values are greater than the threshold; if FALSE the criterion tests whether values are less than the threshold. The current value of ABOVE is the suggested default. [TRUE]

COMP = LITERAL (Read)

The components whose flagged values are to be substituted. It may be "Data", "Error", "Variance", or "Quality". If "Quality" is specified, then the quality values are treated as numerical values in the range 0 to 255. ["Data"]

IN = NDF (Read)

Input NDF structure containing the array to be tested.

VALUE = _DOUBLE (Read)

Threshold against which the values of the array elements will be tested. It must lie in within the minimum and maximum values of the data type of the array being processed, unless ABS=TRUE or the component is the variance or quality array, in which case the minimum is zero. The suggested default is the current value.

Results Parameters:

NUMBER = _INTEGER (Write)

The number of elements that satisfied the criterion.

Examples:

```
numb image 100
```

This counts the number of elements in the data array of the NDF called image that exceed 100.

```
numb spectrum 100 noabove
```

This counts the number of elements in the data array of the NDF called spectrum that are less than 100.

```
numb cube 100 abs
```

This counts the number of elements in the data array of the NDF called cube whose absolute values exceed 100.

```
numb image −100 number=(count)
```

This counts the number of elements in the data array of the NDF called image that exceed −100 and write the number to ICL variable COUNT.

```
numb image 200 v
```

This counts the number of elements in the variance array of the NDF called image that exceed 200.

Implementation Status:

- This routine correctly processes the DATA, QUALITY, TITLE, and VARIANCE components of an NDF data structure.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.
- Huge NDFs are supported.

ODDEVEN
Removes odd-even defects from a one-dimensional NDF

Description:

This application forms a smoothed signal for a one-dimensional NDF whose elements have oscillating biases. It averages the signal levels of alternating pixels. Both elements must be good and not deviate from each other by more than a threshold for the averaging to take place.

This application is intended for images exhibiting alternating patterns in columns or rows, the so called odd-even noise, arising from electronic interference or readout through different channels. However, you must supply a vector collapsed along the unaffected axis, such that the vector exhibits the pattern. See task COLLAPSE using the Mode or Median estimators. The smoothed image is then subtracted from the supplied vector to yield the odd-even pattern.

Usage:

```
oddeven in out [thresh]
```

Parameters:

IN = NDF (Read)
The one-dimensional NDF containing the input array to be filtered.

OUT = NDF (Write)
The NDF to contain the filtered image.

THRESH = _DOUBLE (Read)
The maximum difference between adjacent elements for the averaging filter to be applied. This allows anomalous pixels to be excluded. If null, `!`, is given, then there is no limit. `[!]`

TITLE = LITERAL (Read)
The title of the output NDF. A null (`!`) value means using the title of the input NDF. `[!]`

Examples:

```
oddeven raw clean
```

The one-dimensional NDF called raw is filtered such that adjacent pixels are averaged to form the output NDF call clean.

```
oddeven out=clean in=raw thresh=20
```

As above except only those adjacent pixels whose values differ by no more than 20 are averaged.

Related Applications :

KAPPA: BLOCK, CHPIX, FFCLEAN, GLITCH, ZAPLIN; FIGARO: BCLEAN, CLEAN, ISEDIT, TIPPEX.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- VARIANCE is merely propagated.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single- or double-precision floating point as appropriate.

OUTLINE
Draws an outline of a two-dimensional NDF

Description:

This application draws an outline of a two-dimensional NDF on the current graphics device, aligning it with any existing plot.

Annotated axes can be produced (see Parameter AXES), and the appearance of the axes and curve can be controlled in detail (see Parameter STYLE). The axes show co-ordinates in the current co-ordinate Frame of the supplied NDF.

This command is a synonym for `contour mode=bounds penrot=yes clear=no`.

Usage:

```
outline ndf
```

Parameters:

AXES = _LOGICAL (Read)

`TRUE` if labelled and annotated axes are to be drawn around the plot, showing the current co-ordinate Frame of the supplied NDF. The appearance of the axes can be controlled using the STYLE parameter. `[TRUE]`

DEVICE = DEVICE (Read)

The plotting device. [current graphics device]

LABOUT = _LOGICAL (Read)

Specifies the position at which to place a label identifying the input NDF within the plot. The label is drawn parallel to the first pixel axis. Two values should be supplied for LABPOS. The first value specifies the distance in millimetres along the first pixel axis from the centre of the bottom-left pixel to the left edge of the label. The second value specifies the distance in millimetres along the second pixel axis from the centre of the bottom-left pixel to the baseline of the label. If a null (`!`) value is given, no label is produced. The appearance of the label can be set by using the STYLE parameter (for instance `"Size(strings)=2"`). [current value]

MARGIN(4) = _REAL (Read)

The widths of the margins to leave around the outline for axis annotation. The widths should be given as fractions of the corresponding dimension of the current picture. The actual margins used may be increased to preserve the aspect ratio of the DATA 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. If a null (`!`) value is supplied, the value used is `0.15` (for all edges) if annotated axes are being produced, and zero otherwise. [current value]

NDF = NDF (Read)

NDF structure containing the two-dimensional image to be outlined.

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the outline and annotated axes.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the outline is controlled by the attributes Colour(Curves), Width(Curves), *etc*. [current value]

Related Applications :

KAPPA: WCSFRAME, CONTOUR, PICDEF; CCDPACK: DRAWNDF.

OUTSET
Mask pixels inside or outside a specified circle in a two-dimensional NDF

Description:

This routine assigns a specified value (which may be *bad*) to either the outside or inside of a specified circle within a specified component of a given two-dimensional NDF .

Usage:

```
outset in out centre diam
```

Parameters:

CENTRE = LITERAL (Read)

The co-ordinates of the centre of the circle. The position must be given in the current co-ordinate Frame of the NDF (supplying a colon `":"` will display details of the current co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas. See also Parameter USEAXIS. The current co-ordinate Frame can be changed using task WCSFRAME.

COMP = LITERAL (Read)

The NDF array component to be masked. It may be `"Data"`, or `"Variance"`, or `"Error"` (where `"Error"` is equivalent to `"Variance"`). `["Data"]`

CONST = LITERAL (Given)

The constant numerical value to assign to the masked pixels, or the string `"bad"`. `["bad"]`

DIAM = LITERAL (Read)

The diameter of the circle. If the current co-ordinate Frame of the NDF is a SKY Frame (*e.g.* RA and DEC), then the value should be supplied as an increment of celestial latitude (*e.g.* DEC). Thus, `"10.2"` means 10.2 arcseconds, `"30:0"` would mean 30 arcminutes, and `"1:0:0"` would mean 1 degree. If the current co-ordinate Frame is not a SKY Frame, then the diameter should be specified as an increment along Axis 1 of the current co-ordinate Frame. Thus, if the current Frame is PIXEL, the value should be given simply as a number of pixels.

IN = NDF (Read)

The name of the source NDF.

INSIDE = _LOGICAL (Read)

If a `TRUE` value is supplied, the constant value is assigned to the inside of the circle. Otherwise, it is assigned to the outside. `[FALSE]`

OUT = NDF (Write)

The name of the masked NDF.

TITLE = LITERAL (Read)

Title for the output NDF structure. A null value (`!`) propagates the title from the input NDF to the output NDF. `[!]`

USEAXIS = GROUP (Read)

USEAXIS is only accessed if the current co-ordinate Frame of the NDF has more than two axes. A group of strings should be supplied designating the axes that are to be used when specifying the circle via Parameters CENTRE and DIAM. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as `"RA"` or `"VRAD"`.
- A generic option where `"SPEC"` requests the spectral axis, `"TIME"` selects the time axis, `"SKYLON"` and `"SKYLAT"` picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (`!`) value is supplied, the axes with the same indices as the two used pixel axes within the NDF are used. `[!]`

Examples:

```
outset neb1 nebm "13.5,201.3" 20 const=0
```

This copies NDF neb1 to nebm, setting pixels to zero in the DATA array if they fall outside the specified circle. Assuming the current co-ordinate Frame of neb1 is PIXEL, the circle is centred at pixel co-ordinates (13.5, 201.3) and has a diameter of 20 pixels.

```
outset neb1 nebm "15:23:43.2 -22:23:34.2" "10:0" inside comp=var
```

This copies NDF neb1 to nebm, setting pixels bad in the variance array if they fall inside the specified circle. Assuming the current co-ordinate Frame of neb1 is a SKY Frame describing RA and DEC, the aperture is centred at RA 15:23:43.2 and DEC -22:23:34.2, and has a diameter of 10 arcminutes.

Related Applications :

KAPPA: ARDMASK, REGIONMASK.

Implementation Status:

- This routine correctly processes the WCS, AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- Bad pixels and quality masking are supported.
- All non-complex numeric data types can be handled.

PALDEF
Loads the default palette to a colour table

Description:

This application loads the standard palette of colours to fill the portion of the current graphics device's colour table which is reserved for the palette. The palette comprises 16 colours and is intended to provide coloured annotations, borders, axes, graphs *etc.* that are unaffected by changes to the lookup table used for images.

The standard palette is as follows 0: Black 1: White 2: Red 3: Green 4: Blue 5: Yellow 6: Magenta 7: Cyan 8–15: Black

Usage:

```
paldef [device]
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device to be used. [Current graphics device]

Examples:

```
paldef
```

This loads the standard palette into the reserved portion of the colour table of the current graphics device.

```
paldef xwindows
```

This loads the standard palette into the reserved portion of the colour table of the xwindows device.

Notes:

- The effects of this command will only be immediately apparent when run on X windows which have 256 colours (or other similar pseudocolour devices). On other devices (for instance, X windows with more than 256 colours) the effects will only become apparent when subsequent graphics applications are run.

Related Applications :

KAPPA: PALENTRY, PALREAD, PALSAVE.

PALENTRY
Enters a colour into an graphics device's palette

Description:

This application obtains a colour and enters it into the palette portion of the current graphics device's colour table. The palette comprises up to 16 colours and is intended to provide coloured annotations, borders, axes, graphs *etc.* that are unaffected by changes to the lookup table used for images.

A colour is specified either by the giving the red, green, blue intensities; or named colours.

Usage:

```
palentry palnum colour [device]
```

Parameters:

COLOUR() = LITERAL (Read)

A colour to be added to the palette at the entry given by Parameter PALNUM. It is one of the following options.

- A named colour from the standard colour set, which may be abbreviated. If the abbreviated name is ambiguous the first match (in alphabetical order) is selected. The case of the name is ignored. Some examples are `"Pink"`, `"Yellow"`, `"Aquamarine"`, and `"Orchid"`.
- Normalised red, green, and blue intensities separated by commas or spaces. Each value must lie in the range 0.0–1.0. For example, `"1.0,1.0,0.5"` would give a pale yellow.
- An HTML colour code such as `#ff002d`.

DEVICE = DEVICE (Read)

Name of the graphics device to be used. [Current graphics device]

PALNUM = _INTEGER (Read)

The number of the palette entry whose colour is to be modified. PALNUM must lie in the range zero to the minimum of 15 or the number of colour indices minus one. The suggested default is 1.

Examples:

```
palentry 5 gold
```

This makes palette entry number 5 have the colour gold in the reserved portion of the colour table of the current image display.

```
palentry 12 [1.0,1.0,0.3] xwindows
```

This makes the xwindows device's palette entry number 12 have a pale-yellow colour.

Notes:

- The effects of this command will only be immediately apparent when run on X windows which have 256 colours (or other similar pseudocolour devices). On other devices (for instance, X windows with more than 256 colours) the effects will only become apparent when subsequent graphics applications are run.

Related Applications :

KAPPA: PALDEF, PALREAD, PALSAVE.

PALREAD
Fills the palette of a colour table from an NDF

Description:

This application reads a palette of colours from an NDF , stored as red, green and blue intensities, to fill the portion of the current graphics device's colour table which is reserved for the palette. The palette comprises 16 colours and is intended to provide coloured annotations, borders, axes, graphs *etc.* that are unaffected by changes to the lookup table used for images.

Usage:

```
palread palette [device]
```

Parameters:

DEVICE = DEVICE (Read)

Name of the graphics device to be used. [Current graphics device]

PALETTE = NDF (Read)

The name of the NDF containing the palette of reserved colours as its data array. The palette must be two-dimensional, the first dimension being 3, and the second 16. If the second dimension is greater than 16 only the first 16 colours are used; if it has less than 16 just fill as much of the palette as is possible starting from the first colour. The palette's values must lie in the range 0.0–1.0.

Examples:

```
palread rustic
```

This loads the palette stored in the NDF called rustic into the reserved portion of the colour table of the current graphics device.

```
palread rustic xwindows
```

This loads the palette stored in the NDF called rustic into the reserved portion of the colour table of the xwindows device.

Notes:

- The effects of this command will only be immediately apparent when run on X windows which have 256 colours (or other similar pseudocolour devices). On other devices (for instance, X windows with more than 256 colours) the effects will only become apparent when subsequent graphics applications are run.

Related Applications :

KAPPA: PALDEF, PALENTRY, PALSAVE.

PALSAVE
Saves the current palette of a colour table to an NDF

Description:

This application reads the palette portion of the current graphics device's colour table and saves it in an NDF . The palette comprises 16 colours and is intended to provide coloured annotations, borders, axes, graphs *etc.* that are unaffected by changes to the lookup table used for images. Thus once you have established a palette of colours you prefer, it is straightforward to recover the palette at a future time.

Usage:

```
palsave palette [device] [title]
```

Parameters:

DEVICE = DEVICE (Read)
Name of the graphics device to be used. [Current graphics device]

PALETTE = NDF (Write)
The NDF in which the current colour-table reserved pens are to be stored. Thus if you have created non-standard colours for annotation, doodling, colour of axes *etc.* they may be stored for future use.

TITLE = LITERAL (Read)
Title for the output NDF. `["KAPPA - Palsave"]`

Examples:

```
palsave rustic
```

This saves the palette of the colour table of the current graphics device into the NDF called rustic.

```
palsave hitec xwindows title="Hi-tech-look palette"
```

This saves the palette of the colour table of the xwindows device in the NDF called hitec. The NDF has a title called `"Hi-tech-look palette"`.

Related Applications :

KAPPA: PALDEF, PALENTRY, PALREAD.

PARGET
Obtains the value or values of an application parameter

Description:

This application reports the value or values of a parameter from a named task. It does this by searching the parameter file of the task. The purpose is to offer an easier-to-use interface for passing values (especially results parameters) between tasks in shell scripts. The values are formatted in lines with as many values as can be accommodated across the screen up to a maximum of 132 characters; values are space separated. However, in scripts the values are likely to be written to a script variable. Thus for example in the C-shell:

```
set med = ‘parget median histat‘
```

would redirect the output to shell variable med, and thus a reference to $med would substitute the median value obtained the last time application HISTAT was invoked. If the parameter comprises a vector of values these can be stored in a C-shell array. For instance,

```
set perval = ‘parget perval histat‘
```

would assign elements of the shell array perval[1], perval[2], *etc.* to the last-computed percentile values of HISTAT. For other scripting languages such as Python, the alternative vector format produced by setting Parameter VECTOR to TRUE may be more appropriate.

Single elements of an parameter array may also be accessed using the array index in parentheses.

Usage:

```
parget parname applic
```

Parameters:

APPLIC = LITERAL (Read)
The name of the application from which the parameter comes.

PARNAME = LITERAL (Read)
The parameter whose value or values are to be reported.

VECTOR = _LOGICAL (Read)
If TRUE, then vector parameters will be displayed as a comma-separated list of values enclosed in square brackets. If FALSE, vector values are printed as a space-separated list with no enclosing brackets. Additionally, if VECTOR is TRUE, string values (whether vector or scalar) are enclosed in single quotes and any embedded quotes are escaped using a backslash. [FALSE]

Examples:

```
parget mean stats
```

Report the value of Parameter MEAN for the application STATS.

```
parget mincoord \
```

This reports the values of Parameter MINCOORD of the current application, in this case STATS.

```
parget applic=ndftrace parname=flabel(2)
```

This reports the value of the second element of Parameter FLABEL for the application NDFTRACE.

Notes:

- The parameter file is located in the `$ADAM_USER` directory, if defined, otherwise in the `adam` subdirectory of `$HOME`. If it cannot be located there, the task reports an error.
- The parameter must exist in the selected application parameter file and not be a structure, except one of type ADAM_PARNAME.
- This task is not designed for use with ICL, where passing parameter values is quite straightforward. It does not operate with monolith parameter files.

PASTE
Pastes a series of NDFs upon each other

Description:

This application copies a series of NDFs , in the order supplied and taking account of origin information, on to a 'base' NDF to produce an output NDF. The output NDF is therefore a copy of the base NDF obscured wholly or partially by the other input NDFs. This operation is analogous to pasting in publishing. It is intended for image editing and the creation of insets.

The dimensions of the NDFs may be different, and indeed so may their dimensionalities. The output NDF can be constrained to have the dimensions of the base NDF, so the pasted NDFs are clipped. Normally, the output NDF will have dimensions such that all the input NDFs are accommodated in full.

Bad values in the pasted NDFs are by default transparent, so the underlying data are not replaced during the copying.

Input NDFs can be shifted in pixel space before pasting them into the output NDF (see Parameter SHIFT).

Usage:

```
paste in p1 [p2] ...  [p25] out=?
```

Parameters:

CONFINE = _LOGICAL (Read)

This parameter controls the dimensions of the output NDF. If CONFINE is `FALSE` the output NDF just accommodates all the input NDFs. If CONFINE is `TRUE`, the output NDF's dimensions matches those of the base NDF. `[FALSE]`

IN = NDF (Read)

This parameter is either:
a) the base NDF on to which the other input NDFs supplied via Parameters P1 to P25 will be pasted; or
b) a group of input NDFs (of any dimensionality) comprising all the input NDFs, of which the first is deemed to be the base NDF, and the remainder are to be pasted in the order supplied.

The group should be given as a comma-separated list, in which each list element can be:

- an NDF name, optionally containing wild-cards and/or regular expressions (`"*"`, `"?"`, `"[a-z]"` *etc.*).
- the name of a text file, preceded by an up-arrow character `"^"`. Each line in the text file should contain a comma-separated list of elements, each of which can in turn be an NDF name (with optional wild-cards, *etc.*), or another file specification (preceded by an up-arrow). Comments can be included in the file by commencing lines with a hash character `"#"`.

If the value supplied for this parameter ends with a hyphen "-", then you are re-prompted for further input until a value is given which does not end with a hyphen. All the NDFs given in this way are concatenated into a single group.

The group can contain no more than 1000 names.

OUT = NDF (Write)

The NDF resulting from pasting of the input NDFs on to the base NDF. Its dimensions may be different from the base NDF. See Parameter CONFINE.

P1-P25 = NDF (Read)

The NDFs to be pasted on to the base NDF. The NDFs are pasted in the order P1, P2, ... P25. There can be no missing NDFs, *e.g.* in order for P3 to be processed there must be a P2 given as well. A null value (!) indicates that there is no NDF. NDFs P2 to P25 are defaulted to !. At least one NDF must be pasted, therefore P1 may not be null.

P1 to P25 are ignored if the group specified through Parameter IN comprises more than one NDF.

SHIFT(*) = _INTEGER (Read)

An incremental shift to apply to the pixel origin of each input NDF before pasting it into the output NDF. If supplied, this parameter allows a set of NDFs with the same pixel bounds to be placed 'side-by-side' in the output NDF. For instance, this allows a set of images to be pasted into a cube. The first input NDF is not shifted. The pixel origin of the second NDF is shifted by the number of pixels given in SHIFT. The pixel origin of the third NDF is shifted by twice the number of pixels given in SHIFT. Each subsequent input NDF is shifted by a further multiple of SHIFT. If null (!) is supplied, no shifts are applied. [!]

TITLE = LITERAL (Read)

Title for the output NDF structure. A null value (!) propagates the title from the base NDF to the output NDF. [!]

TRANSP = _LOGICAL (Read)

If TRANSP is TRUE, bad values within the pasted NDFs are not copied to the output NDF as if the bad values were transparent. If TRANSP is FALSE, all values are copied during the paste and a bad value will obscure an underlying value. [TRUE]

Examples:

```
paste aa inset out=bb
```

This pastes the NDF called inset on to the arrays in the NDF called aa to produce the NDF bb. Bad values are transparent. The bounds and dimensionality of bb may be larger than those of aa.

```
paste aa inset out=bb notransp
```

As above except that bad values are copied from the NDF inset to NDF bb.

```
paste aa inset out=bb confine
```

As the first example except that the bounds of NDF bb match those of NDF aa.

```
paste in="aa,inset" out=bb
```

The same as the first example.

```
paste in="aa,inset,inset2,inset3" out=bb
```

Similar to first example, but now two further NDFs inset2 and inset3 are also pasted.

```
paste ccd fudge inset out=ccdc
```

This pastes the NDF called fudge, followed by NDF inset on to the arrays in the NDF called ccd to produce the NDF ccdc. Bad values are transparent. The bounds and dimensionality of ccd may be larger than those of ccdc.

```
paste in="canvas,^shapes.lis" out=collage confine
```

This pastes the NDFs listed in the text file `shapes.lis` in the order given on the NDF called canvas. Bad values are transparent. The bounds of NDF collage match those of NDF canvas.

```
paste in=^planes out=cube shift=[0,0,1]
```

Assuming the text file `planes` contains a list of two-dimensional NDFs, this arranges them into a cube, one behind the other.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY, components of an NDF data structure and propagates all extensions. Propagation is from the base NDF.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

PERMAXES
Permute an NDF's pixel axes

Description:

This application re-orders the pixel axes of an NDF , together with all related information (AXIS structures, and the axes of all co-ordinate Frames stored in the WCS component of the NDF).

Usage:

```
permaxes in out perm
```

Parameters:

IN = NDF (Read)
The input NDF data structure.

OUT = NDF (Write)
The output NDF data structure.

PERM() = _INTEGER (Read)
A list of integers defining how the pixel axes are to be permuted. The list must contain one element for each pixel axis in the NDF. The first element is the index of the pixel axis within the input NDF which is to become Axis 1 in the output NDF. The second element is the index of the pixel axis within the input NDF which is to become Axis 2 in the output NDF, *etc*. Axes are numbered from 1.

TITLE = LITERAL (Read)
A title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
permaxes a b [2,1]
```

Swaps the axes in the two-dimensional NDF called a, to produce a new two-dimensional NDF called b.

```
permaxes a b [3,1,2]
```

Creates a new three-dimensional NDF called b in which Axis 1 corresponds to Axis 3 in the input three-dimensional NDF called a, Axis 2 corresponds to input Axis 1, Axis 3 corresponds to input Axis 2.

Notes:

- If any WCS co-ordinate Frame has more axes then the number of pixel axes in the NDF, then the high numbered surplus axes in the WCS Frame are left unchanged.

- If any WCS co-ordinate Frame has fewer axes then the number of pixel axes in the NDF, then the Frame is left unchanged if the specified permutation would change any of the high numbered surplus pixel axes. A warning message is issued if this occurs.

Related Applications :

KAPPA: ROTATE, FLIP; FIGARO: IREVX, IREVY, IROT90.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. The data type of the input pixels is preserved in the output NDF.
- Huge NDFs are supported.

PICBASE
Selects the BASE picture from the graphics database.

Description:

This command selects the BASE picture. Subsequent plotting for the chosen device will be in this new current picture. The BASE picture is the largest picture available, and encompasses all other pictures. By default the chosen device is the current one.

This command is a synonym for `piclist picnum=1`.

Usage:

```
picbase
```

Parameters:

DEVICE = DEVICE (Read)
The graphics workstation. [The current graphics device]

Examples:

```
picbase
```

This selects the BASE picture for the current graphics device.

```
picbase device=x2w
```

This selects the BASE picture for the x2w device.

Related Applications :

KAPPA: PICCUR, PICDATA, PICFRAME, PICLAST, PICLIST, PICSEL.

PICCUR
Uses a graphics cursor to change the current picture

Description:

This application allows you to select a new current picture in the graphics database using the cursor. Each time a position is selected (usually by pressing a button on the mouse), details of the topmost picture in the AGI database which encompasses that position are displayed, together with the cursor position (in millimetres from the bottom -left corner of the graphics device). On exit the last picture selected becomes the current picture.

Usage:

```
piccur [device] [name]
```

Parameters:

DEVICE = DEVICE (Read)
The graphics workstation. [The current graphics device]

NAME = LITERAL (Read)
Only pictures of this name are to be selected. For instance, if you want to select a DATA picture which is covered by a transparent FRAME picture, then you could specify NAME="DATA". A null (!) or blank string means that pictures of all names may be selected. [!]

SINGLE = _LOGICAL (Read)
If TRUE then the user can supply only one position using the cursor, where-upon the application immediately exits, leaving the selected picture as the current picture. If FALSE is supplied, then the user can supply multiple positions. Once all positions have been supplied, a button is pressed to indicate that no more positions are required. [FALSE]

Examples:

```
piccur
```

This selects a picture on the current graphics device by use of the cursor. The picture containing the last-selected point becomes the current picture.

```
piccur name=data
```

This is like the previous example, but only DATA pictures can be selected.

Notes:

- Nothing is displayed on the screen when the message filter environment variable MSG_FILTER is set to QUIET.

Related Applications :

KAPPA: CURSOR, PICBASE, PICDATA, PICEMPTY, PICENTIRE, PICFRAME, PICLIST, PICSEL, PICVIS.

PICDATA
Selects the last DATA picture from the graphics database.

Description:

This command selects the last-created DATA picture. Subsequent plotting for the chosen device will be in this new current picture. By default the chosen device is the current one.

This command is a synonym for `piclist name=data picnum=last`.

Usage:

```
picdata
```

Parameters:

DEVICE = DEVICE (Read)
The graphics workstation. [The current graphics device]

Examples:

```
picdata
```

This selects the last DATA picture for the current graphics device.

```
picdata device=xw
```

This selects the last DATA picture for the xw device.

Related Applications :

KAPPA: PICCUR, PICBASE, PICFRAME, PICLAST, PICLIST, PICSEL.

PICDEF
Defines a new graphics-database FRAME picture or an array of FRAME pictures

Description:

This application creates either one new graphics-database FRAME picture or a grid of new FRAME pictures. It offers a variety of ways by which you can define a new picture's location and extent. You may constrain a new picture to lie within either the current or the BASE picture, and the new picture adopts the world co-ordinate system of that reference picture.

You may specify a single new picture using one of three methods: 1. moving a cursor to define the lower and upper bounds via pressing choice number 1 (the application will instruct what to do for the specific graphics device), provided a cursor is available on the chosen graphics workstation; 2. obtaining the bounds from the environment (in world co-ordinates of the reference picture); 3. or by giving a position and size for the new picture. The position is specified by a two-character code. The first controls the vertical location, and may be `"T"`, `"B"`, or `"C"` to create the new picture 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 picture to the left, right, or in the centre respectively. Thus a code of `"BR"` will make a new picture in the bottom-right corner. The size of the new picture along each axis may be specified either in centimetres, or as a fraction of the corresponding axis of the reference picture. The picture may also be forced to have a specified aspect ratio.

The picture created becomes the current picture on exit.

Alternatively, you can create an array of *n*-by-*m* equal-sized pictures by giving the number of pictures in the horizontal and vertical directions. These may or may not be abutted. For easy reference in later processing the pictures may be labelled automatically. The label consists of a prefix you define, followed by the number of the picture. The numbering starts at a defined value, usually one, and increments by one for each new picture starting from the bottom-left corner and moving from left to right to the end of the line. This is repeated in each line until the top-right picture. Thus if the prefix were `"GALAXY"`, the start number is one and the array comprises three pictures horizontally and two vertically, the top-left picture would have the label `"GALAXY4"`. On completion the bottom-left picture in the array becomes the current picture.

Usage:

```
picdef [mode] [fraction] { xpic ypic prefix=?
                         { lbound=?  ubound=?
                       mode
```

Parameters:

ASPECT = _REAL (Read)

The aspect ratio (x/y) of the picture to be created in modes other than Cursor, Array, and XY. The new picture is the largest possible with the chosen aspect ratio that

will fit within the part of the reference picture defined by the fractional sizes (see Parameter FRACTION). The justification comes from the value of MODE. Thus to obtain the largest picture, set FRACTION=1.0. A null value (!) does not apply an aspect-ratio constraint, and therefore the new picture fills the part of the reference picture defined by the fractional sizes. [!]

CURRENT = _LOGICAL (Read)

TRUE if the new picture is to lie within the current picture, otherwise the new picture can lie anywhere within the BASE picture. In other words, when it is TRUE the current picture is the reference picture, and when FALSE, the base is the reference picture. [FALSE]

DEVICE = DEVICE (Read)

The graphics device. [Current graphics device]

FILL = _REAL (Read)

The linear filling factor for the Array mode. In other words the fractional size (applied to both co-ordinates) of the new picture within each of the XPIC $*$ YPIC abutted sections of the picture being sub-divided. Each new picture is located centrally within the section. A filling factor of 1.0 means that the pictures in the array abut. Smaller factors permit a gap between the pictures. For example, FILL = 0.9 would give a gap between the created pictures of 10 per cent of the height and width of each picture, with exterior borders of 5 per cent. FILL must lie between 0.1 and 1.0. [1.0]

FRACTION( ) = _REAL (Read)

The fractional size of the new picture along each axis, applicable for modes other than Array, Cursor, and XY. Thus FRACTION controls the relative shape as well as the size of the new picture. If only a single value is given then it is applied to both x and y axes, whereupon the new picture has the shape of the reference picture. So a value of 0.5 would create a picture 0.25 the area of the current or BASE picture. The default is 0.5, unless Parameter ASPECT is not null, when the default is 1.0. This parameter is not used if the picture size is specified in centimetres using Parameter SIZE. []

LABELNO = _INTEGER (Read)

The number used to form the label for the first (bottom-left) picture in Array mode. It cannot be negative. [1]

LBOUND(2) = _REAL (Read)

Co-ordinates of the lower bound that defines the new picture. The suggested default is the bottom-left of the current picture. (XY mode)

MODE = LITERAL (Read)

Method for selecting the new picture. The options are "Cursor" for cursor mode (provided the graphics device has one), "XY" to select x-y limits, and "Array" to create a grid of equal-sized FRAME pictures. The remainder are locations specified by two characters, the first corresponding to the vertical position and the second the horizontal. For the vertical, valid positions are T(op), B(ottom), or C(entre); and for the horizontal the options are L(eft), R(ight), or C(entre). ["Cursor"]

OUTLINE = _LOGICAL (Read)

If TRUE, a box that delimits the new picture is drawn. [TRUE]

PREFIX = LITERAL (Read)

The prefix to be given to the labels of picture created in Array mode. It should contain no more than twelve characters. If the empty string "" is given, the pictures will have

enumerated labels. Note that the database can contain only one picture with a given label, and so merely numbering labels increases the chance of losing existing labels. A ! response means no labelling is required. The suggested default is the last-used prefix.

SIZE(2) = _REAL (Read)
The size of the new picture along both axes, in centimetres, applicable for modes other than Array, Cursor, and XY. If a single value is given, it is used for both axes. If a null value (!) is given, then the size of the picture is determined by Parameter FRACTION. [!]

UBOUND(2) = _REAL (Read)
Co-ordinates of the upper bound that defines the new picture. The suggested default is the top-right of the current picture. (XY mode)

XPIC = _INTEGER (Read)
The number of new pictures to be formed horizontally in the BASE or current picture in Array mode. The total number of new pictures is XPIC $*$ YPIC. The value must lie in the range 1–20. The suggested default is 2.

YPIC = _INTEGER (Read)
The number of new pictures to be formed vertically in the BASE or current picture in Array mode. The value must lie in the range 1–20. The suggested default is the value of Parameter XPIC.

Examples:

```
picdef tr
```

Creates a new FRAME picture in the top-right quarter of the full display area on the current graphics device, and an outline is drawn around the new picture. This picture becomes the current picture.

```
picdef bl aspect=1.0
```

Creates a new FRAME picture within the full display area on the current graphics device, and an outline is drawn around the new picture. This picture is the largest square possible, and it is justified to the bottom-left corner. It becomes the current picture.

```
picdef bl size=[15,10]
```

Creates a new FRAME picture within the full display area on the current graphics device, and an outline is drawn around the new picture. This picture is 15 by 10 centimetres in size and it is justified to the bottom-left corner. It becomes the current picture.

```
picdef cc 0.7 current nooutline
```

Creates a new FRAME picture situated in the centre of the current picture on the current graphics device. The new picture has the same shape as the current one, but it is linearly reduced by a factor of 0.7. No outline is drawn around it. The new picture becomes the current picture.

```
picdef cc [0.8,0.5] current nooutline
```

As above except that its height is half that of the current picture, and its width is 0.8 of the current picture.

```
picdef cu device=graphon
```

Creates a new FRAME picture within the full display area of the Graphon device. The bounds of the new picture are defined by cursor interaction. An outline is drawn around the new picture which becomes the current picture.

```
picdef mode=a prefix=M xpic=3 ypic=2
```

Creates six new equally sized and abutting FRAME pictures within the full display area of the current graphics device. All are outlined. They have labels M1, M2, M3, M4, M5, and M6. The bottom-left picture (M1) becomes the current picture.

```
picdef mode=a prefix="" xpic=3 ypic=2 fill=0.8
```

As above except that the pictures do not abut since each is 0.8 times smaller in both dimensions, and the labels are 1, 2, 3, 4, 5, and 6.

Related Applications :

KAPPA: PICBASE, PICCUR, PICDATA, PICFRAME, PICGRID, PICLABEL, PICLIST, PICSEL, PICXY.

PICEMPTY
Finds the first empty FRAME picture in the graphics database

Description:

This application selects the first, *i.e.* oldest, empty FRAME picture in the graphics database for a graphics device. Empty means that there is no additional picture lying completely with its bounds. This implies that the FRAME is clear for plotting. This task is probably most useful for plotting data in a grid of FRAME pictures.

Usage:

```
picempty [device]
```

Parameters:

DEVICE = DEVICE (Read)

The graphics workstation. [The current graphics device]

Examples:

```
picempty
```

This selects the first empty FRAME picture for the current graphics device.

```
picempty xwindows
```

This selects the first empty FRAME picture for the xwindows graphics device.

Notes:

- An error is returned if there is no empty FRAME picture, and the current picture remains unchanged.

Related Applications :

KAPPA: PICENTIRE, PICGRID, PICLAST, PICLIST, PICSEL, PICVIS.

Timing :

The execution time is approximately proportional to the number of pictures in the database before the first empty FRAME picture is identified.

PICENTIRE
Finds the first unobscured and unobscuring FRAME picture in the graphics database

Description:

This application selects the first, *i.e.* oldest, FRAME picture in the graphics database for a graphics device, subject to the following criterion. The picture must not obstruct any other picture except the BASE, and must itself not be obstructed. Unobstructed means that there is no younger picture overlying it either wholly or in part. This means that plotting can occur within the selected FRAME picture without overwriting or obscuring earlier plots.

Usage:

```
picentire [device]
```

Parameters:

DEVICE = DEVICE (Read)

The graphics workstation. [The current graphics device]

Examples:

```
picentire
```

This selects the first unobscured and unobscuring FRAME picture for the current graphics device.

```
picentire xwindows
```

This selects the first unobscured and unobscuring FRAME picture for the xwindows graphics device.

Notes:

- An error is returned if there is no suitable FRAME picture, and the current picture remains unchanged.
- This routine cannot know whether or a picture has been cleared, and hence is safe to reuse, as such information is not stored in the graphics database.

Related Applications :

KAPPA: PICEMPTY, PICGRID, PICLAST, PICLIST, PICSEL, PICVIS.

Timing :

The execution time is approximately proportional to a linear combination of the number of pictures in the database before the unobstructed FRAME picture is found, and the square of the number of pictures in the database.

PICFRAME
Selects the last FRAME picture from the graphics database.

Description:

This command selects the last-created FRAME picture. Subsequent plotting for the chosen device will be in this new current picture. By default the chosen device is the current one.

This command is a synonym for `piclist name=frame picnum=last`.

Usage:

```
picframe
```

Parameters:

DEVICE = DEVICE (Read)
The graphics workstation. [The current graphics device]

Examples:

```
picframe
```

This selects the last FRAME picture for the current graphics device.

```
picframe device=xw
```

This selects the last FRAME picture for the xw device.

Related Applications :

KAPPA: PICBASE, PICCUR, PICDATA, PICLAST, PICLIST, PICSEL.

PICGRID
Creates an array of FRAME pictures

Description:

This command creates a two-dimensional grid of equally sized new FRAME pictures in the graphics database. The array of pictures do not have to abut, but abutting is the default. The new pictures are formed within either the current or BASE picture, and they adopt the world co-ordinate system of that enclosing picture. On completion, the bottom-left picture in the array becomes the current picture.

For easy reference in later processing the pictures have integer labels. The numbering starts at a defined value, usually one, and increments by one for each new picture starting from the bottom-left corner and moving from left to right to the end of the line. This is repeated in each line until the top-right picture.

This command is a synonym for `picdef array 1.0 prefix=""`.

Usage:

```
picgrid xpic ypic
```

Parameters:

CURRENT = _LOGICAL (Read)
`TRUE` if the new pictures are to lie within the current picture, otherwise the new pictures can lie anywhere within the BASE picture. In other words, when CURRENT is `TRUE` the current picture is the reference picture, and when it is `FALSE` the BASE is the reference picture. [`FALSE`]

DEVICE = DEVICE (Read)
The graphics device. [Current graphics device]

FILL = _REAL (Read)
The linear filling factor for the array. In other words the fractional size (applied to both co-ordinates) of the new picture within each of the XPIC $*$ YPIC abutted sections of the picture being sub-divided. Each new picture is located centrally within the section. A filling factor of 1.0 means that the pictures in the array abut. Smaller factors permit a gap between the pictures. For example, FILL=`0.9` would give a gap between the created pictures of 10 per cent of the height and width of each picture, with exterior borders of 5 per cent. FILL must lie between 0.1 and 1.0. [`1.0`]

LABELNO = _INTEGER (Read)
The number used to form the label for the first (bottom-left) picture. It cannot be negative. [`1`]

OUTLINE = _LOGICAL (Read)
If `TRUE`, a box that delimits the new picture is drawn. [`TRUE`]

XPIC = _INTEGER (Read)
The number of new pictures to be formed horizontally in the BASE picture. The total number of new pictures is XPIC $*$ YPIC. The value must lie in the range 1–20. The suggested default is 2.

YPIC = _INTEGER (Read)
The number of new pictures to be formed vertically in the BASE picture. The total number of new pictures is XPIC $*$ YPIC. The value must lie in the range 1–20. The suggested default is the value of Parameter XPIC.

Examples:

```
picgrid 3 2
```

Creates six new equally sized and abutting FRAME pictures within the full display area of the current graphics device. All the pictures are outlined. They have labels 1, 2, 3, 4, 5, and 6. The bottom-left picture (1) becomes the current picture.

```
picgrid xpic=3 ypic=2 fill=0.8 labelno=11 nooutline
```

As above except that the pictures do not abut since each is 0.8 times smaller in both dimensions, the labels are 11 to 16, and there are no picture outlines drawn.

```
picgrid device=xw current
```

This creates a 2-by-2 grid of new FRAME pictures within the current picture on device xw.

Related Applications :

KAPPA: PICCUR, PICDEF, PICLABEL, PICSEL, PICXY.

PICIN
Finds the attributes of a picture interior to the current picture

Description:

This application finds the attributes of a picture, selected by name, that was created since the current picture and lies within the bounds of the current picture. The search starts from the most-recent picture, unless the current picture is included, whereupon the current picture is tested first.

The attributes reported are the name, comment, label, name of the reference data object, the bounds in the co-ordinate Frame selected by Parameter FRAME.

Usage:

```
picin [name] [device] [frame]
```

Parameters:

CURRENT = _LOGICAL (Read)

If this is TRUE, the current picture is compared against the chosen name before searching from the most-recent picture within the current picture. [FALSE]

DESCRIBE = _LOGICAL (Read)

This controls whether or not the report (when REPORT=TRUE) should contain a description of the Frame being used. [FALSE]

DEVICE = DEVICE (Read)

Name of the graphics device about which information is required. [Current graphics device]

EPOCH = _DOUBLE (Read)

If a 'Sky Co-ordinate System' specification is supplied (using Parameter FRAME) for a celestial co-ordinate system, then an epoch value is needed to qualify it. This is the epoch at which the displayed sky co-ordinates 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.

FRAME = LITERAL (Read)

A string determining the co-ordinate Frame in which the bounds of the picture are to be reported. When a picture is created by an application such as PICDEF, DISPLAY, the WCS information describing the available co-ordinate systems are stored with the picture in the graphics database. This application can report bounds in any of the co-ordinate Frames stored with the current picture. The string supplied for FRAME can be one of the following:

- A domain name such as SKY, AXIS, PIXEL, NDC, BASEPIC, CURPIC. the special domain AGI_WORLD is used to refer to the world co-ordinate system stored in the AGI graphics database. This can be useful if no WCS information was store with the picture when it was created.

- An integer value giving the index of the required Frame.
- An IRAS90 *Sky Co-ordinate System* (SCS) values such as `"EQUAT(J2000)"` (see SUN/163).

If a null value (`!`) is supplied, bounds are reported in the co-ordinate Frame which was current when the picture was created. `[!]`

NAME = LITERAL (Read)
The name of the picture to be found within the current picture. If it is null (`!`), the first interior picture is selected. `[DATA]`

PNAME = LITERAL (Write)
The name of the picture.

REPORT = _LOGICAL (Read)
If this is `FALSE` details of the picture are not reported, merely the results are written to the output parameters. It is intended for use within procedures. `[TRUE]`

Results Parameters:

COMMENT = LITERAL (Write)
The comment of the picture. Up to 132 characters will be written.

DOMAIN = LITERAL (Write)
The Domain name of the current co-ordinate Frame for the picture.

LABEL = LITERAL (Write)
The label of the picture. It is blank if there is no label.

REFNAM = LITERAL (Write)
The reference object associated with the picture. It is blank if there is no reference object. Up to 132 characters will be written.

X1 = LITERAL (Write)
The lowest value found within the picture for Axis 1 of the requested co-ordinate Frame (see Parameter FRAME).

X2 = LITERAL (Write)
The highest value found within the picture for Axis 1 of the requested co-ordinate Frame (see Parameter FRAME).

Y1 = LITERAL (Write)
The lowest value found within the picture for Axis 2 of the requested co-ordinate Frame (see Parameter FRAME).

Y2 = LITERAL (Write)
The highest value found within the picture for Axis 2 of the requested co-ordinate Frame (see Parameter FRAME).

Examples:

```
picin
```

This reports the attributes of the last DATA picture within the current picture for the current graphics device. The bounds of the picture in its current co-ordinate Frame are reported.

```
picin frame=pixel
```

As above but the bounds of the picture in the PIXEL Frame are reported.

```
picin refnam=(object) current
```

This reports the attributes of the last data picture within the current picture for the current graphics device. If there is a reference data object, its name is written to the ICL variable OBJECT. The search includes the current picture.

```
picin x1=(x1) x2=(x2) y1=(y1) y2=(y2)
```

This reports the attributes of the last DATA picture within the current picture for the current graphics device. The bounds of the current picture are written to the ICL variables: X1, X2, Y1, Y2.

Notes:

This application is intended for use within procedures. Also if a DATA picture is selected and the current picture is included in the search, this application informs about the same picture that an application that works in a cursor interaction mode would select, and so acts as a check that the correct picture will be accessed.

Related Applications :

KAPPA: GDSTATE, PICDEF, PICLIST, PICTRANS, PICXY.

PICLABEL
Labels the current graphics-database picture

Description:

This application annotates the current graphics-database picture of a specified device with a label you define. This provides an easy-to-remember handle for selecting pictures in subsequent processing.

Usage:

```
piclabel label [device]
```

Parameters:

DEVICE = DEVICE (Read)
The graphics device. [Current graphics device]

LABEL = LITERAL (Read)
The label to be given to the current picture. It is limited to 15 characters, but may be in mixed case. If it is null (`!`) a blank label is inserted in the database.

Examples:

```
piclabel GALAXY
```

This makes the current picture of the current graphics device have a label of `"GALAXY"`.

```
piclabel A3 x2w
```

This labels the current picture on the x2w device `"A3"`.

Notes:

The label must be unique for the chosen device. If the new label clashes with an existing one, then the existing label is deleted.

Related Applications :

KAPPA: PICDEF, PICLIST, PICSEL.

PICLAST
Selects the last picture from the graphics database.

Description:

This command selects the last-created picture. Subsequent plotting for the chosen device will be in this new current picture. By default the chosen device is the current one.

This command is a synonym for `piclist picnum=last`.

Usage:

```
piclast
```

Parameters:

DEVICE = DEVICE (Read)
The graphics workstation. [current graphics device]

Examples:

```
piclast
```

This selects the last picture for the current graphics device.

```
piclast device=x2w
```

This selects the last picture for the x2w device.

Related Applications :

KAPPA: PICBASE, PICCUR, PICDATA, PICFRAME, PICLIST, PICSEL.

PICLIST
Lists the pictures in the graphics database for a device

Description:

This application produces a summary of the contents of the graphics database for a graphics device, and/or permits a picture specified by its order in the list to be made the new current picture. The list may either be reported or written to a text file.

The headed list has one line per picture. Each line comprises a reference number; the picture's name, comment (up to 24 characters), and label; and a flag to indicate whether or not there is a reference data object associated with the picture. A `"C"` in the first column indicates that the picture that was current when this application was invoked. In the text file, because there is more room, the name of the reference object is given (up to 64 characters) instead of the reference flag. Pictures are listed in chronological order of their creation.

Usage:

```
piclist [name] [logfile] [device] picnum=?
```

Parameters:

DEVICE = DEVICE (Read)
The graphics workstation. [The current graphics device]

LOGFILE = FILENAME (Write)
The name of the text file in which the list of pictures will be made. A null string (`!`) means the list will be reported to you. The suggested default is the current value. `[!]`

NAME = LITERAL (Read)
Only pictures of this name are to be selected. A null string (`!`) or blanks means that pictures of all names may be selected. `[!]`

PICNUM = LITERAL (Read)
The reference number of the picture to be made the current picture when the application exits. PICNUM=`"Last"` selects the last picture in the database. Parameter PICNUM is not accessed if the list is written to the text file. A null (`!`) or any other error causes the current picture on entry to be current again on exit. The suggested default is null.

Examples:

```
piclist
```

This reports all the pictures in the graphics database for the current graphics device.

```
piclist device=ps_l
```

This reports all the pictures in the graphics database for the ps_l device.

```
piclist data
```

This reports all the DATA pictures in the graphics database for the current graphics device.

```
piclist data logfile=datapic.dat
```

This lists all the DATA pictures in the graphics database for the current graphics device into the text file `datapic.dat`.

```
piclist frame picnum=5
```

This selects the fifth most ancient FRAME picture (in the graphics database for the current graphics device) as the current picture. The pictures are not listed.

```
piclist picnum=last
```

This makes the last picture in the graphics database for the current graphics device the current picture. The pictures are not listed.

Related Applications :

KAPPA: PICBASE, PICDATA, PICEMPTY, PICENTIRE, PICFRAME, PICIN, PICLAST, PICSEL, PICVIS.

Timing :

The execution time is approximately proportional to the number of pictures in the database for the chosen graphics device. Selecting only a subset by name is slightly faster.

PICSEL
Selects a graphics-database picture by its label

Description:

This application selects by label a graphics-database picture of a specified device. If such a picture is found then it becomes the current picture on exit, otherwise the input picture remains current. Labels in the database are stored in the case supplied when they were created. However, the comparisons of the label you supply with the labels in the database are made in uppercase, and leading spaces are ignored.

Should the label not be found the current picture is unchanged.

Usage:

```
picsel label [device]
```

Parameters:

DEVICE = DEVICE (Read)
The graphics device. [Current graphics device]

LABEL = LITERAL (Read)
The label of the picture to be selected.

Examples:

```
picsel GALAXY
```

This makes the picture labelled `"GALAXY"` the current picture on the current graphics device. Should there be no picture of this name, the current picture is unchanged.

```
picsel A3 xwindows
```

This makes the picture labelled `"A3"` the current picture on the xwindows device. Should there be no picture of this name, the current picture is unchanged.

Related Applications :

KAPPA: PICDATA, PICDEF, PICEMPTY, PICENTIRE, PICFRAME, PICLABEL, PICLAST, PICVIS.

PICTRANS
Transforms a graphics position from one picture co-ordinate Frame to another

Description:

This application transforms a position on a graphics device from one co-ordinate Frame to another. The input and output Frames may be chosen freely from the Frames available in the WCS information stored with the current picture in the AGI graphics database. The transformed position is formatted for display and written to the screen and also to an output parameter.

Usage:

```
pictrans posin framein [frameout] [device]
```

Parameters:

DEVICE = DEVICE (Read)

The graphics workstation. [The current graphics device]

EPOCHIN = _DOUBLE (Read)

If a 'Sky Co-ordinate System' specification is supplied (using Parameter FRAMEIN) for a celestial co-ordinate system, then an epoch value is needed to qualify it. This is the epoch at which the supplied sky position was 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.

EPOCHOUT = _DOUBLE (Read)

If a 'Sky Co-ordinate System' specification is supplied (using Parameter FRAMEOUT) for a celestial co-ordinate system, then an epoch value is needed to qualify it. This is the epoch at which the transformed sky position is required. 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.

FRAMEIN = LITERAL (Read)

A string specifying the co-ordinate Frame in which the input position is supplied (see Parameter POSIN). The string can be one of the following options.

- A domain name such as SKY, AXIS, PIXEL, NDC, BASEPIC, CURPIC.
- 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).

If a null parameter value is supplied, then the current Frame in the current picture is used. [!]

FRAMEOUT = LITERAL (Read)
A string specifying the co-ordinate Frame in which the transformed position is required. If a null parameter value is supplied, then the current Frame in the picture is used. The string can be one of the following options.

- A domain name such as SKY, AXIS, PIXEL, GRAPHICS, CURPIC, NDC, BASEPIC.
- 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).

If a null parameter value is supplied, then the BASEPIC Frame is used. `["BASEPIC"]`

POSIN = LITERAL (Read)
The co-ordinates of the position to be transformed, in the co-ordinate Frame specified by Parameter FRAMEIN (supplying a colon `":"` will display details of the required co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas.

Results Parameters:

BOUND = _LOGICAL (Write)
BOUND is `TRUE` when the supplied point lies within the bounds of the current picture.

POSOUT = LITERAL (Write)
The formatted co-ordinates of the transformed position, in the co-ordinate Frame specified by Parameter FRAMEOUT. The position will be stored as a list of formatted axis values separated by spaces.

Examples:

```
pictrans "100.3,-20.1" framein=pixel
```

This converts the position (100.3, −20.1), in pixel co-ordinates within the current picture of the current graphics device, to the BASEPIC co-ordinates of that point in the BASE picture.

```
pictrans "100.3,-20.1" framein=pixel frameout=graphics
```

This converts the position (100.3, −20.1), in pixel co-ordinates within the current picture of the current graphics device, to the GRAPHICS co-ordinates of that point (*i.e.* millimetres from the bottom-left corner of the graphics device).

```
pictrans "10 10" framein=graphics frameout=basepic
```

This converts the position (10, 10), in graphics co-ordinates (*i.e.* the point which is 10mm above and to the right of the lower-left corner of the graphics device), into BASEPIC co-ordinates.

Notes:

- BASEPIC co-ordinates locate a position within the entire graphics device. The bottom-left corner of the device screen has BASEPIC co-ordinates of (0, 0). The shorter dimension of the screen has length 1.0, and the other axis has a length greater than 1.0.
- NDC co-ordinates are like BASEPIC co-ordinates except that the top-right corner of the screen has NDC co-ordinates of (1, 1). That is, both axes of the screen have unit length.
- GRAPHICS co-ordinates also span the entire graphics device but are measured in millimetres from the bottom-left corner.
- CURPIC co-ordinates locate a point within the current picture. The bottom-left corner of the current picture has CURPIC co-ordinates of (0, 0). The shorter dimension of the current picture has length 1.0, and the other axis has a length greater than 1.0.
- The transformed position is not written to the screen when the message filter environment variable MSG_FILTER is set to `QUIET`. The creation of the output Parameter POSOUT is unaffected by MSG_FILTER.

Related Applications :

KAPPA: GDSTATE, PICIN, PICXY.

PICVIS
Finds the first unobscured FRAME picture in the graphics database

Description:

This application selects the first, *i.e.* oldest, unobstructed FRAME picture in the graphics database for a graphics device. Unobstructed means that there is no younger picture overlying it either wholly or in part.

Usage:

```
picvis [device]
```

Parameters:

DEVICE = DEVICE (Read)
The graphics workstation. [The current graphics device]

Examples:

```
picvis
```

This selects the first unobscured FRAME picture for the current graphics device.

```
picvis xwindows
```

This selects the first unobscured FRAME picture for the xwindows graphics device.

Notes:

- An error is returned if there is no unobscured FRAME picture, and the current picture remains unchanged.
- This routine cannot know whether or a picture has been cleared, and hence is safe to reuse, as such information is not stored in the graphics database.

Related Applications :

KAPPA: PICEMPTY, PICENTIRE, PICGRID, PICLAST, PICLIST, PICSEL.

Timing :

The execution time is approximately proportional to a linear combination of the number of pictures in the database before the unobstructed FRAME picture is found, and the square of the number of pictures in the database.

PICXY
Creates a new FRAME picture defined by co-ordinate bounds

Description:

This command creates a new FRAME picture in the graphics database. The bounds of the new picture are defined through two parameters. The new picture is formed within either the current or BASE picture, and it adopts the world co-ordinate system of that reference picture. On completion the new picture becomes the current picture.

This command is a synonym for `picdef xy 1.0`.

Usage:

```
picxy lbound ubound
```

Parameters:

CURRENT = _LOGICAL (Read)
`TRUE` if the new picture is to lie within the current picture, otherwise the new picture can lie anywhere within the BASE picture. In other words, when CURRENT is `TRUE` the current picture is the reference picture, and when it is `FALSE` the base is the reference picture. `[FALSE]`

DEVICE = DEVICE (Read)
The graphics device. [current graphics device]

LBOUND(2) = _REAL (Read)
Co-ordinates of the lower bound that defines the new picture. The suggested default is the bottom-left of the current picture.

OUTLINE = _LOGICAL (Read)
If `TRUE`, a box that delimits the new picture is drawn. `[TRUE]`

UBOUND(2) = _REAL (Read)
Co-ordinates of the upper bound that defines the new picture. The suggested default is the top-right of the current picture.

Examples:

```
picxy [0.1,0.2] [0.9,0.6]
```

This creates a new FRAME picture in the BASE picture extending from (0.1, 0.2) to (0.9, 0.6), which becomes the new current picture. An outline is drawn around the picture.

```
picxy ubound=[1.1,0.9] lbound=[0.1,0.2] current nooutline
```

This creates a new FRAME picture in the current picture extending from (0.1, 0.2) to (1.1, 0.9), which becomes the new current picture. No outline is drawn.

Related Applications :

KAPPA: PICCUR, PICDEF, PICGRID, PICSEL.

PIXBIN
Places each pixel value in an input NDF into an output bin

Description:

This application collects groups of pixel values together from an input NDF and places each group into a single column of an output NDF. Each such output column represents a "bin" into which a group of input pixels is placed.

If the input NDF has *N* pixel axes, the user provides a set of *M* *N*-dimensional "index" NDFs (where *M* is between 1 and 6). For each pixel in the input NDF, the corresponding value in each of the *M* index NDFs is found. This vector of *M* values is used (after rounding them to the nearest integer) to determine the pixel indices within the output (*M*-dimensional) NDF at which to store the input pixel value. Thus each output pixel corresponds to a bin into which one or more input pixels can be placed, as selected by the index NDFs.

There are many possible ways in which the input pixels values that fall in a single bin could be combined to create a single representative output value for each bin. For instance, the output NDF could contain the mean of the input values that fall in each bin, or the maximum, or the standard deviation, etc. However, this application does not store a single representative value for each bin. Instead it stores all the separate input pixel values that fall in each bin. This requires an extra trailing pixel axis in the output NDF, with a lower pixel bounds of 1 and and upper pixel bound equal to the maximum population of any bin. Each "column" of values parallel to this final output pixel axis represents one bin, and contains the corresponding input pixel values at its lower end, with bad values filling any unused higher pixels. The COLLAPSE application could then be used to get a representative value for each bin by collapsing this final pixel axis using any of the many estimators provided by COLLAPSE.

An extra group of *M* NDFs can be supplied that define the WCS to be stored in the output NDF—see Parameter WCS.

Usage:

```
pixbin in out index [wcs]
```

Parameters:

IN = NDF (Read)
The input *N*-dimensional NDF.

INDEX = NDF (Read)
A group of index NDFs (all with *N*-dimensions). The number of index NDFs (referred to below as "*M*") supplied should be in the range 1–6 and determines the dimensionality of the output NDF. A section is taken from each one so that it matches the input NDF supplied by Parameter IN. The data values in the *J*th index NDF are converted to _INTEGER (by finding the nearest integer) and then used as the pixel indices on the *J*th output pixel axis.

OUT = NDF (Write)
The output NDF containing all the values from the input NDF collected into a set of

bins. This NDF will have $M + 1$ pixel axes, where M is the number of index NDF supplied using Parameter INDEX. The final pixel axis enumerates the individual input pixels that fall within each bin.

WCS = NDF (Read)

An optional group of NDFs (all with N-dimensions) that define the WCS to be stored in the output NDF. The number of NDFs in this group should be M, the number of index NDFs (see Parameter INDEX). The data values in the Jth WCS NDF determine the values to be stored for the Jth axis in the WCS of the output NDF (the WCS values on the final trailing axis in the output NDF, axis $M + 1$, are just equal to pixel index). If a null (`!`) value is supplied, no WCS is stored in the output NDF. The WCS values for each of the first M output axes are described using a look-up-table (one for each axis) that converts value in an index NDF into the corresponding value in a WCS NDF. For all pixels with the same integer index value, the mean of the corresponding WCS values is found and stored in the look-up-table. The label and unit for each axis is taken from the Label and Unit components of the corresponding WCS NDF. `[!]`

Examples:

```
pixbin m31 binned radius
```

Here the pixel values in a two-dimensional NDF called m31 are placed into bins as defined by the contents of a single two-dimensional NDF called radius, to create a two-dimensional output NDF called binned. (The number of pixel axes in the output is always one more than the number of index NDFs.) The data values in NDF radius are used as the pixel indices along the first axis of the output NDF, at which to store each input pixel value. Each column in the output NDF contains the individual input pixel values assigned to that bin, padded with bad values if necessary to fill the column.

Related Applications :

KAPPA: COLLAPSE

Implementation Status:

- This routine correctly processes the DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, and HISTORY, components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

PIXDUPE
Expands an NDF by pixel duplication

Description:

This routine expands the size of an NDF structure by duplicating each input pixel a specified number of times along each dimension, to create a new NDF structure. Each block of output pixels (formed by duplicating a single input pixel) can optionally be masked, for instance to set selected pixels within each output block bad.

Usage:

```
pixdupe in out expand
```

Parameters:

EXPAND() = _INTEGER (Read)

Linear expansion factors to be used to create the new data array. The number of factors should equal the number of dimensions in the input NDF. If fewer are supplied the last value in the list of expansion factors is given to the remaining dimensions. Thus if a uniform expansion is required in all dimensions, just one value need be entered. If the net expansion is one, an error results. The suggested default is the current value.

IMASK() = _INTEGER (Read)

Only used if a null (`!`) value is supplied for Parameter MASK. If accessed, the number of values supplied for this parameter should equal the number of pixel axes in the output NDF. A mask array is then created which has bad values at every element except for the element with indices given by IMASK, which is set to the value `1.0`. See Parameter MASK for a description of the use of the mask array. If a null value is supplied for IMASK, then no mask is used, and every output pixel in an output block is set to the value of the corresponding input pixel. `[!]`

IN = NDF (Read)

Input NDF structure to be expanded.

MASK = NDF (Read)

An input NDF structure holding the mask to be used. If a null (`!`) value is supplied, Parameter IMASK will be used to determine the mask. If supplied, the NDF Data array will be trimmed or padded (with bad values) to create an array in which the lengths of the pixel axes are equal to the values supplied for Parameter EXPAND. Each block of pixels in the output array (*i.e.* the block of output pixels which are created from a single input pixel) are multiplied by this mask array before being stored in the output NDF. `[!]`

OUT = NDF (Write)

Output NDF structure.

TITLE = LITERAL (Read)

Title for the output NDF structure. A null value (`!`) propagates the title from the input NDF to the output NDF. `[!]`

Examples:

```
pixdupe aa bb 2
```

This expands the NDF called aa duplicating pixels along each dimension, and stores the enlarged data in the NDF called bb. Thus if aa is two-dimensional, this command would result in a four-fold increase in the array components.

```
pixdupe cosmos galaxy [2,1]
```

This expands the NDF called cosmos by duplicating along the first axis, and stores the enlarged data in the NDF called galaxy.

```
pixdupe cube1 cube2 [3,1,2] title="Reconfigured cube"
```

This expands the NDF called cube1 by having three pixels for each pixel along the first axis and duplicating along the third axis, and stores the enlarged data in the NDF called cube2. The title of cube2 is `"Reconfigured cube"`.

Related Applications :

KAPPA: COMPADD, COMPAVE, COMPICK, INTERLEAVE.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY, components of an NDF data structure, and propagates all extensions.
- The AXIS centre, width and variance values in the output are formed by duplicating the corresponding input AXIS values.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

PLUCK
Plucks slices from an NDF at arbitrary positions

Description:

This application's function is to extract data at scientifically relevant points such as the spatial location of a source or wavelength of a spectral feature, rather than at data sampling points (for which NDFCOPY is appropriate). This is achieved by the extraction of interpolated slices from an NDF . A slice is located at a supplied set of co-ordinates in the current WCS Frame for some but not all axes, and it possesses one fewer significant dimension per supplied co-ordinate. The slices run parallel to pixel axes of the NDF.

The interpolation uses one of a selection of resampling methods to effect the non-integer shifts along the fixed axes, applied to each output element along the retained axes (see the METHOD, PARAMS, and TOL parameters).

Three routes are available for obtaining the fixed positions, selected using Parameter MODE:

- from the parameter system (see Parameter POS);
- from a specified positions list (see Parameter INCAT); or
- from a simple text file containing a list of co-ordinates (see Parameter COIN).

In the first mode the application loops, asking for new extraction co-ordinates until it is told to quit or encounters an error. However there is no looping if the position has been supplied on the command line.

Each extracted dataset is written to a new NDF, which however, may reside in a single container file (see the CONTAINER parameter).

Usage:

```
                                    { pos
pluck in axes out method [mode]     { coin=?
                                    { incat=?
                                  mode
```

Parameters:

AXES( ) = _INTEGER (Read)

The WCS axis or axes to remain in the output NDF. The slice will therefore contain an array comprising all the elements along these axes. The maximum number of axes is one fewer than the number of WCS axes in the NDF.

Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).

- Its Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If the axes of the current Frame are not parallel to the NDF pixel axes, then the pixel axis which is most nearly parallel to the specified current Frame axis will be used.

COIN = FILENAME (Read)
Name of a text file containing the co-ordinates of slices to be plucked. It is only accessed if Parameter MODE is given the value "File". Each line should contain the formatted axis values for a single position, in the current Frame of the NDF. Axis values can be separated by spaces, tabs or commas. The file may contain comment lines with the first character # or !.

CONTAINER = _LOGICAL (Read)
If TRUE, each slice extracted is written as an NDF component of the HDS container file specified by the OUT parameter. The *n*th component will be named PLUCK_*n*. If set FALSE, each extraction is written to a separate file. On-the-fly format conversion to foreign formats is not possible when CONTAINER=TRUE. [FALSE]

DESCRIBE = _LOGICAL (Read)
If TRUE, a detailed description of the co-ordinate Frame in which the fixed co-ordinates are to be supplied is displayed before the positions themselves. It is ignored if MODE="Catalogue". [current value]

INCAT = FILENAME (Read)
A catalogue containing a positions list giving the co-ordinates of the fixed positions, such as produced by applications CURSOR, LISTMAKE, *etc.* It is only accessed if Parameter MODE is given the value "Catalogue". The catalogue should have a WCS Frame common with the NDF, so that the NDF and catalogue FrameSets can be aligned.

MODE = LITERAL (Read)
The mode in which the initial co-ordinates are to be obtained. The supplied string can be one of the following values.

- "Interface" — positions are obtained using Parameter POS.
- "Catalogue" — positions are obtained from a positions list using Parameter INCAT.
- "File" — positions are obtained from a text file using Parameter COIN.

[current value]

IN = NDF (Read)
The NDF structure containing the data to be extracted. It must have at least two dimensions.

METHOD = LITERAL (Read)
The method to use when sampling the input pixel values. For details of these schemes, see the descriptions of routine AST_RESAMPLEx in SUN/210. METHOD can take the following values.

- `"Linear"` — When resampling, the output pixel values are calculated by linear interpolation in the input NDF among the two nearest pixel values along each axis chosen by AXES. This method produces smoother output NDFs than the nearest-neighbour scheme, but is marginally slower.
- `"Sinc"` — Uses the $\mathrm{sinc}(\pi x)$ kernel, where x is the pixel offset from the interpolation point and $\mathrm{sinc}(z) = \sin(z)/z$. Use of this scheme is not recommended.
- `"SincSinc"` — Uses the $\mathrm{sinc}(\pi x)\mathrm{sinc}(k\pi x)$ kernel. A valuable general-purpose scheme, intermediate in its visual effect on NDFs between the linear option and using the nearest neighbour.
- `"SincCos"` — Uses the $\mathrm{sinc}(\pi x)\cos(k\pi x)$ kernel. Gives similar results to the `"SincSinc"` scheme.
- `"SincGauss"` — Uses the $\mathrm{sinc}(\pi x)e^{-kx^2}$ kernel. Good results can be obtained by matching the FWHM of the envelope function to the point-spread function of the input data (see Parameter PARAMS).
- `"Somb"` — Uses the $\mathrm{somb}(\pi x)$ kernel, where x is the pixel offset from the interpolation point, and $\mathrm{somb}(z) = 2 * J_1(z)/z$. J_1 is the first-order Bessel function of the first kind. This scheme is similar to the `"Sinc"` scheme.
- `"SombCos"` — Uses the $\mathrm{somb}(\pi x)\cos(k\pi x)$ kernel. This scheme is similar to the `"SincCos"` scheme.
- `"BlockAve"` — Block averaging over all pixels in the surrounding N-dimensional cube.

All methods propagate variances from input to output, but the variance estimates produced by interpolation schemes need to be treated with care since the spatial smoothing produced by these methods introduces correlations variance estimates. The initial default is `"SincSinc"`. [current value]

OUT = NDF (Write)

The name for the output NDF, or the name of the single container file if CONTAINER=`TRUE`.

PARAMS(2) = _DOUBLE (Read)

An optional array which consists of additional parameters required by the Sinc, SincSinc, SincCos, SincGauss, Somb, SombCos, and Gauss methods.

PARAMS(1) is required by all the above schemes. It is used to specify how many pixels are to contribute to the interpolated result on either side of the interpolation in each dimension. Typically, a value of `2` is appropriate and the minimum allowed value is `1` (*i.e.* one pixel on each side). A value of zero or fewer indicates that a suitable number of pixels should be calculated automatically. `[0]`

PARAMS(2) is required only by the SombCos, Gauss, SincSinc, SincCos, and SincGauss schemes. For the SombCos, SincSinc, and SincCos schemes, it specifies the number of pixels at which the envelope of the function goes to zero. The minimum value is `1.0`, and the run-time default value is `2.0`. For the Gauss and SincGauss schemes, it specifies the full-width at half-maximum (FWHM) of the Gaussian envelope. The minimum value is `0.1`, and the run-time default is `1.0`. On astronomical images and spectra, good results are often obtained by approximately matching the FWHM of the envelope function, given by PARAMS(2), to the point-spread function of the input data. []

POS( ) = LITERAL (Read)
An the co-ordinates of the next slice to be extracted, in the current co-ordinate Frame of the NDF (supplying a colon `":"` will display details of the current co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas. POS is only accessed if Parameter MODE is given the value `"Interface"`. If the co-ordinates are supplied on the command line only one slice will be extracted; otherwise the application will ask for further positions which may be terminated by supplying the null value (`!`).

TITLE = LITERAL (Read)
A Title for every output NDF structure. A null value (`!`) propagates the title from the input NDF to all output NDFs. `[!]`

TOL = _DOUBLE (Read)
The maximum tolerable geometrical distortion that may be introduced as a result of approximating non-linear Mappings by a set of piece-wise linear transforms. Both algorithms approximate non-linear co-ordinate transformations in order to improve performance, and this parameter controls how inaccurate the resulting approximation is allowed to be, as a displacement in pixels of the input NDF. A value of zero will ensure that no such approximation is done, at the expense of increasing execution time. `[0.05]`

Examples:

```
pluck omc1 pos="5:35:13.7,-5:22:13.6" axes=FREQ method=sincgauss
params=[3,5] out=omc1_trap
```

The NDF omc1 is a spectral-imaging cube with (Right ascension, declination, frequency) World Co-ordinate axes. This example extracts a spectrum at RA=$5^{h}35^{m}13.7^{s}$, Dec=$-5°22'13.6''$ using the SincGauss interpolation method. Three pixels either side of the point are used to interpolate, the full-width half-maximum of the Gaussian is five pixels. The resultant spectrum called omc1_trap, is still a cube, but its spatial dimensions each only have one element.

```
pluck omc1 mode=cat incat=a axes=FREQ container out=omc1_spectra
```

This example reads the fixed positions from the positions list in file `a.FIT`. The selected spectra are stored in an HDS container file called `omc1_spectra.sdf`.

```
pluck omc1 mode=cat incat=a axes=SPEC container out=omc1_spectra
```

As the previous example, plucking spectra, this time by selecting the generic spectral axis.

```
pluck omc1 pos=3.45732E11 axes="RA,Dec" method=lin out=peakplane
```

This example extracts a plane from omc1 at frequency 3.45732E11 Hz using linear interpolation and stores it in NDF peakplane.

Notes:

- In Interface or File modes all positions should be supplied in the current co-ordinate Frame of the NDF. A description of the co-ordinate Frame being used is given if Parameter DESCRIBE is set to a TRUE value. Application WCSFRAME can be used to change the current co-ordinate Frame of the NDF before running this application if required.
- The output NDF has the same dimensionality as the input NDF, although the axes with fixed co-ordinates (those not specified by the AXES parameter) are degenerate, having bounds of 1:1. The retention of these insignificant axes enables the co-ordinates of where the slice originated to be recorded. Such fixed co-ordinates may be examined with say NDFTRACE. NDFCOPY may be used to trim the degenerate axes if their presence prevents some old non-KAPPA tasks from operating.
- In Catalogue or File modes the table file need only contain columns supplying the fixed positions. In this case the co-ordinates along the retained axes are deemed to be independent, that is they do not affect the shifts required of the other axes. In practice this assumption only affects File mode, as catalogues made with CURSOR or LISTMAKE will contain WCS information.

  In Interface mode representaive co-ordinates along retained axes are the midpoints of the bounds of an array that would contain the resampled copy of the whole input array.

Related Applications :

KAPPA: NDFCOPY, REGRID.

Implementation Status:

- The LABEL, UNITS, and HISTORY components, and all extensions are propagated. TITLE is controlled by the TITLE parameter. DATA, VARIANCE, AXIS, and WCS are propagated after appropriate modification. The QUALITY component is not propagated.
- The processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- The minimum number of dimensions in the input NDF is two.
- Processing a group of input NDFs is not supported unless CONTAINER=TRUE or when only one output NDF is created per input file.

POW
Takes the specified power of each pixel of an NDF

Description:

This routine copies the supplied input NDF , raising each value in the DATA array to the specified power. The VARIANCE component, if present, is modified appropriately. Negative data values will only generate good output pixels when the power is an integer.

Usage:

```
pow in power out
```

Parameters:

IN = NDF (Read)
The input NDF structure.

OUT = NDF (Write)
The output NDF structure.

POWER = _DOUBLE (Read)
The power.

TITLE = LITERAL (Read)
A title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

Examples:

```
pow m51 2 m51sq
```

Square all values in the NDF called m51, and store the results in the NDF called m51sq.

Related Applications :

KAPPA: ADD, CADD, CMULT, CDIV, CSUB, DIV, MATHS, MULT, SUB.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single-precision floating point, or double precision, if appropriate, but the numeric type of the input pixels is preserved in the output NDF.

PROFILE
Creates a one-dimensional profile through an *n*-dimensional NDF

Description:

This application samples an *n*-dimensional NDF at a set of positions, producing a one-dimensional output NDF containing the sample values. Nearest-neighbour interpolation is used.

The samples can be placed at specified positions within the input NDF, or can be spaced evenly along a poly-line joining a set of vertices (see Parameter MODE). The positions of the samples may be saved in an output positions list (see Parameter OUTCAT).

Usage:

```
                  { start finish [nsamp]
profile in out    {
                  { incat=?
              mode
```

Parameters:

CATFRAME = LITERAL (Read)

A string determining the co-ordinate Frame in which positions are to be stored in the output catalogue associated with Parameter OUTCAT. The string supplied for CATFRAME can be one of the following options.

- A Domain name such as SKY, AXIS, PIXEL.
- An integer value giving the index of the required Frame.
- An IRAS90 *Sky Co-ordinate System* (SCS) values such as `"EQUAT(J2000)"` (see SUN/163).

If a null (`!`) value is supplied, the positions will be stored in the current Frame. `[!]`

CATEPOCH = _DOUBLE (Read)

The epoch at which the sky positions stored in the output catalogue were determined. It will only be accessed if an epoch value is needed to qualify the co-ordinate Frame specified by COLFRAME. If required, 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.

FINISH = LITERAL (Read)

The co-ordinates of the last sample in the profile, in the current co-ordinate Frame of the NDF (supplying `":"` will display details of the required co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces. This parameter is only accessed if Parameter MODE is set to `"Curve"` and a null (`!`) value is given for INCAT. If the last (top-right) pixel in the NDF has valid co-ordinates in the current co-ordinate Frame of the NDF, then these co-ordinates will be used as the suggested default. Otherwise there will be no suggested default.

GEODESIC = _LOGICAL (Read)
If TRUE then the line segments which form the profile will be geodesic curves within the current co-ordinate Frame of the NDF. Otherwise, the line segments are simple straight lines. This parameter is only accessed if Parameter MODE is set to "Curve". As an example, consider a profile consisting of a single line segment which starts at RA=0h DEC=+80d and finishes at RA=12h DEC=+80d. If GEODESIC is FALSE, the line segment will be a line of constant declination, *i.e.* the "straight" line from the position (0,80) to the position (12, 80), passing through (1, 80), (2, 80), *etc*. If GEODESIC is TRUE, then the line segment will be the curve of shortest distance on the celestial sphere between the start and end. In this particaular case, this will be a great circle passing through the north celestial pole. [FALSE]

IN = NDF (Read)
Input NDF structure containing the data to be profiled.

INCAT = FILENAME (Read)
A catalogue containing a set of vertices or sample positions defining the required profile. The file should be in the format of a *positions list* such as produced by applications CURSOR and LISTMAKE. If a null value (!) is given then Parameters START and FINISH will be used to obtain the vertex positions. If Parameter MODE is given the value "Curve", then the Parameter INCAT is only accessed if a value is given for it on the command line (otherwise a null value is assumed).

MODE = LITERAL (Read)
The mode by which the sample positions are selected. The alternatives are listed below.

- "Curve" — The samples are placed evenly along a curve specified by a set of vertices obtained from the user. The line segments joining these vertices may be linear or geodesic (see Parameter GEODESIC). Multiple vertices may be supplied using a text file (see Parameter INCAT). Alternatively, a single line segment can be specified using Parameters START and FINISH. The number of samples to take along the curve is specified by Parameter NSAMP.
- "Points" — The positions at which samples should be taken are given explicitly by the user in a text file (see Parameter INCAT). No other sample positions are used.

["Curve"]

NSAMP = _INTEGER (Read)
The number of samples required along the length of the profile. The first sample is at the first supplied vertex, and the last sample is at the last supplied vertex. The sample positions are evenly spaced within the current co-ordinate Frame of the NDF. If a null value is supplied, a default value is used equal to one more than the length of the profile in pixels. This is only accessed if Parameter MODE is given the value "Curve". [!]

OUT = NDF (Write)
The output NDF. This will be one-dimensional with length specified by Parameter NSAMP.

OUTCAT = FILENAME (Write)
An output positions list in which to store the sample positions. This is the name of a

catalogue which can be used to communicate positions to subsequent applications. It includes information describing the available WCS co-ordinate Frames as well as the positions themselves. If a null value is supplied, no output positions list is produced. See also Parameter CATFRAME. [!]

START = LITERAL (Read)
The co-ordinates of the first sample in the profile, in the current co-ordinate Frame of the NDF (supplying ":" will display details of the required co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces. This parameter is only accessed if Parameter MODE is set to "Curve" and a null (!) value is given for INCAT. If the first (bottom-left) pixel in the NDF has valid co-ordinates in the current co-ordinate Frame of the NDF, then these co-ordinates will be used as the suggested default. Otherwise there will be no suggested default.

Examples:

```
profile my_data prof "0 0" "100 100" 40 outcat=samps
```

Creates a one-dimensional NDF called prof, holding a profile of the data values in the input NDF my_data along a profile starting at pixel co-ordinates [0.0, 0.0] and ending at pixel co-ordinates [100.0, 100.0]. The profile consists of forty samples spread evenly (in the pixel co-ordinate Frame) between these two positions. This example assumes that the current co-ordinate Frame in the NDF my_data represents pixel co-ordinates. This can be ensured by issuing the command "wcsframe my_data pixel" before running profile. A FITS binary catalogue is created called samps.FIT containing the positions of all samples in the profile, together with information describing all the co-ordinate Frames in which the positions of the samples are known. This file may be examined using application LISTSHOW.

```
profile my_data prof "15:32:47 23:40:08" "15:32:47 23:42"
```

This example is the same as the last one except that it is assumed that the current co-ordinate Frame in the input NDF my_data is an equatorial (RA/DEC) system. It creates a one-dimensional profile starting at RA=15:32:47 DEC=23:40:08, and ending at the same RA and DEC=23:42:00. The number of points in the profile is determined by the resolution of the data.

```
profile allsky prof incat=prof_path npoint=200 geodesic outcat=aa.fit
```

This examples creates a profile of the NDF allsky through a set of points given in a FITS binary catalogue called prof_path.FIT. Such catalogues can be created (for example) using application CURSOR. Each line segment is a geodesic curve. The profile is sampled at 200 points. The samples positions are written to the output positions list aa.fit.

```
profile allsky2 prof2 mode=point incat=aa.fit
```

This examples creates a profile of the NDF allsky2 containing samples at the positions given in the positions list aa.fit. Thus, the profiles created by this example and

the previous example will sample the two images allsky and allsky2 at the same positions and so can be compared directly.

Notes:

- This application uses the conventions of the CURSA package for determining the formats of input and output positions list catalogues. If a file type of .fit is given, then the catalogue is assumed to be a FITS binary table. If a file type of .txt is given, then the catalogue is assumed to be stored in a text file in *Small Text List* (STL) format. If no file type is given, then `.fit` is assumed.

Related Applications :

KAPPA: LINPLOT, CURSOR, LISTMAKE, LISTSHOW; CURSA: XCATVIEW.

Implementation Status:

- This routine correctly processes the DATA, VARIANCE, WCS, LABEL, TITLE, and UNITS components of the NDF.
- All non-complex numeric data types can be handled. Only double-precision floating-point data can be processed directly. Other non-complex data types will undergo a type conversion before the profile is produced.

PROVADD
Stores provenance information in an NDF

Description:

This application modifies the provenance information stored in an NDF. It records a second specified NDF as a direct parent of the first NDF. If an NDF has more than one direct parent then this application should be run multiple times, once for each parent.

Usage:

```
provadd ndf parent creator isroot moretext
```

Parameters:

CREATOR = LITERAL (Read)

A text identifier for the software that created the main NDF (usually the name of the calling application). The format of the identifier is arbitrary, but the form `"PACKAGE:COMMAND"` is recommended. If a null (`!`) value is supplied, no creator information is stored. `[!]`

ISROOT = _LOGICAL (Read)

If `TRUE`, then the NDF given by Parameter PARENT will be treated as a root NDF. That is, any provenance information within PARENT describing its own parents is ignored. If `FALSE`, then any provenance information within PARENT is copied into the main NDF. PARENT is then a root NDF only if it contains no provenance information. `[FALSE]`

MORETEXT = GROUP (Read)

A group of "keyword=value" strings that give additional information about the parent NDF, and how it was used in the creation of the main NDF. If supplied, this information is stored with the provenance in the main NDF.

The supplied value should be either a comma-separated list of strings, or the name of a text file preceded by an up-arrow character `"^"`, containing one or more comma-separated list of strings. Each string is either a "keyword=value" setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner (any blank lines or lines beginning with # are ignored). Within a text file, newlines can be used as delimiters as well as commas.

Each individual setting should be of the form:

<keyword>=<value>

where <keyword> is either a simple name, or a dot-delimited hierarchy of names (*e.g.* `"camera.settings.exp=1.0"`). The <value> string should not contain any commas. `[!]`

NDF = NDF (Read and Write)

The NDF which is to be modified.

PARENT = NDF (Read)

An NDF that is to be recorded as a direct parent of the NDF given by Parameter NDF.

Examples:

```
provadd m51_ff ff
```

Records the fact that NDF ff was used in the creation of NDF m51_ff.

Notes:

Provenance information is stored in an NDF extension called PROVENANCE, and is propagated automatically by all KAPPA applications.

Related Applications :

KAPPA: PROVMOD, PROVSHOW, HISCOM.

PROVMOD
Modifies provenance information for an NDF

Description:

This application modifies the provenance information stored in the PROVENANCE extension of an NDF.

Usage:

```
provmod ndf ancestor path
```

Parameters:

ANCESTOR = LITERAL (Read)

Specifies the indices of one or more ancestors that are to be modified. An index of zero refers to the supplied NDF itself. A positive index refers to one of the NDFs listed in the ANCESTORS table in the PROVENANCE extension of the NDF. The maximum number of ancestors is limited to 100 unless "ALL" or "*" is specified. The supplied parameter value can take any of the following forms.

- "ALL" or "*" — All ancestors.
- "xx,yy,zz" — A list of ancestor indices.
- "xx:yy" — Ancestor indices between *xx* and *yy* inclusively. When *xx* is omitted, the range begins from 0; when *yy* is omitted, the range ends with the maximum value it can take, that is the number of ancestors described in the PROVENANCE extension.
- Any reasonable combination of above values separated by commas. ["ALL"]

CREATOR = LITERAL (Read)

If the supplied string includes no equals signs, then it is a new value for the "CREATOR" string read from each of the ancestors being modified. If the supplied string includes one or more equals signs, then it specifies one or more substitutions to be performed on the "CREATOR" string read from each of the ancestors being modified. See "Substitution Syntax" below. If null (!) is supplied, the CREATOR item is left unchanged. [!]

DATE = LITERAL (Read)

If the supplied string includes no equals signs, then it is a new value for the "DATE" string read from each of the ancestors being modified. If the supplied string includes one or more equals signs, then it specifies one or more substitutions to be performed on the "DATE" string read from each of the ancestors being modified. See "Substitution Syntax" below. If null (!) is supplied, the DATE item is left unchanged. [!]

MORETEXT = GROUP (Read)

This parameter is accessed only if a single ancestor is being modified (see Parameter ANCESTORS). It gives information to store in the MORE component of the ancestor (any existing information is first removed). If a null (!) value is supplied, then existing MORE component is left unchanged.

The supplied value should be either a comma-separated list of strings, or the name of a text file preceded by an up-arrow character "^", containing one or more comma-separated list of strings. Each string is either a "keyword=value" setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner (any blank lines or lines beginning with # are ignored). Within a text file, newlines can be used as delimiters as well as commas.

Each individual setting should be of the form:

<keyword>=<value>

where <keyword> is either a simple name, or a dot-delimited hierarchy of names (*e.g.* "camera.settings.exp=1.0"). The <value> string should not contain any commas. [!]

NDF = NDF (Update)

The NDF data structure.

PATH = LITERAL (Read)

If the supplied string includes no equals signs, then it is a new value for the "PATH" string read from each of the ancestors being modified. If the supplied string includes one or more equals signs, then it specifies one or more substitutions to be performed on the "PATH" string read from each of the ancestors being modified. See "Substitution Syntax" below. If null (!) is supplied, the PATH item is left unchanged. [!]

Examples:

```
provmod ff path=/home/dsb/real-file.sdf
```

This modifies any ancestor within the NDF called ff by setting its PATH to "/home/dsb/real-file.sdf".

```
provmod ff ancestor=3 moretext="obsidss=acsis_00026_20080322T055855_1"
```

This modifies ancestor Number 3 by storing a value of acsis_00026_20080322T055855_1 for key obsidss within the additonal information for the ancestor. Any existing additional information is removed.

```
provmod ff path='(_x)$=_y'
```

This modifies any ancestor within the NDF called ff that has a path ending in "_x" by replacing the final "_x" with "_y".

```
provmod ff path='(.*)_(.*)=$2=$1'
```

This modifies any ancestor within the NDF called ff that has a path consisting of two parts separated by an underscore by swapping the parts. If there is more than one underscore in the ancestor path, then the final underscore is used (because the initial quantifier ".*" is greedy).

```
provmod ff path='(.*?)_(.*)=$2=$1'
```

This modifies any ancestor within the NDF called ff that has a path consisting of two parts separated by an underscore by swapping the parts. If there is more than one underscore in the ancestor path, then the first underscore is used (because the initial quantifier "`.*?`" is not greedy).

Substitution Syntax :

The syntax for the CREATOR, DATE, and PATH parameter values is a minimal form of regular expression. The following atoms are allowed.

- "`[chars]`" — Matches any of the characters within the brackets.
- "`[^chars]`" — Matches any character that is not within the brackets (ignoring the initial "`^`" character).
- "`.`" — Matches any single character.
- "`\d`" — Matches a single digit.
- "`\D`" — Matches anything but a single digit.
- "`\w`" — Matches any alphanumeric character, and "`_`".
- "`\W`" — Matches anything but alphanumeric characters, and "`_`".
- "`\s`" — Matches white space.
- "`\S`" — Matches anything but white space.

Any other character that has no special significance within a regular expression matches itself. Characters that have special significance can be matched by preceding them with a backslash (\) in which case their special significance is ignored (note, this does not apply to the characters in the set *dDsSwW*).

Note, minus signs ("`-`") within brackets have no special significance, so ranges of characters must be specified explicitly.

The following quantifiers are allowed.

- "`*`" — Matches zero or more of the preceding atom, choosing the largest possible number that gives a match.
- "`*?`"— Matches zero or more of the preceding atom, choosing the smallest possible number that gives a match.
- "`+`" — Matches one or more of the preceding atom, choosing the largest possible number that gives a match.
- "`+?`"— Matches one or more of the preceding atom, choosing the smallest possible number that gives a match.
- "`?`" — Matches zero or one of the preceding atom.
- "`{n}`" — Matches exactly *n* occurrences of the preceding atom.

The following constraints are allowed.

- "`^`" — Matches the start of the test string.
- "`$`" — Matches the end of the test string.

Multiple templates can be concatenated, using the "|" character to separate them. The test string is compared against each one in turn until a match is found.

A template should use parentheses to enclose the sub-strings that are to be replaced, and the set of corresponding replacement values should be appended to the end of the string, separated by "=" characters. The section of the test string that matches the first parenthesised section in the template string will be replaced by the first replacement string. The section of the test string that matches the second parenthesised section in the template string will be replaced by the second replacement string, and so on.

The replacement strings can include the tokens "$1","$2", *etc.* The section of the test string that matched the corresponding parenthesised section in the template is used in place of the token.

See the "Examples" section above for how to use these facilities.

Related Applications :

KAPPA: PROVADD, PROVREM, PROVSHOW.

PROVREM
Removes selected provenance information from an NDF

Description:

This application removes selected ancestors, either by hiding them, or deleting them from the provenance information stored in a given NDF. The 'generation gap' caused by removing an ancestor is bridged by assigning all the direct parents of the removed ancestor to each of the direct children of the ancestor.

The ancestors to be removed can be specified either by giving their indices (Parameter ANCESTOR), or by comparing each ancestor with a supplied pattern matching template (Parameter PATTERN).

If an ancestor is hidden rather than deleted (see Parameter HIDE), the ancestor is retained within the NDF, but a flag is set telling later applications to ignore the ancestor (exactly how the flag is used will depend on the particular application).

Usage:

```
provrem ndf pattern item
```

Parameters:

ANCESTOR = LITERAL (Read)

Specifies the indices of one or more ancestors that are to be removed. If a null (`!`) value is supplied, the ancestors to be removed are instead determined using the PATTERN parameter. Each supplied index must be positive and refers to one of the NDFs listed in the ANCESTORS table in the PROVENANCE extension of the NDF (including any hidden ancestors). Note, if ancestor indices are determined using the PROVSHOW command, then PROVSHOW should be run with the HIDE parameter set to `FALSE`; otherwise incorrect ancestor indices may be determined, resulting in the wrong ancestors being removed by PROVREM.

The maximum number of ancestors that can be removed is limited to 100 unless `"LL"`, `"*"` or `!` is specified. The supplied parameter value can take any of the following forms.

- `"ALL"` or `"*"` — All ancestors.
- `"xx,yy,zz"` — A list of ancestor indices.
- `"xx:yy"` — Ancestor indices between *xx* and *yy* inclusively. When *xx* is omitted, the range begins from 0; when *yy* is omitted the range ends with the maximum value it can take, that is the number of ancestors described in the PROVENANCE extension.
- Any reasonable combination of above values separated by commas. `[!]`

HIDE = _LOGICAL (Read)

If `TRUE`, then the ancestors are not deleted, but instead have a flag set indicating that they have been hidden. All information about hidden ancestors is retained unchanged, and can be viewed using PROVSHOW if the HIDE parameter is set `FALSE` when running PROVSHOW. `[FALSE]`

ITEM = LITERAL (Read)
Specifies the item of provenance information that is checked against the pattern matching template specified for Parameter PATTERN. It can be "PATH", "CREATOR" or "DATE". ["PATH"]

NDF = NDF (Update)
The NDF data structure.

PATTERN = LITERAL (Read)
Specifies a pattern matching template using the syntax described below in "Pattern Matching Syntax". Each ancestor listed in the PROVENANCE extension of the NDF is compared with this template, and each ancestor that matches is removed. The item of provenance information to be compared to the pattern is specified by Parameter ITEM.

REMOVE = _LOGICAL (Read)
If TRUE, then the ancestors specified by Parameter PATTERN or ANCESTORS are removed. Otherwise, these ancestors are retained and all other ancestors are removed. [TRUE]

Examples:

```
provrem ff ancestor=1
```

This removes the first ancestor from the NDF called ff.

```
provrem ff ancestor=all
```

This erases all provenance information.

```
provrem ff pattern='_xb$|_yb$' hide
```

This hides, but does not permanently delete, all ancestors that have paths that end with "_xb" or "_yb". Note, provenance paths do not include a trailing ".sdf" string.

```
provrem ff pattern='_ave'
```

This removes all ancestors that have paths that contain the string "_ave" anywhere.

```
provrem ff pattern='_ave' remove=no
```

This removes all ancestors that have paths that do not contain the string "_ave" anywhere.

```
provrem ff pattern='_d[^/]*$'
```

This removes all ancestors that have file base-names that begin with "_d" . The pattern matches "_d" followed by any number of characters that are not "/", followed by the end of the string.

```
provrem ff pattern='^m51|^m31'
```

This removes all ancestors that have paths that begin with "m51" or "m31".

Pattern Matching Syntax :

The syntax for the PATTERN parameter value is a minimal form of regular expression. The following atoms are allowed.

- "[chars]" — Matches any of the characters within the brackets.
- "[^chars]" — Matches any character that is not within the brackets (ignoring the initial "^" character).
- "." — Matches any single character.
- "\d" — Matches a single digit.
- "\D" — Matches anything but a single digit.
- "\w" — Matches any alphanumeric character, and "_".
- "\W" — Matches anything but alphanumeric characters, and "_".
- "\s" — Matches white space.
- "\S" — Matches anything but white space.

Any other character that has no special significance within a regular expression matches itself. Characters that have special significance can be matched by preceding them with a backslash (\) in which case their special significance is ignored (note, this does not apply to the characters in the set dDsSwW).

Note, minus signs ("-") within brackets have no special significance, so ranges of characters must be specified explicitly.

The following quantifiers are allowed.

- "*" — Matches zero or more of the preceding atom, choosing the largest possible number that gives a match.
- "*?"— Matches zero or more of the preceding atom, choosing the smallest possible number that gives a match.
- "+" — Matches one or more of the preceding atom, choosing the largest possible number that gives a match.
- "+?"— Matches one or more of the preceding atom, choosing the smallest possible number that gives a match.
- "?" — Matches zero or one of the preceding atom.
- "{n}" — Matches exactly *n* occurrences of the preceding atom.

The following constraints are allowed.

- "^" — Matches the start of the test string.
- "$" — Matches the end of the test string.

Multiple templates can be concatenated, using the "|" character to separate them. The test string is compared against each one in turn until a match is found.

Related Applications :

KAPPA: PROVADD, PROVMOD, PROVSHOW.

PROVSHOW
Displays provenance information for an NDF

Description:

This application displays details of the NDFs that were used in the creation of the supplied NDF. This information is read from the PROVENANCE extension within the NDF, and includes both immediate parent NDFs and older ancestor NDFs (*i.e.* the parents of the parents, *etc.*).

Each displayed NDF (see Parameter SHOW) is described in a block of lines. The first line holds an integer index for the NDF followed by the path to that NDF. Note, this path is where the NDF was when the provenance information was recorded. It is of course possible that the NDF may subsequently have been moved or deleted.

The remaining lines in the NDF description are as follows.

- `"Parents"` — A comma-separated list of integers that are the indices of the immediate parents of the NDF. These are the integers that are displayed on the first line of each NDF description.
- `"Date"` — The formatted UTC date and time at which the provenance information for the NDF was recorded.
- `"Creator"` — A string identifying the software that created the NDF.
- `"More"` — A summary of any extra information about the NDF stored with the provenance information. In general this may be an arbitrary HDS structure and so full details cannot be given on a single line. The HDSTRACE command can be used to examine the MORE field in detail. To see full details of the NDF with `"ID"` value of `12` (say), enter (from a UNIX shell) `"hdstrace fred.more.provenance.ancestors'(12)'"`, where `fred` is the name of the NDF supplied for Parameter NDF. If the NDF has no extra information, this item will not be present.
- `"History"` — This is only displayed if Parameter HISTORY is set to a `TRUE` value. It contains information copied from the HISTORY component of the ancestor NDF. See Parameter HISTORY.

In addition, a text file can be created containing the paths for the direct parents of the supplied NDF. See Parameter PARENTS.

Usage:

```
provshow ndf [show]
```

Parameters:

DOTFILE = FILENAME (Read)

Name of a new text file in which to store a description of the provenance tree using the "dot" format. This file can be visualised using third-party tools such as Graphviz, ZGRViewer, OmniGraffle, *etc.*

HIDE = _LOGICAL (Read)
If TRUE, then any ancestors which are flagged as 'hidden' (for example, using PROVREM) are excluded from the display. If FALSE, then all requested ancestors, whether hidden or not, are included in the display (but hidden ancestors will be highlighted as such). Note, choosing to exclude hidden ancestors may change the index displayed for each ancestor. The default is to display hidden ancestors if and only if history is being displayed (see Parameter HISTORY). []

HISTORY = _LOGICAL (Read)
If TRUE, any history records stored with each ancestor are included in the displayed information. Since the amount of history information displayed can be large, and thus swamp other information, the default is not to display history information.

When an existing NDF is used in the creation of a new NDF, the provenance system will copy selected records from the HISTORY component of the existing NDF and store them with the provenance information in the new NDF. The history records copied are those that describe operations performed on the existing NDF itself. Inherited history records that describe operations performed on ancestors of the existing NDF are not copied. [FALSE]

INEXT = LITERAL (Read)
Determines which ancestor to display next. Only used if Parameter SHOW is set to "Tree". The user is re-prompted for a new value for this parameter after each NDF is displayed. The new value should be the integer identifier for one of the parents of the currently displayed NDF. Alternatively, the string "up" can be supplied, causing the previously displayed NDF to be displayed again.

NDF = NDF (Read)
The NDF data structure.

PARENTS = FILENAME (Read)
Name of a new text file in which to put the paths to the direct parents of the supplied NDF. These are written one per line with no extra text. If null, no file is created. [!]

SHOW = LITERAL (Read)
Determines which ancestors are displayed on the screen. It can take any of the following case-insensitive values (or any abbreviation).

- "All" — Display all ancestors, including the supplied NDF itself.
- "Roots" — Display only the root ancestors (i.e. ancestors that do not themselves have any recorded parents). The supplied NDF itself is not displayed.
- "Parents" — Display only the direct parents of the supplied NDF. The supplied NDF itself is not displayed.
- "Tree" — Display the top level NDF and then asks the user which parent to display next (see Parameter INEXT). The whole family tree can be navigated in this way.

["All"]

Examples:

```
provshow m51
```

This displays information about the NDF m51, and all its recorded ancestors.

```
provshow m51 roots
```

This displays information about the root ancestors of the NDF m51.

```
provshow m51 parents
```

This displays information about the direct parents of the NDF m51.

Notes:

- An input NDF is included in the provenance of an output NDF only if the DATA component of the input NDF is mapped for read or update access by the application. In other words, input NDFs which are accessed only for their metadata (*e.g.* WCS information) are not included in the output provenance of an application.
- If a KAPPA application uses one or more input NDFs to create an output NDF, the output NDF may or may not contain provenance information depending on two things: 1) whether any of the input NDFs already contain provenance information, and 2) the value of the AUTOPROV environment variable. It is usually necessary to set the AUTOPROV variable to "1" in order to create output NDFs that contain provenance information. The exception to this if you are supplied with NDFs from another source that already contain provenance. If such NDFs are used as inputs to KAPPA applications, then the output NDFs will contain provenance even if the AUTOPROV variable is unset. However, setting AUTOPROV to "0" will always prevent provenance information being stored in the output NDFs.
- Some other packages, such as CCDPACK, follow the same strategy for creating and propagating provenance information.

Related Applications :

KAPPA: PROVADD, HISLIST.

PSF
Determines the parameters of a model star profile by fitting star images in a two-dimensional NDF

Description:

This application finds a set of parameters to describe a model Gaussian star image. It can be used for profile-fitting stellar photometry, to evaluate correction terms to aperture photometry, or for filtering.

The model has a Sérsic radial profile:

$$D = A \exp^{-0.5\,(r/\sigma)^{\gamma}}$$

where r is calculated from the true radial distance from the star centre allowing for image ellipticity, σ is the Gaussian precision constant or profile width. The application combines a number of star images you specify and determines a mean seeing-disc size, radial fall-off parameter (γ), axis ratio, and orientation of a model star image.

A table, giving details of the seeing and ellipticity of each star image used can be reported to an output text file. This table indicates if any star could not be used. Reasons for rejecting stars are too-many bad pixels present in the image, the star is too close to the edge of the data array, the 'star' is a poor fit to model or it could not be located.

An optional plot of the mean profile and the fitted function may be produced. The two-dimensional point-spread function may be stored in an NDF for later use, as may the one-dimensional fitted profile.

Usage:

```
psf in incat [device] [out] [cut] [range] [isize] [poscols]
```

Parameters:

AXES = _LOGICAL (Read)

TRUE if labelled and annotated axes are to be drawn around the plot. The width of the margins left for the annotation may be controlled using Parameter MARGIN. The appearance of the axes (colours, founts, *etc.*) can be controlled using the Parameter STYLE. The dynamic default is TRUE if CLEAR is TRUE, and FALSE otherwise. []

CLEAR = _LOGICAL (Read)

If TRUE the current picture is cleared before the plot is drawn. If CLEAR is FALSE not only is the existing plot retained, but also an attempt is made to align the new picture with the existing picture. Thus you can generate a composite plot within a single set of axes, say using different colours or modes to distinguish data from different datasets. [TRUE]

COFILE = FILENAME (Read)

Name of a text file containing the co-ordinates of the stars to be used. It is only accessed if Parameter INCAT is given a null (!) value. Each line should contain the formatted axis values for a single position, in the current Frame of the NDF. Columns

can be separated by spaces, tabs or commas. The file may contain comment lines with the first character # or `!`. Other columns may be included in the file, in which case the columns holding the required co-ordinates should be specified using Parameter POSCOLS.

CUT = _REAL (Read)
This parameter controls the size of the output NDF. If it is null, `!`, the dimension of the square NDF will be the size of the region used to calculate the radial profile, which usually is given by RANGE $*$ width in pixels $*$ AXISR, unless truncated. If CUT has a value it is the threshold which must be included in the PSF NDF, and it is given as the fraction of the peak amplitude of the PSF. For example, if CUT=`0.5` the NDF would contain the point-spread function to half maximum. CUT must be greater than 0 and less than 1. The suggested default is 0.0001. `[!]`

DEVICE = DEVICE (Read)
The graphics workstation on which to produce a plot of the mean radial profile of the stars and the fitted function. A null (`!`) name indicates that no plot is required. [current graphics device]

GAUSS = _LOGICAL (Read)
If `TRUE`, the γ coefficient is fixed to be 2; in other words the best-fitting two-dimensional Gaussian is evaluated. If `FALSE`, γ is a free parameter of the fit, and the derived value is returned in Parameter GAMMA. `[FALSE]`

IN = NDF (Read)
The NDF containing the star images to be fitted.

INCAT = FILENAME (Read)
A catalogue containing a positions list (such as produced by applications CURSOR, LISTMAKE) giving the star positions to use. If a null (`!`) value is supplied Parameter COFILE will be used to get the star positions from a simple text file.

ISIZE = _INTEGER (Read)
The side of the square area to be used when forming the marginal profiles for a star image, given as a number of pixels. It should be sufficiently large to contain the entire star image. It should be an odd number and must lie in the range from 3 to 101. `[15]`

LOGFILE = FILENAME (Read)
Text file to contain the table of parameters for each star. A null (`!`) name indicates that no log file is required. `[!]`

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. If a null (`!`) value is supplied, the value used is `0.15` (for all edges) if either annotated axes or a key are produced, and zero otherwise. [current value]

MARKER = _INTEGER (Read)
The PGPLOT marker type to use for the data values in the plot. [current value]

MINOR = _LOGICAL (Read)
If MINOR is `TRUE` the horizontal axis of the plot is annotated with distance along

the minor axis from the centre of the PSF. If MINOR is FALSE, the distance along the major axis is used. [TRUE]

NORM = _LOGICAL (Read)

If TRUE, the model PSF is normalized so that it has a peak value of unity. Otherwise, its peak value is equal to the peak value of the fit to the first usable star, in the data units of the input NDF. [TRUE]

OUT = NDF (Write)

The NDF containing the fitted point-spread function evaluated at each pixel. If null, !, is entered no output NDF will be created. The dimensions of the array are controlled by Parameter CUT. The pixel origin is chosen to align the model PSF with the first fitted star in pixel co-ordinates, thus allowing the NDF holding the model PSF to be compared directly with the input NDF. A WCS component is stored in the output NDF holding a copy of the input WCS component. An additional Frame with Domain name OFFSET is added, and is made the current Frame. This Frame measures the distance from the PSF centre in the units in which the FWHM is reported. These changes allows the NDF holding the model PSF to be compared directly with the input NDF. [!]

POSCOLS = _INTEGER (Read)

Column positions of the co-ordinates (x then y) in an input record of the file specified by Parameter COFILE. The columns must be different amongst themselves. If there is duplication new values will be requested. Only accessed if INCAT is given a null (!) value. If a null (!) value is supplied for POSCOLS, the values [1,2] will be used. [!]

PROFOUT = NDF (Write)

The NDF containing the one-dimensional fitted profile as displayed in the plot. If null, !, is entered no output NDF will be created. The DATA component of this NDF holds the fitted PSF value at each radial bin. The VARIANCE component holds the square of the residuals between the fitted values and the binned values derived from the input NDF. An AXIS component is included in the NDF containing the radial distance as displayed in the plot. [!]

RANGE = _REAL (Read)

The number of image profile widths out to which the radial star profile is to be fitted. (There is an upper limit of 100 pixels to the radius at which data are actually used.) [4.0]

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use when drawing the annotated axes, data values, and the model profile.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes.

All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported). The appearance of the model curve is controlled by the attributes Colour(Curves), Width(Curves), *etc.* (the synonym Line may be used in place of Curves). The appearance of the markers representing the real data is controlled by Colour(Markers), Width(Markers), *etc.* (the synonym Symbols may be used in place of Markers). [current value]

TITLE = LITERAL (Read)

The title for the NDF to contain the fitted point-spread function. If null, !, is entered the NDF will not contain a title. ["KAPPA - PSF"]

USEAXIS = GROUP (Read)

USEAXIS is only accessed if the current co-ordinate Frame of the NDF has more than two axes. A group of two strings should be supplied specifying the two axes which are to be used when determining distances, reporting positions, *etc*. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (!) value is supplied, the axes with the same indices as the two significant NDF pixel axes are used. [!]

XLEFT = _DOUBLE (Read)

The axis value to place at the left hand end of the horizontal axis of the plot. If a null (!) value is supplied, a suitable default value will be found and used. 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 of the plot. If a null (!) value is supplied, a suitable default value will be found and used. 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 of the plot. If a null (!) value is supplied, a suitable default value will be found and used. 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 of the plot. If a null (!) value is supplied, a suitable default value will be found and used. The value supplied may be greater than or less than the value supplied for YBOT. [!]

Results Parameters:

AMP1 = _REAL (Write)
The fitted peak amplitude of the first usable star, in the data units of the input NDF.

AXISR = _REAL (Write)
The axis ratio of the star images: the ratio of the major axis length to that of the minor axis.

CENTRE = LITERAL (Write)
The formatted co-ordinates of the first fitted star position, in the current Frame of the NDF.

FWHM = _REAL (Write)
The seeing-disc size: the full width at half maximum across the minor axis of the stars. It is in units defined by the current Frame of the NDF. For instance, a value in arcseconds will be reported if the current Frame is a SKY Frame, but pixels will be used if it is a PIXEL Frame.

GAMMA = _REAL (Write)
The radial fall-off parameter, γ, of the star images. See the description for more details. A γ of two would be a Gaussian.

ORIENT = _REAL (Write)
The orientation of the major axis of the star images, in degrees. If the current Frame of the NDF is a SKY Frame, this will be a position angle (measured from north through east). Otherwise, it will be measured from the positive direction of the first current Frame axis (`"X"`) towards the second current Frame axis (`"Y"`).

TOTAL = _REAL (Write)
The flux of the fitted function integrated to infinite radius. Its unit is the product of the data unit of the input NDF and the square of the radial unit, such as pixel or arcsec, for the current WCS Frame, when NORM=`FALSE`. When NORM=`TRUE`, TOTAL is just measured in the squared radial unit. Therefore, for direct comparison of total flux, the same units must be used.

Examples:

```
psf ngc6405i starlist.FIT \
```

Derives the mean point-spread function for the stars images in the NDF called ngc6405i that are situated near the co-ordinates given in the positions list `starlist.FIT`. A plot of the profile is drawn on the current graphics device.

```
psf ngc6405i starlist device=!
```

As above but there is no graphical output, and the file type of the input positions list is defaulted.

```
psf ngc6405i cofile=starlist.dat gauss \
```

As the first example, except the psf is fitted to a two-dimensional Gaussian, and the positions are given in a simple text file (`starlist.dat`) instead of a positions list.

```
psf incat=starlist.FIT in=ngc6405i logfile=fit.log fwhm=(seeing) \
```

As the first example, but the results, including the fits to each star, are written to the text file `fit.log`. The full-width half-maximum is written to the ICL variable SEEING rather than the parameter file.

```
psf ngc6405i starlist isize=31 style="'title=Point spread function'"
```

As the first example, but the area including a star image is 31 pixels square, say because the seeing is poor or the pixels are smaller than normal. The graph is titled `"Point spread function"`.

Notes:

- Values for the FWHM seeing are given in arcseconds if the Current co-ordinate Frame of the NDF is a SKY Frame.
- The stars used to determine the mean image parameters should be chosen to represent those whose magnitudes are to be found using a stellar photometry application, and to be sufficiently bright, uncrowded, and noise-free to allow an accurate fit to be made.
- It is assumed that the image scale does not vary significantly across the image.
- The method to calculate the fit is as follows.

  - Marginal profiles of each star image are formed in four directions: at 0, 45, 90 and 135 degrees to the x axis. The profiles are cleaned via an iterative modal filter that removes contamination such as neighbouring stars; moving from the centre of the star, the filter prevents each data point from exceeding the maximum of the two previous data values.
  - A Gaussian curve and background is fitted to each profile iteratively refining the parameters until parameters differ by less than 0.1 per cent from the previous iteration. If convergence is not met after fifteen iterations, each fit parameter is approximately the average of its last pair of values. The initial background is the lower quartile.
    Using the resulting four Gaussian centres, a mean centre is found for each star. Iterations cease when the mean centroid position shifts by less 0.001 from the previous iteration, or after three iterations if the nominal tolerance is not achieved.
  - The four Gaussian widths of all the stars are combined modally, using an amplitude-weighted average with rejection of erroneous data (using a maximum-likelihood function for a statistical model in which any of the centres has a constant probability of being corrupt). From the average widths along the four profiles, the seeing-disc size, axis ratio and axis inclination are calculated.
  - The data surrounding each star is then binned into isophotal zones which are elliptical annuli centred on the star—the ellipse parameters being those just calculated. The data in each zone is processed to remove erroneous points (using the aforementioned maximum-likelihood function) and to find an average value.

A Gaussian profile is fitted to these average values and the derived amplitude is used to normalise the values to an amplitude of unity. The normalised values are put into bins together with the corresponding data from all other stars and these binned data represent a weighted average radial profile for the set of stars, with the image ellipticity removed. Finally a radial profile is fitted to these data, giving the radial profile parameter gamma and a final re-estimate of the seeing-disc size.

- If a plot was requested the application stores two pictures in the graphics database in the following order: a FRAME of the specified size containing the title, annotated axes, and line plot; and a DATA picture, containing just the data plot. Note, the FRAME picture is only created if annotated axes have been drawn, or if non-zero margins were specified using Parameter MARGIN. The NDF associated with the plot is not stored by reference with the DATA picture. On exit the current database picture for the chosen device reverts to the input picture.

Related Applications :

PHOTOM; Starman.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, WCS, and TITLE components of an NDF data structure.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. The output point-spread-function NDF has the same type as the input NDF.

QUALTOBAD
Set selected NDF pixels bad on the basis of Quality

Description:

This routine produces a copy of an input NDF in which selected pixels are set bad. The selection is based on the values in the QUALITY component of the input NDF; any pixel which holds a set of qualities satisfying the quality expression given for Parameter QEXP is set bad in the output NDF. Named qualities can be associated with specified pixels using the SETQUAL task.

Usage:

```
qualtobad in out qexp
```

Parameters:

IN = NDF (Read)
The input NDF.

OUT = NDF (Write)
The output NDF.

QEXP = LITERAL (Read)
The quality expression.

TITLE = LITERAL (Read)
Title for the output NDF. A null (`!`) value will cause the input title to be used. `[!]`

Examples:

```
qualtobad m51* *_clean saturated.or.glitch
```

This example copies all NDFs starting with the string `"m51"` to a set of corresponding output NDFs. The name of each output NDF is formed by extending the name of the input NDF with the string `"_clean"`. Any pixels which hold either of the qualities `"saturated"` or `"glitch"` are set to the bad value in the output NDFs.

Related Applications :

KAPPA: REMQUAL, SETBB, SETQUAL, SHOWQUAL.

REGIONMASK
Applies a mask to a region of an NDF

Description:

This routine masks out a region of an NDF by setting pixels to the bad value, or to a specified constant value. The region to be masked is specified by a file (see Parameter REGION) that should contain a description of the region in a form readable by the Starlink AST library (see SUN/211 or SUN/210). Such formats include AST's own native format and other formats that can be converted automatically to an AST Region (*e.g.* IVOA MOC and STC-S regions). AST Regions can be created, for instance, using the Starlink ATOOLS package (a high-level interface to the facilities of the AST library).

Usage:

```
regionmask in region out
```

Parameters:

CONST = LITERAL (Given)
The constant numerical value to assign to the region, or the string "Bad". ["Bad"]

IN = NDF (Read)
The name of the input NDF.

INSIDE = _LOGICAL (Read)
If a TRUE value is supplied, the constant value is assigned to the inside of the region. Otherwise, it is assigned to the outside. [TRUE]

OUT = NDF (Write)
The name of the output NDF.

REGION = FILENAME (Read)
The name of the file containing a description of the Region. This can be a text file holding a dump of an AST Region (any sub-class of Region may be supplied—*e.g.* Box, Polygon, CmpRegion, Prism, *etc.*), or any file that can be converted automatically to an AST Region (for instance an IVOA MOC in text or FITS format, an IVOA STC-S region in text format). An NDF may also be supplied, in which case the rectangular boundary of the NDF is used as the Region. If the axes spanned by the Region are not the same as those of the current WCS Frame in the input NDF, an attempt will be made to create an equivalent new Region that does match the current WCS Frame. An error will be reported if this is not possible.

Examples:

```
regionmask a1060 galaxies.txt a1060_sky
```

This copies input NDF a1060 to the output NDF a1060_sky, setting pixels bad if they are contained within the Region specified in text file "galaxies.txt".

Related Applications :

KAPPA: ARDMASK; ATOOLS: ASTBOX, ASTCMPREGION, ASTELLIPSE, ASTINTERVAL, ASTPOLYGON.

Implementation Status:

- This routine correctly processes the WCS, AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

REGRID
Applies a geometrical transformation to an NDF

Description:

This application uses a specified Mapping to re-grid the pixel positions in an NDF . The specified Mapping should transform pixel co-ordinates in the input NDF into the corresponding pixel co-ordinates in the output NDF.

By default, the bounds of the output pixel grid are chosen so that they just encompass all the transformed input data, but they can be set explicitly using Parameters LBOUND and UBOUND.

Two algorithms are available for determining the output pixel values: resampling and rebinning (the method used is determined by the REBIN parameter).

The Mapping to use can be supplied in several different ways (see Parameter MAPPING).

Usage:

```
regrid in out [method]
```

Parameters:

AXES() = _INTEGER (Read)

The indices of the pixel axes that are to be re-gridded. These should be in the range 1 to NDIM (the number of pixel axes in the NDF). Each value may appear at most once. The order of the supplied values is insignificant. If a null (`!`) value is supplied, then all pixel axes are re-gridded. Otherwise, only the specified pixel axes are regridded. Note, it is not always possible to specify completely arbitrary combinations of pixel axes to be regridded. For instance, if the current WCS Frame contains RA and Dec. axes, then it is not possible to regrid one of the corresponding pixel axes without the other. An error will be reported in such cases. `[!]`

CONSERVE = _LOGICAL (Read)

If set `TRUE`, then the output pixel values will be scaled in such a way as to preserve the total data value in a feature on the sky. The scaling factor is the ratio of the output pixel size to the input pixel size. This option can only be used if the Mapping is successfully approximated by one or more linear transformations. Thus an error will be reported if it used when the TOL parameter is set to zero (which stops the use of linear approximations), or if the Mapping is too non-linear to be approximated by a piece-wise linear transformation. The ratio of output to input pixel size is evaluated once for each panel of the piece-wise linear approximation to the Mapping, and is assumed to be constant for all output pixels in the panel. This parameter is ignored if the NORM parameter is set `FALSE`. `[TRUE]`

IN = NDF (Read)

The NDF to be transformed.

LBOUND( ) = _INTEGER (Read)

The lower pixel-index bounds of the output NDF. The number of values must be equal to the number of dimensions in the output NDF. If a null value is supplied,

default bounds will be used which are just low enough to fit in all the transformed pixels of the input NDF. [!]

MAPPING = FILENAME (Read)

The name of a file containing the Mapping to be used, or null (!) if the input NDF is to be mapped into its own current Frame. If a file is supplied, the forward direction of the Mapping should transform pixel co-ordinates in the input NDF into the corresponding pixel co-ordinates in the output NDF. If only a subset of pixel axes are being re-gridded, then the inputs to the Mapping should correspond to the pixel axes specified via Parameter AXES. The file may be one of the following.

- A text file containing a textual representation of the AST Mapping to use. Such files can be created by WCSADD.
- A text file containing a textual representation of an AST FrameSet. If the FrameSet contains a Frame with Domain PIXEL, then the Mapping used is the Mapping from the PIXEL Frame to the current Frame. If there is no PIXEL Frame in the FrameSet, then the Mapping used is the Mapping from the base Frame to the Current Frame.
- A FITS file. The Mapping used is the Mapping from the FITS pixel co-ordinates in which the centre of the bottom-left pixel is at co-ordinates (1,1), to the co-ordinate system represented by the primary WCS headers, CRVAL, CRPIX, *etc*.
- An NDF. The Mapping used is the Mapping from the PIXEL Frame to the Current Frame of its WCS FrameSet.

If a null (!) value is supplied, the Mapping used is the Mapping from pixel co-ordinates in the input NDF to the current Frame in the input NDF. The output NDF will then have pixel co-ordinates which match the co-ordinates of the current Frame of the input NDF (apart from possible additional scalings as specified by the SCALE parameter).

METHOD = LITERAL (Read)

The method to use when sampling the input pixel values (if resampling), or dividing an input pixel value between a group of neighbouring output pixels (if rebinning). For details of these schemes, see the descriptions of routines AST_RESAMPLEx and AST_REBINSEQx in SUN/210. METHOD can take the following values.

- "Bilinear" — When resampling, the output pixel values are calculated by bi-linear interpolation among the four nearest pixels values in the input NDF. When rebinning, the input pixel value is divided up bi-linearly between the four nearest output pixels. Produces smoother output NDFs than the nearest-neighbour scheme, but is marginally slower.
- "Nearest" — When resampling, the output pixel values are assigned the value of the single nearest input pixel. When rebinning, the input pixel value is assigned completely to the single nearest output pixel.
- "Sinc" — Uses the $\mathrm{sinc}(\pi x)$ kernel, where x is the pixel offset from the interpolation point (resampling) or transformed input pixel centre (rebinning), and $\mathrm{sinc}(z) = \sin(z)/z$. Use of this scheme is not recommended.
- "SincSinc" — Uses the $\mathrm{sinc}(\pi x)\mathrm{sinc}(k\pi x)$ kernel. A valuable general-purpose scheme, intermediate in its visual effect on NDFs between the bi-linear and

nearest-neighbour schemes.

- "SincCos" — Uses the $\mathrm{sinc}(\pi x)\cos(k\pi x)$ kernel. Gives similar results to the "Sincsinc" scheme.
- "SincGauss" — Uses the $\mathrm{sinc}(\pi x)e^{-kx^2}$ kernel. Good results can be obtained by matching the FWHM of the envelope function to the point-spread function of the input data (see Parameter PARAMS).
- "Somb" — Uses the $\mathrm{somb}(\pi x)$ kernel, where x is the pixel offset from the interpolation point (resampling) or transformed input pixel centre (rebinning), and $\mathrm{somb}(z) = 2 * J_1(z)/z$. J_1 is the first-order Bessel function of the first kind. This scheme is similar to the "Sinc" scheme.
- "SombCos" — Uses the $\mathrm{somb}(\pi x)\cos(k\pi x)$ kernel. This scheme is similar to the "SincCos" scheme.
- "Gauss" — Uses the e^{-kx^2} kernel. The FWHM of the Gaussian is given by Parameter PARAMS(2), and the point at which to truncate the Gaussian to zero is given by Parameter PARAMS(1).
- "BlockAve" — Block averaging over all pixels in the surrounding N-dimensional cube. This option is only available when resampling (*i.e.* if REBIN is set to FALSE).

All methods propagate variances from input to output, but the variance estimates produced by these schemes other than nearest neighbour need to be treated with care since the spatial smoothing produced by these methods introduces correlations in the variance estimates. Also, the degree of smoothing produced varies across the NDF. This is because a sample taken at a pixel centre will have no contributions from the neighbouring pixels, whereas a sample taken at the corner of a pixel will have equal contributions from all four neighbouring pixels, resulting in greater smoothing and lower noise. This effect can produce complex Moiré patterns in the output variance estimates, resulting from the interference of the spatial frequencies in the sample positions and in the pixel-centre positions. For these reasons, if you want to use the output variances, you are generally safer using nearest-neighbour interpolation. The initial default is "Nearest". [current value]

NORM = _LOGICAL (Read)

In general, each output pixel contains contributions from multiple input pixel values, and the number of input pixels contributing to each output pixel will vary from pixel to pixel. If NORM is set TRUE (the default), then each output value is normalised by dividing it by the number of contributing input pixels, resulting in each output value being the weighted mean of the contributing input values. However, if NORM is set FALSE, this normalisation is not applied. See also Parameter CONSERVE. [TRUE]

OUT = NDF (Write)

The transformed NDF.

PARAMS(2) = _DOUBLE (Read)

An optional array which consists of additional parameters required by the Sinc, SincSinc, SincCos, SincGauss, Somb, SombCos, and Gauss methods.

PARAMS(1) is required by all the above schemes. It is used to specify how many pixels are to contribute to the interpolated result on either side of the interpolation or binning point in each dimension. Typically, a value of 2 is appropriate and the minimum allowed value is 1 (*i.e.* one pixel on each side). A value of zero or fewer indicates that a suitable number of pixels should be calculated automatically. [0]

PARAMS(2) is required only by the Gauss, SincSinc, SincCos, and SincGauss schemes. For the SombCos, SincSinc, and SincCos schemes, it specifies the number of pixels at which the envelope of the function goes to zero. The minimum value is `1.0`, and the run-time default value is `2.0`. For the Gauss and SincGauss schemes, it specifies the full-width at half-maximum (FWHM) of the Gaussian envelope measured in output pixels. The minimum value is `0.1`, and the run-time default is `1.0`. On astronomical images and spectra, good results are often obtained by approximately matching the FWHM of the envelope function, given by PARAMS(2), to the point-spread function of the input data. `[]`

REBIN = _LOGICAL (Read)
Determines the algorithm used to calculate the output pixel values. If a `TRUE` value is given, a rebinning algorithm is used. Otherwise, a resampling algorithm is used. See the "Choice of Algorithm" topic below. [current value]

SCALE( ) = _DOUBLE (Read)
Axis scaling factors which are used to modify the supplied Mapping. If the number of supplied values is fewer than the number of output axes associated with the Mapping, the final supplied value is duplicated for the missing axes. In effect, transformed input co-ordinate axis values would be multiplied by these factors to obtain the corresponding output pixel co-ordinates. If a null (`!`) value is supplied for SCALE, then default values are used which depends on the value of Parameter MAPPING. If a null value is supplied for MAPPING then the default scaling factors are chosen so that pixels retain their original size (very roughly) after transformation. If as non-null value is supplied for MAPPING then the default scaling factor used is 1.0 for each axis (*i.e.* no scaling). `[!]`

TITLE = LITERAL (Read)
A Title for the output NDF structure. A null value (`!`) propagates the title from the input NDF to the output NDF. `[!]`

TOL = _DOUBLE (Read)
The maximum tolerable geometrical distortion which may be introduced as a result of approximating non-linear Mappings by a set of piece-wise linear transforms. The resampling algorithm approximates non-linear co-ordinate transformations in order to improve performance, and this parameter controls how inaccurate the resulting approximation is allowed to be, as a displacement in pixels of the input NDF. A value of zero will ensure that no such approximation is done, at the expense of increasing execution time. `[0.2]`

UBOUND( ) = _INTEGER (Read)
The upper pixel-index bounds of the output NDF. The number of values must be equal to the number of dimensions of the output NDF. If a null value is supplied, default bounds will be used which are just high enough to fit in all the transformed pixels of the input NDF. `[!]`

WLIM = _REAL (Read)
This parameter is only used if REBIN is set `TRUE`. It specifies the minimum number of good pixels which must contribute to an output pixel for the output pixel to be valid. Note, fractional values are allowed. A null (`!`) value causes a very small positive value to be used resulting in output pixels being set bad only if they receive no significant contribution from any input pixel. `[!]`

Examples:

```
regrid sg28948 sg28948r mapping=rotate.ast
```

Here sg28948 is resampled into a new co-ordinate system using the AST Mapping stored in a text file called `rotate.ast` (which may have been created using WCSADD for instance).

```
regrid flat distorted mapping=!
```

This transforms the NDF called flat into its current co-ordinate Frame, writing the result to an NDF called distorted. It uses nearest-neighbour resampling. If the units of the PIXEL and current co-ordinate Frames of flat are of similar size, then the pixel co-ordinates of distorted will be the same as the current co-ordinates of flat, but if there is a large scale discrepancy a scaling factor will be applied to give the output NDF a similar size to the input one. The output NDF will be just large enough to hold the transformed copies of all the pixels from NDF flat.

```
regrid flat distorted mapping=!  scale=1 method=sinccos params=[0,3]
```

As the previous example, but the additional scaling factor will not be applied even in the case of large size discrepancy, and a sinc∗cos one-dimensional resampling kernel is used which rolls off at a distance of 3 pixels from the central one.

```
regrid flat distorted mapping=!  scale=0.2 method=blockave params=2
```

In this case, an additional shrinking factor of 0.2 is being applied to the output NDF (*i.e.* performed following the Mapping from pixel to current co-ordinates), and the resampling is being done using a block averaging scheme in which a cube extending two pixels either side of the central pixel is averaged over to produce the output value. If the PIXEL-domain and current Frame pixels have (about) the same size, this will result in every pixel from the input NDF adding a contribution to one pixel of the output NDF.

```
regrid a119 a119s mapping=!  lbound=[1,-20] ubound=[256,172]
```

This transforms the NDF called a119 into an NDF called a119s. It uses nearest-neighbour resampling. The shape of a119s is forced to be (1:256,−20:172) regardless of the location of the transformed pixels of a119.

Notes:

- If the input NDF contains a VARIANCE component, a VARIANCE component will be written to the output NDF. It will be calculated on the assumption that errors on the input data values are statistically independent and that their variance estimates may simply be summed (with appropriate weighting factors) when several input pixels contribute to an output data value. If this assumption is not valid, then the output error estimates may be biased. In addition, note that the statistical errors on

neighbouring output data values (as well as the estimates of those errors) may often be correlated, even if the above assumption about the input data is correct, because of the sub-pixel interpolation schemes employed.

- This task is based on the AST_RESAMPLEx and AST_REBINSEQx routines described in SUN/210.

Choice of Algorithm :

The algorithm used to produce the output image is determined by the REBIN parameter, and is based either on resampling the output image or rebinning the corresponding input image.

The resampling algorithm steps through every pixel in the output image, sampling the input image at the corresponding position and storing the sampled input value in the output pixel. The method used for sampling the input image is determined by the METHOD parameter. The rebinning algorithm steps through every pixel in the input image, dividing the input pixel value between a group of neighbouring output pixels, incrementing these output pixel values by their allocated share of the input pixel value, and finally normalising each output value by the total number of contributing input values. The way in which the input sample is divided between the output pixels is determined by the METHOD parameter.

Both algorithms produce an output in which the each pixel value is the weighted mean of the nearby input values, and so do not alter the mean pixel values associated with a source, even if the pixel size changes. Thus the total data sum in a source will change if the input and output pixel sizes differ. However, if the CONSERVE parameter is set `TRUE`, the output values are scaled by the ratio of the output to input pixel size, so that the total data sum in a source is preserved.

A difference between resampling and rebinning is that resampling guarantees to fill the output image with good pixel values (assuming the input image is filled with good input pixel values), whereas holes can be left by the rebinning algorithm if the output image has smaller pixels than the input image. Such holes occur at output pixels that receive no contributions from any input pixels, and will be filled with the value zero in the output image. If this problem occurs, the solution is probably to change the width of the pixel spreading function by assigning a larger value to PARAMS(1) and/or PARAMS(2) (depending on the specific METHOD value being used).

Both algorithms have the capability to introduce artefacts into the output image. These have various causes described below.

- Particularly sharp features in the input can cause rings around the corresponding features in the output image. This can be minimised by suitable settings for the METHOD and PARAMS parameters. In general such rings can be minimised by using a wider interpolation kernel (if resampling) or spreading function (if rebinning), at the cost of degraded resolution.
- The approximation of the Mapping using a piece-wise linear transformation (controlled by Parameter TOL) can produce artefacts at the joints between the panels of the approximation. These can occur when using the rebinning algorithm, or when using the resampling algorithm with CONSERVE set to `TRUE`. They are caused by

the discontinuities between the adjacent panels of the approximation, and can be minimised by reducing the value assigned to the TOL parameter.

Related Applications :

KAPPA: FLIP, ROTATE, SLIDE, WCSADD, WCSALIGN; CCDPACK: TRANLIST, TRANNDF, WCSEDIT.

Implementation Status:

- The LABEL, UNITS, and HISTORY components, and all extensions are propagated. TITLE is controlled by the TITLE parameter. DATA, VARIANCE, and WCS are propagated after appropriate modification. The QUALITY component is also propagated if Nearest-Neighbour interpolation is being used (note, REBIN must be `FALSE`). The AXIS component is not propagated.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. If REBIN is `TRUE`, the data type will be converted to one of _INTEGER, _DOUBLE or _REAL for processing.
- There can be an arbitrary number of NDF dimensions.

REMQUAL
Removes specified quality definitions from an NDF

Description:

This routine removes selected quality name definitions from an NDF (see Task SETQUAL) and optionally clears the corresponding bit in the QUALITY array of the supplied NDF. All quality names information may be removed by specifying a quality name of `"ANY"`.

An error will be reported if an attempt is made to remove a quality name that has been flagged as "read-only" (*e.g.* using the READONLY parameter of the SETQUAL application).

Usage:

```
remqual ndf qnames
```

Parameters:

CLEAR = _LOGICAL (Read)
If `TRUE`, the bits in the NDF's QUALITY array that correspond to the removed quality names will be cleared. If `FALSE`, no change will be made to the QUALITY array. `[FALSE]`

NDF = NDF (Update)
The NDF to be modified.

QNAMES = LITERAL (Read)
A group of up to 10 quality names to be removed from the input NDF. The group may be supplied as a comma-separated list, or within a text file (in which case the name of the text file should be given, preceded by a `"^"` character.) If more than 10 names are supplied, only the first 10 are used. If any of the supplied quality names are not defined in the NDF, then warning messages are given but the application continues to remove any other specified quality names. If the string ANY is specified, then all defined quality names are removed. If no defined quality names remain, the structure used to store quality name information is deleted. This feature can be used to get rid of corrupted quality name information.

Related Applications :

KAPPA: QUALTOBAD, SHOWQUAL, SETQUAL.

Examples:

```
remqual "m51*" any
```

This example will remove all defined quality names from all NDFs with names starting with the string `"m51"`.

RESHAPE
Reshapes an NDF, treating its arrays as vectors

Description:

This application reshapes an NDF to create another NDF by copying array values. The array components in the input NDF are treated as vectors. Each output array is filled in order with values from the input vector, until it is full or the input vector is exhausted. Output data and variance pixels not filled are set to the bad value; unfilled quality pixels are set to zero. The filling is in Fortran order, namely the first dimension, followed by the second dimension,... to the highest dimension.

It is possible to form a vectorized NDF using Parameter VECTORIZE without having to specify the shape.

Usage:

```
reshape in out shape=?
```

Parameters:

IN = NDF (Read)
The input NDF to be reshaped.

OUT = NDF (Read)
The NDF after reshaping.

SHAPE( ) = _INTEGER (Read)
The shape of the output NDF. For example, `[50,30,20]` would create 50 columns by 30 lines by 20 bands. It is only accessed when VECTORIZE=`FALSE`.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (`!`) propagates the title from the base NDF to the output NDF. `[!]`

VECTORIZE = _LOGICAL (Read)
If `TRUE`, the output NDF is the vectorized form of the input NDF. If `FALSE`, Parameter SHAPE is used to specify the new shape. `[FALSE]`

Examples:

```
reshape shear normal shape=[511,512]
```

This reshapes the NDF called shear to form NDF normal, whose shape is 511×512 pixels. One example is where the original image has 512×512 pixels but one pixel was omitted from each line during some data capture, causing the image to be sheared between lines.

```
reshape cube cube1d vectorize
```

This vectorizes the NDF called cube to form NDF cube1d. This could be used for a task that only permits one-dimensional data.

Related Applications :

KAPPA: CHAIN, PASTE, RESHAPE.

Implementation Status:

- This routine correctly processes the DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, and HISTORY, components of an NDF data structure and propagates all extensions. WCS, and AXIS information is lost.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

RIFT
Adds a scalar to a section of an NDF data structure to correct rift-valley defects

Description:

The routine adds a scalar (*i.e.* constant) value to each pixel of an NDF's data array within a sub-section to produce a new NDF data structure.

Usage:

```
rift in scalar out section
```

Parameters:

IN = NDF (Read)
Input NDF data structure, to which the value is to be added.

OUT = NDF (Write)
Output NDF data structure.

SCALAR = _DOUBLE (Read)
The value to be added to the NDF's data array within the section.

SECTION = LITERAL (Read)
The pixels to which a scalar is to be added. This is defined as an NDF section, so that ranges can be defined along any axis, and be given as pixel indices or axis (data) co-ordinates. So for example `"3,4,5"` would select the pixel at (3,4,5); `"3:5,"` would select all elements in columns 3 to 5; `",4"` selects line 4. See Section 9 for details.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
rift aa 10.7 bb "100:105" 20
```

This adds 10 in the columns 100 to 105 in the data array of the NDF called aa and stores the result in the NDF called bb. In other respects bb is a copy of aa.

```
rift cubein -100 cubeout ",,4"
```

This adds −100 to all values in the fourth plane of the data array of the NDF called cubein and stores the result in the NDF called cubeout. In other respects cubeout is a copy of cubeout.

```
rift in=aa scalar=2 out=bb section="-10:5,200~9"
```

This adds 2 to the rectangular section between columns −10 to 5 and lines 196 to 204 of the data array of the NDF called aa and stores the result in the NDF called bb. In other respects bb is a copy of aa.

Notes:

For similar operations performed on a subset, use the appropriate application to process the relevant section and then run PASTE to paste the result back into the full array.

Related Applications :

KAPPA: CADD, CHPIX, GLITCH, PASTE, SEGMENT, ZAPLIN; FIGARO: CSET, ICSET, NCSET, TIPPEX.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- The bad-pixel flag is set to `TRUE` if undefined values are created during the arithmetic.
- All non-complex numeric data types can be handled.

ROTATE
Rotates a two-dimensional NDF about its centre through any angle

Description:

This routine rotates an array stored in an NDF data structure by an arbitrary angle. The rotation angle can be chosen automatically to make north vertical in the output NDF (see Parameter ANGLE). The origin of the rotation is around the point (0, 0) in pixel co-ordinates. The output array dimensions just accommodate the rotated array. Output pixels can be generated from the input array by one of two methods: nearest-neighbour substitution or by bi-linear interpolation. The latter is slower, but gives better results. Output pixels not corresponding to input pixels take the bad value.

The NDF may have two or three dimensions. If it has three dimensions, then the rotation is applied in turn to each plane in the cube and the result written to the corresponding plane in the output cube. The orientation of the rotation plane can be specified using the AXES parameter.

Usage:

```
rotate in out angle
```

Results Parameters:

ANGLEUSED( ) = _REAL (Write)

An output parameter holding the rotation angle actually used, in degrees. This is useful if a null value is supplied for parameter ANGLE.

Parameters:

ANGLE = _REAL (Read)

Number of clockwise degrees by which the data array is to be rotated. It must lie between -360 and 360 degrees. The suggested default is the current value. If a null (!) value is supplied, then the rotation angle is chosen to make north vertical at the centre of the image. If the current co-ordinate Frame in the input NDF is not a celestial co-ordinate frame, then the rotation angle is chosen to make the second axis of the current Frame vertical.

AXES(2) = _INTEGER (Read)

This parameter is only accessed if the NDF has exactly three significant pixel axes. It should be set to the indices of the NDF pixel axes which span the plane in which rotation is to be applied. All pixel planes parallel to the specified plane will be rotated independently of each other. The dynamic default comprises the indices of the first two significant axes in the NDF. Note that excluding the first significant axis may be very inefficient for large cubes; a prior reconfiguration with application PERMAXES that is compatible with the dynamic default for AXES, will often prove beneficial. []

IN = NDF (Read)

NDF structure containing the two- or three-dimensional array to be rotated.

NNMETH = _LOGICAL (Read)

If TRUE, the nearest-neighbour method will be used to evaluate the output data-array pixels. This is only accessed when the rotation is not a multiple of 90 degrees. [FALSE]

OUT = NDF (Write)
Output NDF to contain the rotated arrays.

QUALITY = _LOGICAL (Read)
This parameter is only accessed when NNMETH is FALSE and ANGLE is not a multiple of 90 degrees. Strictly, the quality values are undefined by the bi-linear interpolation and hence cannot be propagated. However, QUALITY=TRUE offers an approximation to the quality array by propagating the nearest-neighbour quality to the output NDF. [FALSE]

TITLE = LITERAL (Read)
A title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

USEAXIS = GROUP (Read)
USEAXIS is only accessed if the current co-ordinate Frame of the NDF has more than two axes. A group of two strings should be supplied specifying the two axes which are to be used when determining the rotation angle needed to make north vertical. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (!) value is supplied, the axes with the same indices as the two used pixel axes within the NDF are used. [!]

VARIANCE = _LOGICAL (Read)
A TRUE value causes variance values to be used as weights for the pixel values in bi-linear interpolation, and also causes output variances to be created. This parameter is ignored if ANGLE is a multiple of 90 degrees or NNMETH=TRUE; in these cases the variance array is merely propagated. If a null (!) value is supplied, the value used is TRUE if the input NDF has a VARIANCE component, and FALSE otherwise. Note that following this operation the errors are no longer independent. [!]

Examples:

```
rotate ns ew 90
```

This rotates the array components in the NDF called ns by 90 degrees clockwise around pixel co-ordinates [0, 0] and stores the result in the NDF called ew. The former x axis becomes the new y axis, and the former y axis becomes the new x axis. The former y-axis arrays are also reversed in the process.

```
rotate m31 m31r angle=!
```

This rotates the NDF called m31 so that north is vertical and stores the results in an NDF called m31r. This assumes that the current WCS Frame in the input NDF is a celestial co-ordinate Frame.

```
rotate angle=180 out=sn in=ns
```

This rotates the array components in the NDF called ns by 180 degrees clockwise around the pixel co-ordinates [0, 0], and stores the result in the NDF called sn. The axis arrays are flipped in the output NDF.

```
rotate f1 f1r 37.2 novariance
```

This rotates the array components in the NDF called f1 by 37.2 degrees clockwise around the pixel co-ordinates [0, 0], and stores the result in the NDF called f1r. The original axis information is lost. Bi-linear interpolation is used without variance information. No quality or variance information is propagated.

```
rotate f1 f1r 106 nnmeth title="Reoriented features map"
```

This rotates the array components in the NDF called f1 by 106 degrees clockwise around the pixel co-ordinates [0, 0], and stores the result in the NDF called f1r. The original axis information is lost. The resultant array components, all of which are propagated, are calculated by the nearest-neighbour method. The title of the output NDF is `"Reoriented features map"`.

```
rotate velmap rotvelmap 70
```

This rotates the array components in the three-dimensional NDF called velmap by 70 degrees clockwise around the pixel co-ordinates [0,0], and stores the result in the NDF called rotvelmap. The rotation is applied to the first two pixel axes repeated for all the planes in the cube's third pixel axis.

```
rotate velmap rotvelmap 70 axes=[1,3]
```

This as the previous example except that the rotation is applied in the plane given by the first and third pixel axes.

Notes:

- Bad pixels are ignored in the bi-linear interpolation. If all four pixels are bad, the result is bad.

Related Applications :

KAPPA: FLIP, REGRID; FIGARO: IREVX, IREVY, IROT90.

Implementation Status:

The propagation rules depend on Parameters ANGLE and NNMETH.

- For rotations that are multiples of 90-degrees, VARIANCE, QUALITY, AXIS, HISTORY, LABEL WCS, and UNITS components of the input NDF are propagated to the output NDF. The axis and WCS components are switched and flipped as appropriate.
- For the nearest-neighbour method VARIANCE, QUALITY, HISTORY, LABEL, WCS and UNITS components of the input NDF are propagated to the output NDF.
- For the linear-interpolation method HISTORY, LABEL, WCS and UNITS components of the input NDF are propagated to the output NDF. In addition if Parameter VARIANCE is `TRUE`, variance information is derived from the input variance; and if Parameter QUALITY is `TRUE`, QUALITY is propagated using the nearest neighbour.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric types are supported, though for linear interpolation the arithmetic is performed using single- or double-precision floating point as appropriate; and for 90 and 270-degree rotations _INTEGER is used for all integer types.

SCATTER
Displays a scatter plot between data in two NDFs

Description:

This application displays a two-dimensional plot in which the horizontal axis corresponds to the data value in the NDF given by Parameter IN1, and the vertical axis corresponds to the data value in the NDF given by Parameter IN2. Optionally, the variance, standard deviation or quality may be used instead of the data value for either axis (see Parameters COMP1 and COMP2). A symbol is displayed at an appropriate position in the plot for each pixel which has a good value in both NDFs, and falls within the bounds specified by Parameters XLEFT, XRIGHT, YBOT, and YTOP. The type of symbol may be specified using Parameter MARKER.

The supplied arrays may be compressed prior to display (see Parameter COMPRESS). This reduces the number of points in the scatter plot, and also reduces the noise in the data.

The Pearson correlation coefficient of the displayed scatter plot is also calculated and displayed, and written to output Parameter CORR.

A linear fit to the data can be calculated and displayed (see Parameter FIT).

Usage:

```
scatter in1 in2 [comp1] [comp2] [device]
```

Parameters:

AXES = _LOGICAL (Read)

`TRUE` if labelled and annotated axes are to be drawn around the plot. The width of the margins left for the annotation may be controlled using Parameter MARGIN. The appearance of the axes (colours, founts, *etc.*) can be controlled using the Parameter STYLE. The dynamic default is `TRUE` if CLEAR is `TRUE`, and `FALSE` otherwise. `[]`

CLEAR = _LOGICAL (Read)

If `TRUE` the current picture is cleared before the plot is drawn. If CLEAR is `FALSE` not only is the existing plot retained, but also an attempt is made to align the new picture with the existing picture. Thus you can generate a composite plot within a single set of axes, say using different colours or modes to distinguish data from different datasets. `[TRUE]`

COMP1 = LITERAL (Read)

The NDF array component to be displayed on the horizontal axis. It may be `"Data"`, `"Quality"`, `"Variance"`, or `"Error"` (where `"Error"` is an alternative to `"Variance"` and causes the square root of the variance values to be displayed). If `"Quality"` is specified, then the quality values are treated as numerical values (in the range 0 to 255). `["Data"]`

COMP2 = LITERAL (Read)

The NDF array component to be displayed on the vertical axis. It may be `"Data"`, `"Quality"`, `"Variance"`, or `"Error"` (where `"Error"` is an alternative to `"Variance"` and causes the square root of the variance values to be displayed). If `"Quality"` is

specified, then the quality values are treated as numerical values (in the range 0 to 255). ["Data"]

COMPRESS() = _INTEGER (Read)
The compression factors to be used when compressing the supplied arrays prior to display. If any of the supplied values are greater than 1, then the supplied arrays are compressed prior to display by replacing each box of input pixels by a single pixel equal to the mean of the pixels in the box. The size of each box in pixels is given by the compression factors. No compression occurs if all values supplied for this parameter are 1. If the number of values supplied is smaller than the number of axes, the final value supplied is duplicated for the remaining axes. [1]

DEVICE = DEVICE (Read)
The graphics workstation on which to produce the plot. If a null value (!) is supplied no plot will be made. [current graphics device]

FIT = _LOGICAL (Read)
If TRUE, then a linear fit to the scatter points is added to the plot. The slope and offset of this fit is displayed on the screen and written to output Parameters SLOPE, OFFSET, and RMS. A symmetric linear-fit algorithm is used, which caters for the presence of noise in both X and Y values. Outliers are identified and ignored. Note, the fit is based on just those points that are visible in the scatter plot. Points outside the bounds of the plot are ignored. Points that are inside the plot are also ignored if their reflection through the best-fit line are outside the plot. This avoids biasing the fit if the plot bounds omit more points on one side of the line than the other. [current value]

IN1 = NDF (Read)
The NDF to be displayed on the horizontal axis.

IN2 = NDF (Read)
The NDF to be displayed on the vertical axis.

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. If a null (!) value is supplied, the value used is 0.15 (for all edges) if annotated axes are produced, and zero otherwise. [current value]

MARKER = _INTEGER (Read)
Specifies the symbol with which each position should be marked in the plot. It should be given as an integer PGPLOT marker type. For instance, 0 gives a box, 1 gives a dot, 2 gives a cross, 3 gives an asterisk, 7 gives a triangle. The value must be larger than or equal to -31. [current value]

PERC1(2) = _REAL (Read)
The percentiles that define the default values for XLEFT and XRIGHT. For example, [5,95] would result in the lowest and highest 5% of the data value in IN1 being excluded from the plot if the default values are accepted for XLEFT and XRIGHT. [current value]

PERC2(2) = _REAL (Read)
The percentiles that define the default values for YBOT and YTOP. For example,

[5,95] would result in the lowest and highest 5% of the data value in IN2 being excluded from the plot if the default values are accepted for YBOT and YTOP. [current value]

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use when drawing the annotated axes, and markers.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of markers is controlled by Colour(Markers), Width(Markers), *etc.* (the synonym Symbols may be used in place of Markers). [current value]

XLEFT = _DOUBLE (Read)

The axis value to place at the left hand end of the horizontal axis. If a null (!) value is suplied, the value used is determined by Parameter PERC1. 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. If a null (!) value is suplied, the value used is determined by Parameter PERC1. 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. If a null (!) value is suplied, the value used is determined by Parameter PERC2. 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. If a null (!) value is suplied, the value used is determined by Parameter PERC2. The value supplied may be greater than or less than the value supplied for YBOT. [!]

Results Parameters:

CORR = _DOUBLE (Write)

The Pearson correlation coefficient of the visible points in the scatter plot (points outside

the plot are ignored). A value of zero is stored if the correlation coefficient cannot be calculated.

NPIX = _INTEGER (Write)
The number of pixels used to form the correlation coefficient.

OFFSET = _DOUBLE (Write)
An output parameter giving the offset in the linear fit: IN2 = SLOPE $*$ IN1 + OFFSET. Only used if Parameter FIT is `TRUE`.

RMS = _DOUBLE (Write)
An output parameter giving the RMS residual of the data (excluding outliers) about the linear fit. Only used if Parameter FIT is `TRUE`.

SLOPE = _DOUBLE (Write)
An output parameter giving the slope of the linear fit: IN2 = SLOPE $*$ IN1 + OFFSET. Only used if Parameter FIT is `TRUE`.

Examples:

```
scatter cl123a cl123b
```

This displays a scatter plot of the data value in NDF cl123b against the data value in NDF cl123a, on the current graphics device.

```
scatter cl123a cl123a pscol_l comp2=error compress=3
```

This displays a scatter plot of the error in NDF cl123a against the data value in the same NDF. The graphics device used is pscol_l. The data are compressed by a factor of 3 on each axis before forming the plot.

Notes:

- Any pixels that are bad (after any compression) in either array are excluded from the plot, and from the calculation of the default axis limits
- The application stores two pictures in the graphics database in the following order: a FRAME picture containing the annotated axes and data plot, and a DATA picture containing just the data plot. Note, the FRAME picture is only created if annotated axes have been drawn, or if non-zero margins were specified using Parameter MARGIN. The world co-ordinates in the DATA picture will correspond to data value in the two NDFs.

Related Applications :

KAPPA: NORMALIZE.

Implementation Status:

- Processing of bad pixels and automatic quality masking are supported.
- Only _REAL data can be processed directly. Other non-complex numeric data types will undergo a type conversion before processing occurs.

SEGMENT
Copies polygonal segments from one NDF into another

Description:

This routine copies one or more polygonal segments from the first input NDF (Parameter IN1), and pastes them into the second input NDF (Parameter IN2) at the same pixel co-ordinates. The resulting mosaic is stored in the output NDF (see OUT). Either input NDF may be supplied as null (`!`) in which case the corresponding areas of the output NDF are filled with bad values. For instance, supplying a null value for IN2 allows segments to be cut from IN1 and pasted on to a background of bad values. Supplying a null value for IN1 allows 'holes' to be cut out of IN2 and filled with bad values.

Each polygonal segment is specified by giving the positions of its vertices. This may be done using a graphics cursor, by supplying a positions list or text file containing the positions, or by supplying the positions in response to a parameter prompt. The choice is made by Parameter MODE.

This application may also be used to cut and paste cylinders with polygonal cross-sections from NDFs with more than two dimensions. See the "Notes" section below for further details.

Usage:

```
                        { coords=?
segment in1 in2 out     { incat1-incat20=?
                        { poly1-poly20=?
                      mode
```

Parameters:

COORDS = LITERAL (Read)

The co-ordinates of a single vertex for the current polygon. If Parameter MODE is set to `"Interface"`, this parameter is accessed repeatedly to obtain the co-ordinates of all vertices in the polygon. A null value should be given when the final vertex has been specified. Each position should be supplied within the current co-ordinate Frame of the output NDF (see Parameter OUT). Supplying a colon `":"` will display details of the required co-ordinate Frame. No more than two formatted axis values (separated by a comma or space) may be supplied. If the co-ordinate Frame being used has more than two axes, then the two axes to use must be specified using Parameter USEAXIS.

DEVICE = DEVICE (Read)

The name of the graphics device on which an image is displayed. Only used if Parameter MODE is given the value `"Cursor"`. Any graphics specified by Parameter PLOT will be produced on this device. This device must support cursor interaction. [current graphics device]

IN1 = NDF (Read)

The input NDF containing the data to be copied to the inside of the supplied polygonal

segments. If a null value is supplied, the inside of the polygonal segments will be filled with bad values.

IN2 = NDF (Read)
The input NDF containing the data to be copied to the outside of the supplied polygonal segments. If a null value is supplied, the outside of the polygonal segments will be filled with bad values.

INCAT1-INCAT20 = FILENAME (Read)
If MODE is `"Catalogue"`, each of the Parameters INCAT1 to INCAT20 are used to access catalogues containing the co-ordinates of the vertices of a single polygon. Suitable catalogues may be created using CURSOR, LISTMAKE, *etc*. If a value is assigned to INCAT1 on the command line, you are not prompted for any of the remaining parameters in this group; additional polygon catalogues must also be supplied on the command line. Otherwise, you are prompted for INCAT1, then INCAT2, *etc.* until a null value is given or INCAT20 is reached.

The positions in each catalogue are mapped into the pixel co-ordinate Frame of the output NDF by aligning the WCS information stored in the catalogue with the WCS information in the output NDF. A message indicating the Frame in which the positions were aligned with the output NDF is displayed.

LOGFILE = FILENAME (Write)
The name of a text file in which the co-ordinates of the polygon vertices are to be stored. A null value (`!`) means that no file is created. `[!]`

MARKER = _INTEGER (Read)
This parameter is only accessed if Parameter PLOT is set to `"Chain"` or `"Mark"`. It specifies the type of marker with which each cursor position should be marked, and should be given as an integer PGPLOT marker type. For instance, `0` gives a box, `1` gives a dot, `2` gives a cross, `3` gives an asterisk, `7` gives a triangle. The value must be larger than or equal to -31. [current value]

MODE = LITERAL (Read)
The mode in which the co-ordinates of each polygon vertex are to be obtained. The supplied string can be one of the following selection.

- `"Interface"` — positions are obtained using Parameter COORDS. These positions must be supplied in the current co-ordinate Frame of the output NDF (see Parameter OUT).
- `"Cursor"` — positions are obtained using the graphics cursor of the device specified by Parameter DEVICE. The WCS information stored with the picture in the graphics database is used to map the supplied cursor positions into the pixel co-ordinate Frame of the output NDF. A message is displayed indicating the co-ordinate Frame in which the picture and the output NDF were aligned.
- `"Catalogue"` — positions are obtained from positions lists using Parameters INCAT1 to INCAT20. Each catalogue defines a single polygon. The WCS information in each catalogue is used to map the positions in the catalogue into the pixel co-ordinate Frame of the output NDF. A message is displayed for each catalogue indicating the co-ordinate Frame in which the catalogue and the output NDF were aligned.

- "File" — positions are obtained from text files using Parameters POLY1 to POLY20. Each file defines a single polygon. Each line in a file must contain two formatted axis values in the current co-ordinate Frame of the output NDF (see Parameter OUT), separated by white space or a comma.

[current value]

MAXPOLY = _INTEGER (Read)
The maximum number of polygons which can be used. For instance, this can be set to 1 to ensure that no more than one polygon is used (this sort of thing can be useful when writing procedures or scripts). A null value causes no limit to be imposed (unless MODE="File" or "Catalogue" in which case a limit of 20 is imposed). [!]

MINPOLY = _INTEGER (Read)
The minimum number of polygons which can be used. For instance, this can be set to 2 to ensure that at least two polygons are used. The supplied value must be fewer than or equal to the value given for MAXPOLY and must be greater than zero. [1]

OUT = NDF (Write)
The output NDF. If only one input NDF is supplied (that is, if one of IN1 and IN2 is assigned a null value), then the output NDF has the same shape and size as the supplied input NDF. Also, ancillary data such as WCS information is propagated from the supplied input NDF. In particular, this means that the current co-ordinate Frame of the output NDF (in which vertex positions should be supplied if MODE is "File" or "Interface") is inherited from the input NDF. If two input NDFs are supplied, then the shape and size of the output NDF corresponds to the area of overlap between the two input NDFs (in pixel space), and the WCS information and current Frame are inherited from the NDF associated with Parameter IN1.

PLOT = LITERAL (Read)
The type of graphics to be used to mark the position of each selected vertex. It is only used if Parameter MODE is given the value "Cursor". The appearance of these graphics (colour, size, *etc.*) is controlled by the STYLE parameter. PLOT can take any of the following values.

- "None" — No graphics are produced.
- "Mark" — Each position is marked with a marker of type specified by Parameter MARKER.
- "Poly" — Causes each position to be joined by a line to the previous position. Each polygon is closed by joining the last position to the first.
- "Chain" — This is a combination of "Mark" and "Poly". Each position is marked by a marker and joined by a line to the previous position. Parameter MARKER is used to specify the marker to use. [current value]

POLY1-POLY20 = FILENAME (Read)
If MODE is "File", each of the Parameters POLY1 to POLY20 are used to access text files containing the co-ordinates of the vertices of a single polygon. If a value is assigned to POLY1 on the command line, you are not prompted for any of the remaining parameters in this group; additional polygon files must also be supplied on the command line. Otherwise, you are prompted for POLY1, then POLY2, *etc.* until a null value is given or POLY20 is reached.

Each position should be supplied within the current co-ordinate Frame of the output NDF (see Parameter OUT). No more than two formatted axis values (separated by a comma or space) may be supplied on each line. If the co-ordinate Frame being used has more than two axes, then the two axes to use must be specified using Parameter USEAXIS.

QUALITY = _LOGICAL (Read)

If a TRUE value is supplied for Parameter QUALITY then quality information is copied from the input NDFs to the output NDFs. Otherwise, the quality information is not copied. This parameter is only accessed if all supplied input NDFs have defined QUALITY components. If any of the supplied input NDFs do not have defined QUALITY components, then no quality is copied. Note, if a null input NDF is given then the corresponding output QUALITY values are set to zero. [TRUE]

STYLE = GROUP (Read)

A group of attribute settings describing the style to use when drawing the graphics specified by Parameter PLOT.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the lines forming the edges of each polygon is controlled by the attributes Colour(Curves), Width(Curves), *etc.* (either of the synonyms Lines and Edges may be used in place of Curves). The appearance of the vertex markers is controlled by the attributes Colour(Markers), Size(Markers), *etc.* (the synonyms Vertices may be used in place of Markers). [current value]

USEAXIS = GROUP (Read)

USEAXIS is only accessed if the current co-ordinate Frame of the output NDF has more than two axes. A group of two strings should be supplied specifying the two axes spanning the plane in which the supplied polygons are defined. Each axis can be specified using one of the following options.

- An integer index of an axis within the current Frame of the output NDF (in the range 1 to the number of axes in the current Frame).
- An axis Symbol string such as "RA" or "VRAD".

- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (!) value is supplied, the axes with the same indices as the first two significant NDF pixel axes are used. [!]

VARIANCE = _LOGICAL (Read)

If a TRUE value is supplied for Parameter VARIANCE then variance information is copied from the input NDFs to the output NDFs. Otherwise, the variance information is not copied. This parameter is only accessed if all supplied input NDFs have defined VARIANCE components. If any of the supplied input NDFs do not have defined VARIANCE components, then no variances are copied. Note, if a null input NDF is given then the corresponding output variance values are set bad. [TRUE]

Examples:

```
segment in1=m51a in2=m51b out=m51_comp incat1=coords mode=cat
```

Copies a region of the NDF m51a to the corresponding position in the output NDF m51_comp. The region is defined by the list of vertex co-ordinates held in catalogue coords.FIT. All pixels in the output NDF which fall outside this region are given the corresponding pixel values from NDF m51b.

```
segment in1=m51a out=m51_cut mode=cursor plot=poly accept
```

Copies a region of the NDF m51a to the corresponding position in the output NDF m51_cut. The region is defined by selecting vertices using a graphics cursor. The image m51a should previously have been displayed. Each vertex is joined to the previous vertex by a line on the graphics device. The ACCEPT keyword causes the suggested null default value for IN2 to be accepted. This means that all pixels outside the region identified using the cursor will be set bad in the output NDF.

Notes:

- Supplied positions are mapped into the pixel co-ordinate Frame of the output NDF before being used. This means that the two input NDFs (if supplied) must be aligned in pixel space before using this application.
- The routine can handle NDFs of arbitrary dimensionality. If either input has three or more dimensions then all planes in the NDF pixel arrays are processed in the same way, that is the same polygonal regions are extracted from each plane and copied to the corresponding plane of the output NDF. The plane containing the polygons must be defined using Parameter USEAXIS. This plane is a plane within the current co-ordinate Frame of the output NDF (which is inherited from the first supplied input NDF). This scheme will only work correctly if the selected plane in the current co-ordinate Frame is parallel to one of the planes of the pixel array.

Related Applications :

KAPPA: ARDMASK, ERRCLIP, FILLBAD, FFCLEAN, PASTE, REGIONMASK, SETMAGIC, THRESH.

Implementation Status:

- This routine will propagate VARIANCE component values so long as all supplied input NDFs have defined VARIANCE components, and Parameter VARIANCE is not `FALSE`.
- This routine will propagate QUALITY component values so long as all supplied input NDFs have defined QUALITY components, and Parameter QUALITY is not `FALSE`.
- The UNITS, AXIS, LABEL, TITLE, WCS, and HISTORY components are propagated from the first supplied input NDF, together with all extensions.
- All non-complex numeric types are supported. The following data types are processed directly: _WORD, _INTEGER, _REAL, _DOUBLE.

SETAXIS
Sets values for an axis array component within an NDF data structure

Description:

This routine modifies the values of an axis array component or system within an NDF data structure. There are a number of options (see Parameter MODE). They permit the deletion of the axis system, or an individual variance or width component; the replacement of one or more individual values; assignment of the whole array using Fortran-like mathematical expressions, or values in a text file, or to pixel co-ordinates, or by copying from another NDF.

If an AXIS structure does not exist, a new one whose centres are pixel co-ordinates is created before any modification.

Usage:

```
                                  { file=?
                                  {
                                  { index=?  newval=?
setaxis ndf dim mode [comp]       {
                                  { exprs=?
                                  {
                                  { axisndf=?
                                 mode
```

Parameters:

AXISNDF = NDF (Read)

The Data values in this NDF are used as the axis centre values if Parameter MODE is set to "NDF". The supplied NDF must be one dimensional and must be aligned in pixel co-ordinates with the NDF axis that is being modified.

COMP = LITERAL (Read)

The name of the NDF axis array component to be modified. The choices are: "Centre", "Data", "Error", "Width" or "Variance". "Data" and "Centre" are synonyms and selects the axis centres. "Variance" is the variance of the axis centres, *i.e.* measures the uncertainty of the axis-centre values. "Error" is the alternative to "Variance" and causes the square of the supplied error values to be stored. "Width" selects the axis width array. ["Data"]

DIM = _INTEGER (Read)

The axis dimension for which the array component is to be modified. There are separate arrays for each NDF dimension. The value must lie between 1 and the number of dimensions of the NDF. This defaults to 1 for a one-dimensional NDF. DIM is not accessed when COMP="Centre" and MODE="Delete". The suggested default is the current value. []

EXPRS = LITERAL (Read)

A Fortran-like arithmetic expression giving the value to be assigned to each element of the axis array specified by Parameter COMP. The expression may just contain a constant for the axis widths or variances, but the axis-centre values must vary.

In the latter case and whenever a constant value is not required, there are two tokens available—INDEX and CENTRE—either or both of which may appear in the expression. INDEX represents the pixel index of the corresponding array element, and CENTRE represents the existing axis centres. Either the CENTRE or the INDEX token must appear in the expression when modifying the axis centres. All of the standard Fortran-77 intrinsic functions are available for use in the expression, plus a few others (see SUN/61 for details and an up-to-date list).

Here are some examples. Suppose the axis centres are being changed, then EXPRS="INDEX-0.5" gives pixel co-ordinates, EXPRS="2.3 * INDEX + 10" would give a linear axis at offset 10 and an increment of 2.3 per pixel, EXPRS="LOG(INDEX*5.2)" would give a logarithmic axis, and EXPRS="CENTRE+10" would add ten to all the array centres. If COMP="Width", EXPRS=0.96 would set all the widths to 0.96, and EXPRS="SIND(INDEX-30)+2" would assign the widths to two plus the sine of the pixel index with respect to index 30 measured in degrees.

EXPRS is only accessed when MODE="Expression".

FILE = FILENAME (Read)

Name of the text file containing the free-format axis data. This parameter is only accessed if MODE="File". The suggested default is the current value.

INDEX = _INTEGER (Read)

The pixel index of the array element to change. A null value (!) terminates the loop during multiple replacements. This parameter is only accessed when MODE="Edit". The suggested default is the current value.

LIKE = NDF (Read)

A template NDF containing axis arrays. These arrays will be copied into the NDF given by Parameter NDF. All axes are copied. The other parameters are only accessed if a null (!) value is supplied for LIKE. If the NDF being modified extends beyond the edges of the template NDF, then the template axis arrays will be extrapolated to cover the entire NDF. This is done using linear extrapolation through the last two extreme axis values. [!]

MODE = LITERAL (Read)

The mode of the modification. It can be one of the following.

- "Delete" — Deletes the array, unless COMP="Data" or "Centre" whereupon the whole axis structure is deleted.
- "Edit" — Allows the modification of individual elements within the array.
- "Expression" — Allows a mathematical expression to define the array values. See Parameter EXPRS.
- "File" — The array values are read in from a free-format text file.
- "Linear_WCS" — The axis centres are set to the least-squares linear fit to the values of the selected axis in the current co-ordinate Frame of the NDF. This is useful for exporting to packages with limited FITS WCS compatibility and when the non-linearity is small. "Linear_WCS" is only available when COMP="Data" or "Centre".
- "NDF" — The axis centres are set to the corresponding Data values read from the NDF specified by Parameter AXISNDF. This is only available when COMP="Data" or "Centre".

- "Pixel" — The axis centres are set to pixel co-ordinates. This is only available when COMP="Data" or "Centre".
- "WCS" — The axis centres are set to the values of the selected axis in the current co-ordinate Frame of the NDF. This is only available when COMP="Data" or "Centre".

The suggested default is the current value.

NDF = NDF (Read and Write)
The NDF data structure in which an axis array component is to be modified.

NEWVAL = LITERAL (Read)
Value to substitute in the array element. The range of allowed values depends on the data type of the array being modified. NEWVAL="Bad" instructs that the bad value appropriate for the array data type be substituted. Placing NEWVAL on the command line permits only one element to be replaced. If there are multiple replacements, a null value (!) terminates the loop. This parameter is only accessed when MODE="Edit".

TYPE = LITERAL (Read)
The data type of the modified axis array. TYPE can be either "_REAL" or "_DOUBLE". It is only accessed for MODE="File", "Expression", or "Pixel". If a null (!) value is supplied, the value used is the current data type of the array component if it exists, otherwise it is "_REAL". [!]

Examples:

```
setaxis ff mode=delete
```

This erases the axis structure from the NDF called ff.

```
setaxis ff like=hh
```

This creates axis structures in the NDF called ff by copying them from the NDF called hh, extrapolating them as necessary to cover ff.

```
setaxis abell4 1 expr exprs="CENTRE + 0.1 * (INDEX-1)"
```

This modifies the axis centres along the first axis in the NDF called abell4. The new centre values are spaced by 0.1 more per element than previously.

```
setaxis cube 3 expr error exprs="25.3+0.2*MOD(INDEX,8)"
```

This modifies the axis errors along the third axis in the NDF called cube. The new errors values are given by the expression "25.3+0.2*MOD(INDEX,8)", in other words the noise has a constant term (25.3), and a cyclic ramp component of frequency 8 pixels.

```
setaxis spectrum mode=file file=spaxis.dat
```

This assigns the axis centres along the first axis in the one-dimensional NDF called spectrum. The new centre values are read from the free-format text file called spaxis.dat.

```
setaxis ndf=plate3 dim=2 mode=pixel
```

This assigns pixel co-ordinates to the second axis's centres in the NDF called plate3.

```
setaxis datafile 2 expression exprs="centre" type=_real
```

This modifies the data type of axis centres along the second dimension of the NDF called datafile to be _REAL.

```
setaxis cube 2 edit index=3 newval=129.916
```

This assigns the value 129.916 to the axis centre at index 3 along the second axis of the NDF called cube.

```
setaxis comp=width ndf=cube dim=1 mode=edit index=-16 newval=1E-05
```

This assigns the value 1.0E-05 to the axis width at index −16 along the first axis of the NDF called cube.

Notes:

- An end-of-file error results when MODE="File" and the file does not contain sufficient values to assign to the whole array. In this case the axis array is unchanged. A warning is given if there are more values in a file record than are needed to complete the axis array.
- An invalid expression when MODE="Expression" results in an error and the axis array is unchanged.
- The chapter entitled "The Axis Coordinate System" SUN/33 describes the NDF axis co-ordinates system and is recommended reading especially if you are using axis widths.
- There is no check, apart from constraints on Parameter NEWVAL, that the variance is not negative and the widths are positive.

File Format :

The format is quite flexible. The number of axis-array values that may appear on a line is variable; the values are separated by at least a space, comma, tab or carriage return. A line can have up to 255 characters. In addition a record may have trailing comments designated by a hash or exclamation mark. Here is an example file, though a more regular format would be clearer for the human reader (say 10 values per line with commenting).

```
# Axis Centres along second dimension
-3.4 -0.81
.1 3.3 4.52 5.6 9 10.5 12.  15.3 18.1 20.2
```

```
23 25.3 !  a comment
26.8,27.5 29.  30.76 32.1 32.4567
35.2 37.
<EOF>
```

Related Applications :

KAPPA: AXCONV, AXLABEL, AXUNITS; FIGARO: LXSET, LYSET.

Implementation Status:

Processing is in single- or double-precision floating point.

SETBAD
Sets new bad-pixel flag values for an NDF

Description:

This application sets new logical values for the bad-pixel flags associated with an NDF's data and/or variance arrays. It may either be used to test whether bad pixels are actually present in these arrays and to set their bad-pixel flags accordingly, or to set explicit `TRUE` or `FALSE` values for these flags.

Usage:

```
setbad ndf [value]
```

Parameters:

DATA = _LOGICAL (Read)

This parameter controls whether the NDF's data array is processed. If a `TRUE` value is supplied (the default), then it will be processed. Otherwise it will not be processed, so that the variance array (if present) may be considered on its own. The DATA and VARIANCE parameters should not both be set to `FALSE`. `[TRUE]`

MODIFY = _LOGICAL (Read)

If a `TRUE` value is supplied for this parameter (the default), then the NDF's bad-pixel flags will be permanently modified if necessary. If a `FALSE` value is supplied, then no modifications will be made. This latter mode allows the routine to be used to check for the presence of bad pixels without changing the current state of an NDF's bad-pixel flags. It also allows the routine to be used on NDFs for which write access is not available. `[TRUE]`

NDF = NDF (Read and Write)

The NDF in which bad pixels are to be checked for, and/or whose bad-pixel flags are to be modified. (Note that setting the MODIFY parameter to `FALSE` makes it possible to check for bad pixels without permanently modifying the NDF.)

VALUE = _LOGICAL (Read)

If a null (`!`) value is supplied for this parameter (the default), then the routine will check to see whether any bad pixels are present. This will only involve testing the value of each pixel if the bad-pixel flag value is initially `TRUE`, in which case it will be reset to `FALSE` if no bad pixels are found. If the bad-pixel flag is initially `FALSE`, then it will remain unchanged.

If a logical (`TRUE` or `FALSE`) value is supplied for this parameter, then it indicates the new bad-pixel flag value which is to be set. Setting a `TRUE` value indicates to later applications that there may be bad pixels present in the NDF, for which checks must be made. Conversely, setting a `FALSE` value indicates that there are definitely no bad pixels present, in which case later applications need not check for them and should interpret the pixel values in the NDF literally.

The VALUE parameter is not used (a null value is assumed) if the MODIFY parameter is set to `FALSE` indicating that the NDF is not to be permanently modified. `[!]`

VARIANCE = _LOGICAL (Read)
This parameter controls whether the NDF's variance array is processed. If a TRUE value is supplied (the default), then it will be processed. Otherwise it will not be processed, so that the data array may be considered on its own. The DATA and VARIANCE parameters should not both be set to FALSE. [TRUE]

Examples:

```
setbad ngc1097
```

Checks the data and variance arrays (if present) in the NDF called ngc1097 for the presence of bad pixels. If the initial bad-pixel flag values indicate that bad pixels may be present, but none are found, then the bad-pixel flags will be reset to FALSE. The action taken will be reported.

```
setbad ndf=ngc1368 nomodify
```

Performs the same checks as described above, this time on the NDF called ngc1368. The presence or absence of bad pixels is reported, but the NDF is not modified.

```
setbad myfile nodata
```

Checks the variance array (if present) in the NDF called myfile for the presence of bad pixels, and modifies its bad-pixel flag accordingly. Specifying nodata inhibits processing of the data array, whose bad-pixel flag is left unchanged.

```
setbad halpha false
```

Sets the bad-pixel flag for the NDF called halpha to FALSE. Any pixel values which might previously have been regarded as bad will subsequently be interpreted literally as valid pixels.

```
setbad hbeta true
```

Sets the bad-pixel flags for the NDF called hbeta to be TRUE. If any pixels have the special 'bad' value, then they will subsequently be regarded as invalid pixels. Note that if this is followed by a further command such as "setbad hbeta", then an actual check will be made to see whether any pixels have this special value. The bad-pixel flags will be returned to FALSE if they do not.

Bad-pixel Flag Values :

If a bad-pixel flag is TRUE, it indicates that the associated NDF array may contain the special 'bad' value and that affected pixels are to be regarded as invalid. Subsequent applications will need to check for such pixels and, if found, take account of them.

Conversely, if a bad-pixel flag value is FALSE, it indicates that there are no bad pixels present. In this case, any special 'bad' values appearing in the array are to be interpreted literally as valid pixel values.

Quality Components :

Bad pixels may also be introduced into an NDF's data and variance arrays implicitly through the presence of an associated NDF QUALITY component. This application will not take account of such a component, nor will it modify it.

However, if either of the NDF's data or variance arrays do not contain any bad pixels themselves, a check will be made to see whether a QUALITY component is present. If it is (and its associated bad-bits mask is non-zero), then a warning message will be issued indicating that bad pixels may be introduced via this QUALITY component. If required, these bad pixels may be eliminated either by setting the bad-bits mask to zero or by erasing the QUALITY component.

Related Applications :

KAPPA: NOMAGIC, SETMAGIC.

SETBB
Sets a new value for the quality bad-bits mask of an NDF

Description:

This application sets a new value for the bad-bits mask associated with the QUALITY component of an NDF . This 8-bit mask is used to select which of the bits in the quality array should normally be used to generate 'bad' pixels when the NDF is accessed.

Wherever a bit is set to 1 in the bad-bits mask, the corresponding bit will be extracted from the NDF's quality array value for each pixel (the other quality bits being ignored). A pixel is then considered 'bad' if any of the extracted quality bits is set to 1. Effectively, the bad-bits mask therefore allows selective activation of any of the eight 1-bit masks which can be stored in the quality array.

The bit mask can be given either numerically (in decimal, binary, octal or hexadecimal format), or as a set of quality names (see SETQUAL).

Usage:

```
setbb ndf bb
```

Parameters:

AND = _LOGICAL (Read)

By default, the value supplied via the BB parameter will be used literally as the new bad-bits mask value. However, if a `TRUE` value is given for the AND parameter, then a bit-wise 'AND' will first be performed with the old value of the mask. This facility allows individual bits in within the mask to be cleared (*i.e.* reset to zero) without affecting the current state of other bits (see the `"Examples"` section).

The AND parameter is not used if a `TRUE` value is given for the OR parameter. `[FALSE]`

BB = LITERAL (Read)

The new integer value for the bad-bits mask. This may either be specified in normal decimal notation, or may be given using binary, octal or hexadecimal notation by adding a `"B"`, `"O"` or `"Z"` prefix (respectively) to the appropriate string of digits. The value supplied should lie in the range 0 to 255 decimal (or 8 bits of binary).

If the AND and OR parameters are both `FALSE`, then the value supplied will be used directly as the new mask value. However, if either of these logical parameters is set to `TRUE`, then an appropriate bit-wise 'AND' or 'OR' operation with the old mask value will first be performed.

It may also be specified as a comma-separated list of quality names. A quality name is a symbolic name that identifies a specific quality bit (quality names can be defined using SETQUAL, and displayed using SHOWQUAL).

The default value suggested when prompting for this value is chosen so as to leave the original mask value unchanged.

NDF = NDF (Read and Write)

The NDF whose bad-bits mask is to be modified.

OR = _LOGICAL (Read)
By default, the value supplied via the BB parameter will be used literally as the new bad-bits mask value. However, if a TRUE value is given for the OR parameter, then a bit-wise 'OR' will first be performed with the old value of the mask. This facility allows individual bits in within the mask to be set to 1 without affecting the current state of other bits (see the "Examples" section). [FALSE]

Examples:

```
setbb myframe 3
```

Sets the bad-bits mask value for the QUALITY component of the NDF called myframe to the value 3. This means that bits 1 and 2 of the associated quality array will be used to generate bad pixels.

```
setbb myframe "SKY,BACK"
```

Sets the bad-bits mask value for the quality component of the NDF called myframe so that any pixel that is flagged with either of the two qualities "SKY" or "BACK" will be set bad. The NDF should contain information that associates each of these quality names with a specific bit in the quality array. Such information can for instance be created using the SETQUAL command.

```
setbb ndf=myframe bb=b11
```

This example performs the same operation as above, but in this case the new mask value has been specified using binary notation.

```
setbb xspec b10001000 or
```

Causes the bad-bits mask value in the NDF called xspec to undergo a bit-wise 'OR' operation with the binary value 10001000. This causes bits 4 and 8 to be set without changing the state of any other bits in the mask.

```
setbb quasar ze7 and
```

Causes the bad-bits mask value in the NDF called quasar to undergo a bit-wise 'AND' operation with the hexadecimal value E7 (binary 11100111). This causes bits 4 and 5 to be cleared (*i.e.* reset to zero) without changing the state of any other bits in the mask.

Notes:

The bad-bits value will be disregarded if the NDF supplied does not have a QUALITY component present. A warning message will be issued if this should occur.

Related Applications :

KAPPA: QUALTOBAD, REMQUAL, SETQUAL, SHOWQUAL; FIGARO: Q2BAD.

SETBOUND
Sets new bounds for an NDF

Description:

This application sets new pixel-index bounds for an NDF , either trimming it to remove unwanted pixels, or padding it with bad pixels to achieve the required shape. The number of dimensions may also be altered. The NDF is accessed in update mode and modified *in situ*, preserving existing pixel values which lie within the new bounds.

Usage:

```
setbound ndf
```

Parameters:

LIKE = NDF (Read)

This parameter may be used to specify an NDF which is to be used as a shape template. If such a template is supplied, then its bounds will be used to determine the new shape required for the NDF specified via the NDF parameter. By default no template will be used and the new shape will be determined by means of a section specification applied to the NDF being modified (see the "Examples"). [!]

NDF = NDF (Read and Write)

The NDF whose bounds are to be modified. In normal use, an NDF section will be specified for this parameter (see the "Examples") and the routine will use the bounds of this section to determine the new bounds required for the base NDF from which the section is drawn. The base NDF is then accessed in update mode and its bounds are modified *in situ* to make them equal to the bounds of the section specified. If a section is not specified, then the NDF's shape will only be modified if a shape template is supplied via the LIKE parameter.

Examples:

```
setbound datafile(1:512,1:512)
```

Sets the pixel-index bounds of the NDF called datafile to be (1:512, 1:512), either by trimming off unwanted pixels or by padding out with bad pixels, as necessary.

```
setbound alpha(:7,56:)
```

Modifies the NDF called alpha so that its first dimension has an upper bound of 7 and its second dimension has a lower bound of 56. The lower bound of the first dimension and the upper bound of the second dimension remain unchanged.

```
setbound ndf=kg74b(,5500.0~100.0)
```

Sets new bounds for the NDF called kg74b. The bounds of the first dimension are left unchanged, but those of the second dimension are changed so that this dimension

has an extent of 100.0 centred on 5500.0, using the physical units in which this second dimension is calibrated.

```
setbound newspec like=oldspec
```

Changes the bounds of the NDF newspec so that they are equal to the bounds of the NDF called oldspec.

```
setbound xflux(:2048) like=xflux
```

Extracts the section extending from the lower bound of the one-dimensional NDF called xflux up to pixel 2048, and then modifies the bounds of this section to be equal to the original bounds of xflux, replacing xflux with this new NDF. This leaves the final shape unchanged, but sets all pixels from 2049 onwards to be equal to the bad-pixel value.

```
setbound whole(5:10,5:10) like=whole(0:15,0:15)
```

Extracts the section (5:10, 5:10) from the base NDF called whole and then sets its bounds to be equal to those of the section whole(0:15, 0:15), replacing whole with this new NDF. The effect is to select a 6-pixel-square region from the original NDF and then to pad it with a 5-pixel-wide border of bad pixels.

Notes:

This routine modifies the NDF *in situ* and will not release unused file space if the size of the NDF is reduced. If recovery of unused file space is required, then the related application NDFCOPY should be used. This will copy the selected region of an NDF to a new data structure from which any unused space will be eliminated.

Related Applications :

KAPPA: NDFCOPY, SETORIGIN; FIGARO: ISUBSET.

SETEXT
Manipulates the contents of a specified NDF extension

Description:

This task enables the contents of a specified NDF extension to be edited. It can create a new extension or delete an existing one, can create new scalar components within an extension, or modify or display the values of existing scalar components within the extension. The task operates on only one extension at a time, and must be closed down and restarted to work on a new extension.

The task may operate in one of two modes, according to the LOOP parameter. When LOOP=FALSE only a single option is executed at a time, making the task suitable for use from an ICL procedure. When LOOP=TRUE several options may be executed at once, making it easier to modify several extension components interactively in one go.

Usage:

```
                                  { ok
                                  { ctype=?  shape=?  ok
setext ndf xname option cname     { newname=?
                                  { xtype=?
                                option
```

Parameters:

CNAME = LITERAL (Read)

The name of component (residing within the extension) to be examined or modified. It is only accessed when OPTION="Erase", "Get", "Put", or "Rename".

CTYPE = LITERAL (Read)

The type of component (residing within the extension) to be created. Allowed values are "LITERAL", "_LOGICAL", "_DOUBLE", "_REAL", "_INTEGER", "_CHAR", "_BYTE", "_UBYTE", "_UWORD", "_WORD". The length of the character type may be defined by appending the length, for example, "_CHAR*32" is a 32-character component. "LITERAL" and "_CHAR" generate 80-character components. CTYPE is only accessed when OPTION="Put".

CVALUE = LITERAL (Read)

The value(s) for the component. Each value is converted to the appropriate data type for the component. CVALUE is only accessed when OPTION="Put". Note that for an array of values the list must be enclosed in brackets, even in response to a prompt. For convenience, if LOOP=TRUE, you are prompted for each string.

LOOP = _LOGICAL (Read)

LOOP=FALSE requests that only one operation be performed. This allows batch and non-interactive processing or use in procedures. LOOP=TRUE makes SETEXT operate in a looping mode that allows several modifications and/or examinations to be made to the NDF for one activation. Setting OPTION to "Exit" will end the looping. [TRUE]

NDF = NDF (Update)
The NDF to modify or examine.

NEWNAME = LITERAL (Read)
The new name of a renamed extension component. It is only accessed when OPTION="Rename".

OK = _LOGICAL (Read)
This parameter is used to seek confirmation before a component is erased or overwritten. A TRUE value permits the operation. A FALSE value leaves the existing component unchanged. This parameter is ignored when LOOP=FALSE.

OPTION = LITERAL (Read)
The operation to perform on the extension or a component therein. The recognised options are listed below.

"Delete" — Delete an existing NDF extension.
"Erase" — Erase a component within an NDF extension
"Exit" — Exit from the task (when LOOP=TRUE)
"Get" — Display the value of a component within an NDF extension. The component must exist.
"Put" — Change the value of a component within an NDF extension or create a new component.
"Rename" — Renames a component. The component must exist.
"Select"] — Selects another extension. If the extension does not exist a new one is created. This option is not allowed when LOOP=FALSE.

The suggested default is the current value, except for the first option where there is no default.

SHAPE( ) = _INTEGER (Read)
The shape of the component. Thus 3,2 would be a two-dimensional object with three elements along each of two lines. 0 creates a scalar. The suggested default is the shape of the object if it already exists, otherwise it is the current value. It is only accessed when OPTION="Put".

XNAME = LITERAL (Read)
The name of the extension to modify.

XTYPE = LITERAL (Read)
The type of the extension to create. The suggested default is the current value or "EXT" when there is no current value.

Examples:

```
setext hh50 fits delete noloop
```

This deletes the FITS extension in the NDF called hh50.

```
setext myndf select xtype=mytype noloop
```

This creates the extension MYEXT of data type MYTYPE in the NDF called myndf.

```
setext xname=ccdpack ndf=abc erase cname=filter noloop
```

This deletes the FILTER component of the CCDPACK extension in the NDF called abc.

```
setext abc ccdpack put cname=filter cvalue=B ctype=_char noloop
```

This assigns the character value "B" to the FILTER component of the CCDPACK extension a the NDF called abc.

```
setext virgo plate put cname=pitch shape=2 cvalue=[32,16] ctype=_byte
noloop
```

This sets the byte two-element vector of component PITCH of the PLATE extension in the NDF called virgo. The first element of PITCH is set to 32 and the second to 16.

```
setext virgo plate rename cname=filter newname=waveband noloop
```

This renames the FILTER component of the PLATE extension in the NDF called virgo to WAVEBAND.

Notes:

- The "Put" option allows the creation of extension components with any of the primitive data types.
- The task creates the extension automatically if it does not exist and only allows one extension to be modified at a time.

Related Applications :

KAPPA: FITSIMP, FITSLIST, NDFTRACE; CCDPACK: CCDEDIT; FIGARO: FITSKEYS; HDSTRACE; IRAS90: IRASTRACE, PREPARE.

SETLABEL
Sets a new label for an NDF data structure

Description:

This routine sets a new value for the LABEL component of an existing NDF data structure. The NDF is accessed in update mode and any pre-existing label is over-written with a new value. Alternatively, if a 'null' value (`!`) is given for the LABEL parameter, then the NDF's label will be erased.

Usage:

```
setlabel ndf label
```

Parameters:

LABEL = LITERAL (Read)

The value to be assigned to the NDF's LABEL component. This should describe the type of quantity represented in the NDF's data array (*e.g.* `"Surface Brightness"` or `"Flux Density"`). The value may later be used by other applications, for instance to label the axes of graphs where the NDF's data values are plotted. The suggested default is the current value.

NDF = NDF (Read and Write)

The NDF data structure whose label is to be modified.

Examples:

```
setlabel ngc1068 "Surface Brightness"
```

Sets the LABEL component of the NDF structure ngc1068 to be `"Surface Brightness"`.

```
setlabel ndf=datastruct label="Flux Density"
```

Sets the LABEL component of the NDF structure datastruct to be `"Flux Density"`.

```
setlabel raw_data label=!
```

By specifying a null value (`!`), this example erases any previous value of the LABEL component in the NDF structure raw_data.

Related Applications :

KAPPA: AXLABEL, SETTITLE, SETUNITS.

SETMAGIC
Replaces all occurrences of a given value in an NDF array with the bad value

Description:

This task flags all pixels that have a defined value in an NDF with the standard bad ('magic') value. Other values are unchanged. The number of replacements is reported. SETMAGIC's applications include the import of data from software that has a different magic value.

Usage:

```
setmagic in out repval [comp]
```

Parameters:

COMP = LITERAL (Read)

The components whose values are to be flagged as bad. It may be "Data", "Variance", "Error", or "All". The last of the options forces substitution of bad pixels in both the data and variance arrays. This parameter is ignored if the data array is the only array component within the NDF. ["Data"]

IN = NDF (Read)

Input NDF structure containing the data and/or variance array to have some of its elements flagged with the magic-value.

OUT = NDF (Write)

Output NDF structure containing the data and/or variance array that is a copy of the input array, but with bad values flagging the replacement value.

REPVAL = _DOUBLE (Read)

The element value to be substituted with the bad value. The same value is replaced in both the data and variance arrays when COMP="All". It must lie within the minimum and maximum values of the data type of the array with higher precision. The replacement value is converted to data type of the array being converted before the search begins. The suggested default is the current value.

TITLE = LITERAL (Read)

Title for the output NDF structure. A null value (!) propagates the title from the input NDF to the output NDF. [!]

Examples:

```
setmagic irasmap aitoff repval=-2000000
```

This copies the NDF called irasmap to the NDF aitoff, except that any pixels with the IPAC blank value of −2000000 are flagged with the standard bad value in aitoff.

```
setmagic saturn saturnb 9999.0 comp=All
```

This copies the NDF called saturn to the NDF saturnb, except that any elements in the data and variance arrays that have value 9999.0 are flagged with the standard bad value.

Notes:

- The comparison for floating-point values tests that the difference between the replacement value and the element value is less than their mean times the precision of the data type.

Related Applications :

KAPPA: CHPIX, FILLBAD, GLITCH, NOMAGIC, SEGMENT, SUBSTITUTE, ZAPLIN; FIGARO: GOODVAR.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

SETNORM
Sets a new value for one or all of an NDF's axis-normalisation flags

Description:

This routine sets a new value for one or all the normalisation flags in an NDF AXIS data structure. The NDF is accessed in update mode. This flag determines how the NDF's data and variance arrays behave when the associated axis information is modified.

If an AXIS structure does not exist, a new one whose centres are pixel co-ordinates is created.

Usage:

```
setnorm ndf dim
```

Parameters:

ANORM = _LOGICAL (Read)

The normalisation flag for the axis. TRUE means that the data and variance values in the NDF are normalised to the pixel width values for the chosen axis so that the product of data value and width, and variance and the squared width are constant if the width is altered.

A FALSE value means that the data and variance need not alter as the pixel widths are varied. This is the default for an axis. The suggested default is the current value.

DIM = _INTEGER (Read)

The axis dimension for which the normalisation flag is to be modified. There are separate units for each NDF dimension. A value of 0 sets the normalisation flag for all the axes. The value must lie between 0 and the number of dimensions of the NDF. This defaults to 1 for a one-dimensional NDF. The suggested default is the current value. []

NDF = NDF (Read and Write)

The NDF data structure in which an axis-normalisation flag is to be modified.

Examples:

```
setnorm hd23568 0 anorm
```

This sets the normalisation flags along all axes of the NDF structure hd23568 to be true.

```
setnorm ndf=spect noanorm
```

This sets the normalisation flag of the one-dimensional NDF structure spect to be false.

```
setnorm borg 3 anorm
```

This sets the normalisation flag for the third dimension in the NDF structure borg.

Axis Normalisation :

In general, the axis-normalisation property is not needed. An example where it is relevant is a spectrum in which data values representing energy per unit wavelength and each pixel has a known spread in wavelength. The sum of each pixel's data value multiplied by its width gives the energy in a part of the spectrum. A change to the axis width, say to allow for the redshift, necessitates a corresponding modification to the data value to retain this property. In two dimensions an example is where the data measure flux per unit area of sky and the pixel widths are defined in terms of angular size.

Related Applications :

KAPPA: SETAXIS.

SETORIGIN
Sets a new pixel origin for an NDF

Description:

This application sets a new pixel origin value for an NDF data structure. The NDF is accessed in update mode and the indices of the first pixel (the NDF's lower pixel-index bounds) are set to specified integer values, which may be positive or negative. No other properties of the NDF are altered. If required, a template NDF may be supplied and the new origin values will be derived from it.

Usage:

```
setorigin ndf origin
```

Parameters:

LIKE = NDF (Read)

This parameter may be used to supply an NDF which is to be used as a template. If such a template is supplied, then its origin (its lower pixel-index bounds) will be used as the new origin value for the NDF supplied via the NDF parameter. By default, no template will be used and the new origin will be specified via the ORIGIN parameter. [!]

NDF = NDF (Read and Write)

The NDF data structure whose pixel origin is to be modified.

ORIGIN() = _INTEGER (Read)

A one-dimensional array specifying the new pixel origin values, one for each NDF dimension.

Examples:

```
setorigin image_2d [1,1]
```

Sets the indices of the first pixel in the two-dimensional image image_2d to be (1, 1). The image pixel values are unaltered.

```
setorigin ndf=starfield
```

A new pixel origin is set for the NDF structure called starfield. SETORIGIN will prompt for the new origin values, supplying the existing values as defaults.

```
setorigin ndf=cube origin=[-128,-128]
```

Sets the pixel origin values for the first two dimensions of the three-dimensional NDF called cube to be (−128, −128). A value for the third dimension is not specified, so the origin of this dimension will remain unchanged.

```
setorigin betapic like=alphapic
```

Sets the pixel origin of the NDF called betapic to be equal to that of the NDF called alphapic.

Notes:

If the number of new pixel origin values is fewer than the number of NDF dimensions, then the pixel origin of the extra dimensions will remain unchanged. If the number of values exceeds the number of NDF dimensions, then the excess values will be ignored.

Timing :

Setting a new pixel origin is a quick operation whose timing does not depend on the size of the NDF.

Related Applications :

KAPPA: SETBOUND.

SETQUAL
Assigns a specified quality to selected pixels within an NDF

Description:

This routine assigns (or optionally removes) the quality specified by Parameter QNAME to (or from) selected pixels in an NDF . For more information about using quality within KAPPA see Section 16.

The user can select the pixels to be operated on in one of three ways (see Parameter SELECT).

- By giving a 'mask' NDF. Pixels with bad values in the mask NDF will be selected from the corresponding input NDF.
- By giving a list of pixel indices for the pixels that are to be selected.
- By giving an ARD file containing a description of the regions of the NDF that are to be selected. The ARD system (see SUN/183) uses a textual language to describe geometric regions of an array. Text files containing ARD description suitable for use with this routine can be created interactively using the routine ARDGEN or with GAIA.

The operation to be performed on the pixels is specified by Parameter FUNCTION. The given quality may be assigned to or removed from pixels within the NDF. The pixels operated on can either be those selected by the user (as described above), or those not selected. The quality of all other pixels is left unchanged (unless the Parameter FUNCTION is given the value `"NS+HU"` or `"NU+HS"`). Thus for instance if pixel (1, 1) already held the quality specified by QNAME, and the quality was then assigned to pixel (2, 2) this would not cause the quality to be removed from pixel (1, 1).

This routine can also be used to copy all quality information from one NDF to another (see Parameter LIKE).

Usage:

```
setqual ndf qname comment mask
```

Parameters:

ARDFILE = FILENAME (Read)

The name of the ARD file containing a description of the parts of the NDF to be 'selected'. The ARD parameter is only prompted for if the SELECT parameter is given the value `"ARD"`. The co-ordinate system in which positions within this file are given should be indicated by including suitable COFRAME or WCS statements within the file (see SUN/183), but will default to pixel co-ordinates in the absence of any such statements. For instance, starting the file with a line containing the text `"COFRAME(SKY,System=FK5)"` would indicate that positions are specified in RA/DEC (FK5,J2000). The statement `"COFRAME(PIXEL)"` indicates explicitly that positions are specified in pixel co-ordinates.

COMMENT = LITERAL (Read)
A comment to store with the quality name. This parameter is only prompted for if the NDF does not already contain a definition of the quality name.

FUNCTION = LITERAL (Read)
This parameter specifies what function is to be performed on the 'selected' pixels specified using Parameters MASK, LIST or ARDFILE. It can take any of the following values.

- `"HS"` — Ensure that the quality specified by QNAME is held by all the selected pixels. The quality of all other pixels is left unchanged.
- `"HU"` — Ensure that the quality specified by QNAME is held by all the pixels that have not been selected. The quality of the selected pixels is left unchanged.
- `"NS"` — Ensure that the quality specified by QNAME is not held by any of the selected pixels. The quality of all other pixels is left unchanged.
- `"NU"` — Ensure that the quality specified by QNAME is not held by any of the pixels that have not been selected. The quality of the selected pixels is left unchanged.
- `"HS+NU"` — Ensure that the quality specified by QNAME is held by all the selected pixels and not held by any of the other pixels.
- `"HU+NS"` — Ensure that the quality specified by QNAME is held by all the pixels that have not been selected and not held by any of the selected pixels.

`["HS"]`

LIKE = NDF (Read)
An existing NDF from which the QUALITY component and quality names are to be copied. These overwrite any corresponding information in the NDF given by Parameter NDF. If null (`!`), then the operation of this command is instead determined by Parameter SELECT. `[!]`

LIST = LITERAL (Read)
A group of pixels positions within the input NDF listing the pixels that are to be 'selected' (see Parameter FUNCTION). Each position should be giving as a list of pixel indices (*e.g.* $X1$, $Y1$, $X2$, $Y2$,... for a two dimensional NDF). LIST is only prompted for if Parameter SELECT is given the value `"LIST"`.

MASK = NDF (Read)
A mask NDF used to define the 'selected' pixels within the input NDF (see Parameter FUNCTION). The mask should be aligned pixel-for-pixel with the input NDF. Pixels that are bad in the mask NDF are 'selected'. The quality of any pixels that lie outside the bounds of the mask NDF are left unaltered. This parameter is only prompted for if the Parameter SELECT is given the value `"MASK"`.

NDF = NDF (Update)
The NDF in which the quality information is to be stored.

QNAME = LITERAL (Read)
The quality name. If the supplied name is not already defined within the input NDF, then a definition of the name is added to the NDF. The user is warned if the quality name is already defined within the NDF.

QVALUE = _INTEGER (Read)
If not null, then the whole QUALITY array is filled with the constant value given by QVALUE, which must be in the range 0 to 255. No other changes are made to the NDF. [!]

READONLY = _LOGICAL (Read)
If TRUE, then an error will be reported if any attempt is subsequently made to remove the quality name (*e.g.* using REMQUAL). [FALSE]

SELECT = LITERAL (Read)
If Parameter LIKE is null, then this parameter determines how the pixels are selected, and can take the values "Mask", "List" or "ARD" (see Parameters MASK, LIST, and ARD). ["Mask"]

XNAME = LITERAL (Read)
If an NDF already contains any quality name definitions then new quality names are put in the same extension as the old names. If no previous quality names have been stored in the NDF then Parameter XNAME will be used to obtain the name of an NDF extension in which to store the new quality name. The extension will be created if it does not already exist (see Parameter XTYPE). [QUALITY_NAMES]

XTYPE = LITERAL (Read)
If a new NDF extension is created to hold quality names (see Parameter XNAME), then Parameter XTYPE is used to obtain the HDS data type for the created extension. The run time default is to give the extension a type identical to its name. []

Examples:

```
setqual m51 saturated "Saturated pixels" m51_cut
```

This example ensures that the quality "SATURATED" is defined within the NDF m51. The comment "Saturated pixels" is stored with the quality name if it did not already exist in the NDF. The quality SATURATED is then assigned to all pixels for which the corresponding pixel in NDF m51_CUT is bad. The quality of all other pixels is left unchanged.

```
setqual "m51,cena" source_a select=list list=^source_a.lis function=hs+nu
```

This example ensures that pixels within the two NDFs m51 and cena which are included in the list of pixel indices held in text file source_a.lis, have the quality "SOURCE_A", and also ensures that none of the pixels which were not included in source_a.lis have the quality.

```
setqual m51 source_b select=ard ardfile=background.ard
```

This example assigns the quality "source_b" to pixels of the NDF m51 as described by an ARD description stored in the text file "background.ard". This text file could for instance have been created using routine ARDGEN.

Notes:

- All the quality names which are currently defined within an NDF can be listed by application SHOWQUAL. Quality name definitions can be removed from an NDF using application REMQUAL. If there is no room for any more quality names to be added to the NDF then REMQUAL can be used to remove a quality name in order to make room for the new quality names.

Related Applications :

KAPPA: QUALTOBAD, REMQUAL, SHOWQUAL.

SETSKY
Stores new WCS information within an NDF

Description:

This application adds WCS information describing a celestial sky co-ordinate system to a two-dimensional NDF . This information can be stored either in the form of a standard NDF WCS component, or in the form of an *IRAS90 astrometry structure* (see Parameter IRAS90).

The astrometry is determined either by you supplying explicit values for certain projection parameters, or by you providing the sky and corresponding image co-ordinates for a set of positions (see Parameter POSITIONS). In the latter case, the projection parameters are determined automatically by searching through parameter space in order to minimise the sum of the squared residuals between the supplied pixel co-ordinates and the transformed sky co-ordinates. You may force particular projection parameters to take certain values by assigning an explicit value to the corresponding application parameter listed below. The individual residuals at each position can be written out to a logfile so that you can identify any aberrant points. The RMS residual (in pixels) implied by the best-fitting parameters is displayed.

Usage:

```
setsky ndf positions coords epoch [projtype] [lon] [lat] [refcode]
[pixelsize] [orient] [tilt] [logfile]
```

Parameters:

COORDS = LITERAL (Read)

The sky co-ordinate system to use. Valid values include `"Ecliptic"` (IAU 1980), `"Equatorial"` (FK4 and FK5), and `"Galactic"` (IAU 1958). Ecliptic and equatorial co-ordinates are referred to the mean equinox of a given epoch. This epoch is specified by appending it to the system name, in parentheses, for example, `"Equatorial(1994.5)"`. The epoch may be preceded by a single character, `"B"` or `"J"`, indicating that the epoch is Besselian or Julian respectively. If this letter is missing, a Besselian epoch is assumed if the epoch is less than 1984.0, and a Julian epoch is assumed otherwise.

EPOCH = _DOUBLE (Read)

The Julian epoch at which the observation was made (*e.g.* `"1994.0"`).

IRAS90 = _LOGICAL (Read)

If a `TRUE` value is supplied, then the WCS information will be stored in the form of an IRAS90 astrometry structure. This is the form used by the IRAS90 package (see SUN/163). In this case, any existing IRAS90 astrometry structure will be over-written. See the "Notes" section below for warnings about using this form.

If a `FALSE` value is supplied, then the WCS information will be stored in the form of a standard NDF WCS component which will be recognized, used and updated correctly by most other Starlink software.

If a null value (`!`) is supplied, then a `TRUE` value will be used if the supplied NDF already has an IRAS90 extension. Otherwise a `FALSE` value will be used. `[!]`

LAT = LITERAL (Read)
The latitude of the reference point, in the co-ordinate system specified by Parameter COORDS. For example, if COORDS is `"Equatorial"`, LAT is the declination. See SUN/163, Section 4.7.2 for full details of the allowed syntax for specifying this position. For convenience here are some examples how you may specify the declination -45 degrees, 12 arcminutes: `"-45 12 00"`, `"-45 12"`, `"-45d 12m"`, `"-45.2d"`, `"-451200"`, `"-0.78888r"`. The last of these is a radians value. A null value causes the latitude of the reference point to be estimated automatically from the data supplied for Parameter POSITIONS. `[!]`

LOGFILE = FILENAME (Read)
Name of the text file to log the final projection parameter values and the residual at each supplied position. If null, there will be no logging. This parameter is ignored if a null value is given to Parameter POSITIONS. `[!]`

LON= LITERAL (Read)
The longitude of the reference point, in the co-ordinate system specified by Parameter COORDS. For example, if COORDS is `"Equatorial"`, LON is the right ascension. See SUN/163, Section 4.7.2 for full details of the allowed syntax for specifying this position. For convenience here are some examples how you may specify the right ascension 11 hours, 34 minutes, and 56.2 seconds: `"11 34 56.2"`, `"11h 34m 56.2s"`, `"11 34.9366"`, `"11.58228"`, `"113456.2"`. See Parameter LAT for examples of specifying a non-equatorial longitude. A null value causes the longitude of the reference point to be estimated automatically from the data supplied for Parameter POSITIONS. `[!]`

NDF = NDF (Read and Write)
The NDF in which to store the WCS information.

ORIENT = LITERAL (Read)
The position angle of the NDF's *y* axis on the celestial sphere, measured from north through east. North is defined as the direction of increasing sky latitude, and east is the direction of increasing sky longitude. Values are constrained to the range 0 to two-pi radians. A null value causes the position angle to be estimated automatically from the data supplied for Parameter POSITIONS. `[!]`

PIXELREF(2) = REAL (Read)
The pixel co-ordinates of the reference pixel (x then y). This parameter is ignored unless REFCODE=`"Pixel"`. Remember that the centre of a pixel at indices *i,j* is $(i - 0.5, j - 0.5)$. A null value causes the pixel co-ordinates of the reference point to be estimated automatically from the data supplied for Parameter POSITIONS. `[!]`

PIXELSIZE(2) = _REAL (Read)
The *x* and *y* pixel sizes at the reference position. If only one value is given, the pixel is deemed to be square. Values may be given in a variety of units (see Parameter LAT). For example, 0.54 arcseconds could be specified as `"0.54s"` or `"0.009m"` or `"2.618E-6r"`. A null value causes the pixel dimensions to be estimated automatically from the data supplied for Parameter POSITIONS. `[!]`

POSITIONS = LITERAL (Read)
A list of sky co-ordinates and corresponding image co-ordinates for the set of positions which are to be used to determine the astrometry. If a null value is given then the astrometry is determined by the explicit values you supply for each of the other

parameters. Each position is defined by four values, the sky longitude (in the same format as for Parameter LON), the sky latitude (in the same format as for Parameter LAT), the image pixel *x* co-ordinate and the image pixel *y* co-ordinate (both decimal values). These should be supplied (in the order stated) for each position. These values are given in the form of a 'group expression' (see SUN/150). This means that values can be either typed in directly or supplied in a text file. If typed in directly, the items in the list should be separated by commas, and you are re-prompted for further values if the last supplied value ends in a minus sign. If conveyed in a text file, they should again be separated by commas, but can be split across lines. The name of the text file is given in response to the prompt, preceded by an 'up arrow' symbol (^).

PROJTYPE = LITERAL (Read)

The type of projection to use. The options are: `"Aitoff"` — Aitoff equal-area, `"Gnomonic"` — Gnomonic (*i.e.* tangent plane), `"Lambert"` — Lambert normal equivalent cylindrical, `"Orthographic"` — Orthographic.

The following synonyms are also recognised: `"All_sky"` — Aitoff, `"Cylindrical"` — Lambert, `"Tangent_plane"` — Gnomonic.

See SUN/163 for descriptions of these projections. A null value causes the projection to be determined automatically from the data supplied for Parameter POSITIONS. `[!]`

REFCODE = LITERAL (Read)

The code for the reference pixel. If it has value `"Pixel"` this requests that pixel co-ordinates for the reference point be obtained through Parameter PIXELREF. The other options are locations specified by two characters, the first corresponding to the vertical position and the second the horizontal. For the vertical, valid positions are `T`(op), `B`(ottom), or `C`(entre); and for the horizontal the options are `L`(eft), `R`(ight), or `C`(entre). Thus REFCODE=`"CC"` means the reference position is at the centre of the NDF image, and `"BL"` specifies that the reference position is at the centre of the bottom-left pixel in the image. A null value causes the pixel co-ordinates of the reference point to be estimated automatically from the data supplied for Parameter POSITIONS. `[!]`

TILT = LITERAL (Read)

The angle through which the celestial sphere is to be rotated prior to doing the projection. The axis of 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. In common circumstances this can be set to zero. Values may be given in a variety of units (see Parameter LAT). Values are constrained to the range 0 to two-pi radians. A null value causes the latitude of the reference point to be estimated automatically from the data supplied for Parameter POSITIONS. `["0.0"]`

Examples:

```
setsky m51 ^stars.lis ecl(j1994.0) 1994.0 logfile=m51.log
```

This creates a WCS component to a two-dimensional NDF called m51. The values for Parameters PROJTYPE, LON, LAT, PIXELREF, PIXELSIZE, and ORIENT are determined automatically so that they minimised the sum of the squared residuals (in pixels) at each of the positions specified in the file `stars.lis`. This file contains a line

for each position, each line containing an ecliptic longitude and latitude, followed by a pair of image co-ordinates. These values should be separated by commas. The ecliptic co-ordinates were determined at Julian epoch 1994.0, and are referred to the mean equinox at Julian epoch 1994.0. The determined parameter values together with the residual at each position are logged to file `m51.log`.

```
setsky m51 ^stars.lis ecl(j1994.0) 1994.0 orient=0 projtype=orth
```

This creates a WCS component within the two-dimensional NDF called m51. The values for Parameters PROJTYPE, LON, LAT, PIXELREF, and PIXELSIZE are determined automatically as in the previous example. In this example however, an Orthographic projection is forced, and the value zero is assigned to Parameter ORIENT, resulting in north being 'upwards' in the image.

```
setsky virgo "!" eq(j2000.0) 1989.3 gn "12 29" "+12 30" bl 1.1s 0.0d
```

This creates a WCS component within the two-dimensional NDF called virgo. It is a gnomonic projection in the equatorial system at Julian epoch 2000.0. The bottom-left pixel of the image is located at right ascension 12 hours 29 minutes, declination +12 degrees 30 minutes. A pixel at that position is square and has angular size of 1.1 arcseconds. The image was observed at epoch 1989.3. At the bottom-left of the image, north is at the top, parallel to the y-axis of the image.

```
setsky map "!" galactic(1950.0) 1993.8 aitoff 90 0 cc [0.5d,0.007r] 180.0d
```

This creates a WCS component within the two-dimensional NDF called map. It is an Aitoff projection in the galactic system at Besselian epoch 1950.0. The centre of the image is located at galactic longitude 90 degrees, latitude 0 degrees. A pixel at that position is rectangular and has angular size of 0.5 degrees by 0.007 radians. The image was made at epoch 1993.8. At the image centre, south is at the top and is parallel to the y-axis of the image.

```
setsky zodiac "!" ec 1983.4 or 10.3 -5.6 Pixel 20m 0.3d
pixelref=[9.5,-11.2] IRAS90=YES
```

This creates an IRAS90 astrometry extension within the two-dimensional NDF called zodiac. It is an orthographic projection in the Ecliptic system at Besselian epoch 1950.0. The reference point at pixel co-ordinates (9.5, −11.2) corresponds to ecliptic longitude 10.3 degrees, latitude −5.6 degrees. A pixel at that position is square and has angular size of 20 arcminutes. The image was observed at epoch 1983.4. At the reference point the y-axis of the image points to 0.3 degrees east of north.

Notes:

- The GAIA image display tool (SUN/214) provides various interactive tools for storing new WCS information within an NDF.

- This application was written to supply the limited range of WCS functions required by the IRAS90 package. For instance, it does not support the complete range or projections or sky co-ordinate systems which may be represented by the more general NDF WCS component.
- If WCS information is stored in the form of an IRAS90 astrometry structure (see Parameter IRAS90), it will in general be invalidated by any subsequent KAPPA commands which modify the transformation between sky and pixel co-ordinates. For instance, if the image is moved using SLIDE (for example), then the IRAS90 astrometry structure will no longer correctly describe the sky co-ordinates associated with each pixel. For this reason (amongst others) it is better to set Parameter IRAS90 to FALSE.

Related Applications :

ASTROM; IRAS90: SKYALIGN, SKYBOX, SKYGRID, SKYLINE, SKYMARK, SKYPOS, SKYWRITE.

SETTITLE
Sets a new title for an NDF data structure

Description:

This routine sets a new value for the TITLE component of an existing NDF data structure. The NDF is accessed in update mode and any pre-existing title is over-written with a new value. Alternatively, if a 'null' value (`!`) is given for the TITLE parameter, then the NDF's title will be erased.

Usage:

```
settitle ndf title
```

Parameters:

NDF = NDF (Read and Write)
The NDF data structure whose title is to be modified.

TITLE = LITERAL (Read)
The value to be assigned to the NDF's TITLE component (*e.g.* `"NGC1068 with a B filter"` or `"Ice band in HD123456"`). This value may later be used by other applications as a heading for graphs and other forms of display where the NDF's data values are plotted. The suggested default is the current value.

Examples:

```
settitle ngc1068 "NGC1068 with a B filter"
```

Sets the TITLE component of the NDF structure ngc1068 to be `"NGC1068 with a B filter"`.

```
settitle ndf=myspec title="Ice band, short integration"
```

Sets the TITLE component of the NDF structure myspec to be `"Ice band, short integration"`.

```
settitle dat123 title=!
```

By specifying a null value (`!`), this example erases any previous value of the TITLE component in the NDF structure dat123.

Related Applications :

KAPPA: SETLABEL, SETUNITS.

SETTYPE
Sets a new numeric type for the DATA and VARIANCE components of an NDF

Description:

This application allows the numeric type of the DATA and VARIANCE components of an NDF to be changed. The NDF is accessed in update mode and the values stored in these components are converted in situ to the new type. No other attributes of the NDF are changed.

Usage:

```
settype ndf type
```

Parameters:

COMPLEX = _LOGICAL (Read)

If a TRUE value is given for this parameter, then the NDF's array components will be altered so that they hold complex values, an imaginary part containing zeros being created if necessary. If a FALSE value is given, then the components will be altered so that they hold non-complex values, any imaginary part being deleted if necessary. If a null (!) value is supplied, the value used is chosen so that no change is made to the current state. [!]

DATA = _LOGICAL (Read)

If a TRUE value is given for this parameter, then the numeric type of the NDF's data array will be changed. Otherwise, this component's type will remain unchanged. [TRUE]

NDF = NDF (Read and Write)

The NDF data structure whose array components are to have their numeric type changed.

TYPE = LITERAL (Read)

The new numeric type to which the NDF's array components are to be converted. The value given should be one of the following: _DOUBLE, _REAL, _INTEGER, _WORD, _UWORD, _BYTE or _UBYTE (note the leading underscore). Existing pixel values stored in the NDF will not be lost, but will be converted to the new type. Any values which cannot be represented using the new type will be replaced with the bad-pixel value.

VARIANCE = _LOGICAL (Read)

If a TRUE value is given for this parameter, then the numeric type of the NDF's VARIANCE array will be changed. Otherwise, this component's type will remain unchanged. [TRUE]

Examples:

```
settype rawdata _real
```

Converts the data and variance values held in the NDF data structure rawdata

to have a numeric type of _REAL (*i.e.* to be stored as single-precision floating-point numbers).

```
settype inst.run1 _word novariance
```

Converts the data array in the NDF structure inst.run1 to be stored as word (*i.e.* Fortran INTEGER∗2) values. No change is made to the VARIANCE component.

```
settype hd26571 _double complex
```

Causes the DATA and VARIANCE components of the NDF structure hd26571 to be altered so as to hold complex values using double precision numbers. The existing pixel values are converted to this new type.

Timing :

The execution time is approximately proportional to the number of pixel values to be converted.

Related Applications :

FIGARO: RETYPE.

SETUNITS
Sets a new units value for an NDF data structure

Description:

This routine sets a new value for the UNITS component of an existing NDF data structure. The NDF is accessed in update mode and any pre-existing UNITS component is overwritten with a new value. Alternatively, if a 'null' value (`!`) is given for the UNITS parameter, then the NDF's UNITS component will be erased.

There is also an option to modify the pixel values within the NDF to reflect the change in units (see Parameter MODIFY).

Usage:

```
setunits ndf units
```

Parameters:

NDF = NDF (Read and Write)
The NDF data structure whose UNITS component is to be modified.

MODIFY = _LOGICAL (Read)
If a `TRUE` value is supplied, then the pixel values in the DATA and VARIANCE components of the NDF will be modified to reflect the change in units. For this to be possible, both the original Units value in the NDF and the new Units value must both correspond to the format for units strings described in the FITS WCS standard (see `"Representations of world coordinates in FITS"`, Greisen & Calabretta, 2002, A&A—available at (http://www.aoc.nrao.edu/∼egreisen/wcs_AA.ps.gz) If either of the two units strings are not of this form, or if it is not possible to find a transformation between them (for instance, because they represent different quantities), an error is reported. `[FALSE]`

UNITS = LITERAL (Read)
The value to be assigned to the NDF's UNITS component (*e.g.* `"J/(m**2*Angstrom*s)"` or `"count/s"`). This value may later be used by other applications for labelling graphs and other forms of display where the NDF's data values are shown. The suggested default is the current value.

Examples:

```
setunits ngc1342 "count/s"
```

Sets the UNITS component of the NDF structure ngc1342 to have the value `"count/s"`. The pixel values are not changed.

```
setunits ndf=spect units="J/(m**2*Angstrom*s)"
```

Sets the UNITS component of the NDF structure spect to have the value `"J/(m**2*Angstrom*s)"`. The pixel values are not changed.

```
setunits datafile units=!
```

By specifying a null value (!), this example erases any previous value of the UNITS component in the NDF structure datafile. The pixel values are not changed.

```
setunits ndf=spect units="MJy" modify
```

Sets the UNITS component of the NDF structure spect to have the value "MJy". If possible, the pixel values are changed from their old units to the new units. For instance, if the UNITS component of the NDF was original "J/(m**2*s*GHz)", the DATA values will be multiplied by 1.0E11, and the variance values by 1.0E22. However, if the original UNITS component was (say) "K" (Kelvin) then an error would be reported since there is no direct conversion from Kelvin to Megajansky.

Related Applications :

KAPPA: AXUNITS, SETLABEL, SETTITLE.

SETVAR
Sets new values for the VARIANCE component of an NDF data structure

Description:

This routine sets new values for the VARIANCE component of an NDF data structure. The new values can be copied from a specified component of a second NDF or can be generated from the supplied NDF's data array by means of a Fortran-like arithmetic expression. Any previous variance information is over-written with the new values. Alternatively, if a 'null' value (`!`) is given for the variance, then any pre-existing variance information is erased.

Usage:

```
setvar ndf variance
```

Parameters:

COMP = LITERAL (Read)

The name of an NDF array component within the NDF specified by Parameter FROM. The values in this array component are used as the new variance values to be stored in the VARIANCE component of the NDF specified by Parameter NDF. The supplied value must be one of `"Data"` or `"Variance"`. `["Data"]`

FROM = NDF (Read)

An NDF data structure containing the values to be used as the new variance values. The NDF component from which to read the new variance values is specified by Parameter COMP. If NDF is not contained completely within FROM, then the VARIANCE component of NDF will be padded with bad values. If a null (`!`) value is supplied, the new variance values are determined by the expression given for Parameter VARIANCE. `[!]`

NDF = NDF (Read and Write)

The NDF data structure whose variance values are to be modified.

VARIANCE = LITERAL (Read)

A Fortran-like arithmetic expression giving the variance value to be assigned to each pixel in terms of the variable DATA, which represents the value of the corresponding data array pixel. For example, VARIANCE=`"DATA"` implies normal $\sqrt{N}$ error estimates, whereas VARIANCE=`"DATA + 50.7"` might be used if a sky background of 50.7 units had previously been subtracted.

If a 'null' value (`!`) is given for this parameter, then no new VARIANCE component will be created and any pre-existing variance values will be erased.

Examples:

```
setvar ngc4709 data
```

This sets the VARIANCE component within the NDF structure ngc4709 to equal its corresponding data-array component.

```
setvar ndf=arcspec "data - 0.31"
```

This sets the VARIANCE component within the NDF structure arcspec to be its corresponding data-array component less a constant 0.31.

```
setvar cube4 Variance=!
```

This erases the values of the VARIANCE component within the NDF structure cube4, if it exists.

Notes:

- All of the standard Fortran 77 intrinsic functions are available for use in the variance expression, plus a few others (see SUN/61 for details and an up-to-date list).
- Calculations are performed using real arithmetic (or double precision if appropriate) and are constrained to be non-negative.
- The data type of the VARIANCE component is set to match that of the DATA component.

Related Applications :

KAPPA: ERRCLIP; FIGARO: GOODVAR.

SHADOW
Enhances edges in a two-dimensional NDF using a shadow effect

Description:

This routine enhances a two-dimensional NDF by creating a bas-relief or shadow effect, that causes features in an array to appear as though they have been illuminated from the side by some imaginary light source. The enhancement is useful in locating edges and fine detail in an array.

Usage:

```
shadow in out
```

Parameters:

IN = NDF (Read)
The two-dimensional NDF to be enhanced.

OUT = NDF (Write)
The output NDF containing the enhanced image.

SHIFT(2) = _INTEGER (Read)
The shift in x and y pixel indices to be used in the enhancement. If the x shift is positive, positive features in the original array will appear to be lit from the positive x direction, *i.e.* from the right. Similarly, if the y shift is positive, the light source will appear to be shining from the top of the array. A one- or two-pixel shift is normally adequate. `[1,1]`

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
shadow horse horse_bas
```

This enhances the NDF called horse by making it appear to be illuminated from the top-right, and stores the result in the NDF called horse_bas.

```
shadow out=aash in=aa [-1,-1] title="Bas relief"
```

This enhances the NDF called aa by making it appear to be illuminated from the bottom left, and stores the result in the NDF called aash, which has the title `"Bas relief"`.

Related Applications :

KAPPA: LAPLACE, MEDIAN; FIGARO: ICONV3.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- The output NDF will be trimmed compared with the input NDF by the shifts applied.

SHOWQUAL
Display the quality names defined in an NDF

Description:

This routine displays a list of all the quality names currently defined within a supplied NDF (see task SETQUAL). The descriptive comments which were stored with the quality names when they were originally defined are also displayed. An option exists for also displaying the number of pixels which hold each quality.

Usage:

```
showqual ndf [count]
```

Parameters:

COUNT = _LOGICAL (Read)

If TRUE, then the number of pixels in each NDF which holds each defined quality is displayed. These figures are shown in parentheses between the quality name and associated comment. This option adds significantly to the run time. [NO]

NDF = NDF (Read)

The NDF whose quality names are to be listed.

Results Parameters:

QNAMES( ) = LITERAL (Write)

The quality names associated with each bit, starting from the lowest significant bit. Unassigned bits have blank strings.

Examples:

```
showqual "m51,cena" yes
```

This example displays all the quality names currently defined for the two NDFs m51 and cena together with the number of pixels holding each quality.

Related Applications :

KAPPA: REMQUAL, QUALTOBAD, SETQUAL.

SLIDE
Realigns an NDF using a translation

Description:

The pixels of an NDF are shifted by a given number of pixels along each pixel axis. The shift need not be an integer number of pixels, and pixel interpolation will be performed if necessary using the scheme selected by Parameter METHOD. The shifts to use are specified either by an absolute vector given by the ABS parameter or by the difference between a fiducial point and a standard object given by the FID and OBJ parameters respectively. In each case the co-ordinates are specified in the NDF's pixel co-ordinate Frame.

Usage:

```
slide in out abs method
```

Parameters:

ABS( ) = _DOUBLE (Read)

Absolute shifts in pixels. The number of values supplied must match the number of pixel axes in the NDF. It is only used if STYPE="Absolute".

FID( ) = _DOUBLE (Read)

Position of the fiducial point in pixel co-ordinates. The number of values supplied must match the number of pixel axes in the NDF. It is only used if STYPE="Relative".

An object centred at the pixel co-ordinates given by Parameter OBJ in the input NDF will be centred at the pixel co-ordinates given by Parameter FID in the output NDF.

IN = NDF (Read)

The NDF to be translated.

METHOD = LITERAL (Read)

The interpolation method used to perform the translation. The following values are permitted:

- "Nearest" — Nearest-neighbour sampling.
- "Linear" — Linear interpolation.
- "Sinc" — Sum of surrounding pixels weighted using a one-dimensional $\mathrm{sinc}(\pi x)$ kernel.
- "SincSinc" — Sum of surrounding pixels weighted using a one-dimensional $\mathrm{sinc}(\pi x)\mathrm{sinc}(k\pi x)$ kernel.
- "SincCos" — Sum of surrounding pixels weighted using a one-dimensional $\mathrm{sinc}(\pi x)\cos(k\pi x)$ kernel.
- "SincGauss" — Sum of surrounding pixels weighted using a one-dimensional $\mathrm{sinc}(\pi x)e^{-kx^2}$ kernel.
- "BlockAve" — Block averaging over all pixels in the surrounding n-dimensional cube.

In the above, $\mathrm{sinc}(z) = \sin(z)/z$. Some of these schemes will require additional parameters to be supplied via the PARAMS parameter. A more-detailed discussion of these schemes is given in the "Sub-pixel Interpolation Schemes" section below. The initial default is "Linear". [current value]

OBJ = LITERAL (Read)
Position of the standard object in pixel co-ordinates. The number of values supplied must match the number of pixel axes in the NDF. It is only used if STYPE="Relative". An object centred at the pixel co-ordinates given by Parameter OBJ in the input NDF will be centred at the pixel co-ordinates given by Parameter FID in the output NDF.

OUT = NDF (Write)
The translated NDF.

PARAMS( ) = _DOUBLE (Read)
Parameters required to control the resampling scheme. One or more values may be required to specify the exact resampling behaviour, according to the value of the METHOD parameter. See the section on "Sub-pixel Interpolation Schemes".

STYPE = LITERAL (Read)
The sort of shift to be used. The choice is "Relative" or "Absolute". ["Absolute"]

TITLE = LITERAL (Read)
Title for the output NDF. A null (!) value will cause the input title to be used. [!]

Examples:

```
slide m31 m31_acc [3.2,2.3]
```

The pixels in the NDF m31 are shifted by 3.2 pixels in x and 2.3 pixels in y, and written to NDF m31_acc. Linear interpolation is used to produce the output data (and, if present, variance) array.

```
slide m31 m31_acc [3.2,2.3] nearest
```

The same as the previous example except that nearest-neighbour resampling is used. This will be somewhat faster, but may result in features shifted by up to half a pixel.

```
slide speca specb stype=rel fid=11.2 obj=11.7
```

The pixels in the NDF speca are shifted by 0.5 (*i.e.* $11.7 - 11.2$) pixels and the output NDF is written as specb.

```
slide speca specb stype=abs abs=0.5
```

This does just the same as the previous example.

Sub-Pixel Interpolation Schemes :

When performing the translation the pixels are resampled from the input grid to the output grid by default using linear interpolation. For many purposes this default scheme will be adequate, but for greater control over the resampling process the METHOD and PARAMS parameters can be used. Detailed discussion of the use of these parameters can be found in the "Sub-pixel Interpolation Schemes" section of the AST_RESAMPLE documentation.

Notes:

- If the NDF is shifted by a whole number of pixels along each axis, this application merely changes the pixel origin in the NDF. It can thus be compared to the SETORIGIN command.
- Resampled axis centres that are beyond the bounds of the input NDF are given extrapolated values from the first (or last) pair of valid centres.

Implementation Status:

- The LABEL, UNITS, and HISTORY components, and all extensions are propagated. TITLE is controlled by the TITLE parameter. DATA, VARIANCE, AXIS and WCS are propagated after appropriate modification. The QUALITY component is also propagated if nearest-neighbour interpolation is being used.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- There can be an arbitrary number of NDF dimensions.

Related Applications :

KAPPA: REGRID, SQORST, WCSADD.

SQORST
Squashes or stretches an NDF

Description:

An output NDF is produced by squashing or stretching an input NDF along one or more of its dimensions. The shape of the output NDF can be specified in one of two ways, according to the value of the MODE parameter; either a distortion factor is given for each dimension, or its lower and upper pixel bounds are given explicitly.

Usage:

```
                  { factors
sqorst in out     { lbound=?  ubound=?
                  { pixscale=?
                mode
```

Parameters:

AXIS = _INTEGER (Read)

Assigning a value to this parameter indicates that a single axis should be squashed or stretched. If a null (!) value is supplied for AXIS, a squash or stretch factor must be supplied for each axis in the manner indicated by the MODE parameter. If a non-null value is supplied for AXIS, it should be the integer index of the axis to be squashed or stretched (the first axis has index 1). In this case, only a single squash or stretch factor should be supplied, and all other axes will be left unchanged. If MODE is set to "PixelScale", then the supplied value should be the index of a WCS axis. Otherwise it should be the index of a pixel axis. [!]

CENTRE = LITERAL (Read)

Determines the centre about which the WCS co-ordinates are stretching or squashing. The following values are permitted.

- "Centre" — The WCS co-ordinates at the centre of the output NDF are the same as those at the centre of the input NDF.
- "Origin" — The WCS co-ordinates at the pixel origin of the output NDF are the same as those at the pixel origin of the input NDF.

["Centre"]

CONSERVE = _LOGICAL (Read)

If set TRUE, then the output pixel values will be scaled in such a way as to preserve the total data value in a feature on the sky. The scaling factor is the ratio of the output pixel size to the input pixel size. This ratio is evaluated once for each panel of a piece-wise linear approximation to the Mapping, and is assumed to be constant for all output pixels in the panel. [FALSE]

FACTORS( ) = _DOUBLE (Read)

This parameter is used only if MODE="Factors". It defines the factor by which each

dimension will be distorted to produce the output NDF. A factor greater than one is a stretch and less than one is a squash. If no value has been supplied for Parameter AXIS, the number of values supplied for FACTORS must be the same as the number of pixel axes in the NDF. If a non-null value has been supplied for Parameter AXIS, then only a single value should be supplied for FACTORS and that value will be used to distort the axis indicated by Parameter AXIS.

IN = NDF (Read)
The NDF to be squashed or stretched.

LBOUND( ) = _INTEGER (Read)
This parameter is only used if MODE="Bounds". It specifies the lower pixel-index values of the output NDF. If no value has been supplied for Parameter AXIS, the number of values supplied for LBOUND must be the same as the number of pixel axes in the NDF. If a non-null value has been supplied for Parameter AXIS, then only a single value should be supplied for LBOUND and the supplied value will be used as the new lower bounds on the axis indicated by Parameter AXIS. If null (!) is given, the lower pixel bounds of the input NDF will be used.

METHOD = LITERAL (Read)
The interpolation method used to perform the one-dimensional resampling operations which constitute the squash or stretch. The following values are permitted.

- "Auto" — Equivalent to "BlockAve" with an appropriate PARAMS for squashes by a factor of 2 or more, otherwise equivalent to "Linear".
- "Nearest" — Nearest-neighbour sampling.
- "Linear" — Linear interpolation.
- "Sinc" — Sum of surrounding pixels weighted using a one-dimensional $\mathrm{sinc}(\pi x)$ kernel.
- "SincSinc" — Sum of surrounding pixels weighted using a one-dimensional $\mathrm{sinc}(\pi x)\mathrm{sinc}(k\pi x)$ kernel.
- "SincCos" — Sum of surrounding pixels weighted using a one-dimensional $\mathrm{sinc}(\pi x)\cos(k\pi x)$ kernel.
- "SincGauss" — Sum of surrounding pixels weighted using a one-dimensional $\mathrm{sinc}(\pi x)e^{-kx^2}$ kernel.
- "BlockAve" — Block averaging over surrounding pixels.

In the above, $\mathrm{sinc}(z) = \sin(z)/z$. Some of these schemes will require additional parameters to be supplied via the PARAMS parameter. A more-detailed discussion of these schemes is given in the "Sub-Pixel Interpolation Schemes" section below. ["Auto"]

MODE = LITERAL (Read)
This determines how the shape of the output NDF is to be specified. The allowed values and their meanings are as follows.

- "Factors" — the FACTORS parameter will be used to determine the factor by which each dimension should be multiplied.
- "Bounds" — the LBOUND and UBOUND parameters will be used to get the lower and upper pixel bounds of the output NDF.

- "PixelScale" — the PIXSCALE parameter will be used to obtain the new pixel scale to use for each WCS axis.

["Factors"]

OUT = NDF (Write)
The squashed or stretched NDF.

PARAMS( ) = _DOUBLE (Read)
Parameters required to control the resampling scheme. One or more values may be required to specify the exact resampling behaviour, according to the value of the METHOD parameter. See the section on "Sub-pixel Interpolation Schemes".

PIXSCALE = LITERAL (Read)
The PIXSCALE parameter is only used if Parameter MODE is set to "PixelScale". It should be supplied as a comma-separated list of the required new pixel scales. In this context, a pixel scale for a WCS axis is the increment in WCS axis value caused by a movement of one pixel along the WCS axis, and are measured at the first pixel in the array. Pixel scales for celestial axes should be given in arcseconds. An asterisk, "*", can be used instead of a numerical value to indicate that an axis should retain its current scale. The suggested default values are the current pixel scales. If no value has been supplied for Parameter AXIS, the number of values supplied for PIXSCALE must be the same as the number of WCS axes in the NDF. If a non-null value has been supplied for Parameter AXIS, then only a single value should be supplied for PIXSCALE and that value will be used as the new pixel scale on the WCS axis indicated by Parameter AXIS.

TITLE = LITERAL (Read)
Title for the output NDF. A null (!) value causes the input title to be used. [!]

UBOUND( ) = _INTEGER (Read)
This parameter is only used if MODE="Bounds". The upper pixel index values of the output NDF. If no value has been supplied for Parameter AXIS, the number of values supplied for UBOUND must be the same as the number of pixel axes in the NDF. If a non-null value has been supplied for Parameter AXIS, then only a single value should be supplied for UBOUND and the supplied value will be used as the new upper bounds on the axis indicated by Parameter AXIS. If null (!) is given, the upper pixel bounds of the input NDF will be used.

Examples:

```
sqorst block blocktall [1,2,1]
```

The three-dimensional NDF called block is stretched by a factor of two along its second axis to produce an NDF called blocktall with twice as many pixels. The same data block is represented, but each pixel in the output NDF corresponds to half a pixel in the input NDF. The default resampling scheme, linear interpolation in the stretch direction, is used.

```
sqorst block blocktall [1,2,1] method=sincsinc params=[2,2]
```

The same operation as the previous example is performed, except that a Lanczos kernel is used for the interpolation.

```
sqorst cygnus1 squish1 mode=bounds lbound=[1,1] ubound=[50,50]
```

This turns the two-dimensional NDF cygnus1 into a new NDF squish1 which has 50 pixels along each side. The same region of sky is represented, but the input image is squashed along both axes to fit the specified dimensions.

```
sqorst fred mode=pixelscale pixscale=5 axis=3
```

This resamples a cube NDF called fred on to a velocity scale of 5 km/s per pixel along its third axis.

Notes:

If the input NDF contains a VARIANCE component, a VARIANCE component will be written to the output NDF. It will be calculated on the assumption that errors on the input data values are statistically independent and that their variance estimates may simply be summed (with appropriate weighting factors) when several input pixels contribute to an output data value. If this assumption is not valid, then the output error estimates may be biased. In addition, note that the statistical errors on neighbouring output data values (as well as the estimates of those errors) may often be correlated, even if the above assumption about the input data is correct, because of the sub-pixel interpolation schemes employed.

Sub-Pixel Interpolation Schemes :

When squashing or stretching an NDF, a separate one-dimensional resampling operation is performed for each of the dimensions in which a resize is being done. By default (when METHOD="Auto") this is done using linear interpolation, unless it is a squash of a factor of two or more, in which case a block-averaging scheme which averages over 1/FACTOR pixels. For many purposes this default scheme will be adequate, but for greater control over the resampling process the METHOD and PARAMS parameters can be used. Detailed discussion of the use of these parameters can be found in the "Sub-pixel Interpolation Schemes" section of the AST_RESAMPLE documentation. By default, all interpolation schemes preserve flux density rather than total flux, but this may be changed using the CONSERVE parameter.

Implementation Status:

- The LABEL, UNITS, and HISTORY components, and all extensions are propagated. TITLE is controlled by the TITLE parameter. DATA, VARIANCE, AXIS and WCS are propagated after appropriate modification. The QUALITY component is also propagated if nearest-neighbour interpolation is being used.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- There can be an arbitrary number of NDF dimensions.

Related Applications :

KAPPA: REGRID, SLIDE, WCSADD.

STATS
Computes simple statistics for an NDF's pixels

Description:

This application computes and displays simple statistics for the pixels in an NDF's data, quality or variance array. The statistics available are:

- the pixel sum,
- the pixel mean,
- the pixel population standard deviation,
- the pixel population skewness and excess kurtosis,
- the value and position of the minimum- and maximum-valued pixels,
- the total number of pixels in the NDF,
- the number of pixels used in the statistics, and
- the number of pixels omitted.

Iterative κ-sigma clipping may also be applied as an option (see Parameter CLIP).

Order statistics (median and percentiles) may optionally be derived and displayed (see Parameters ORDER and PERCENTILES). Although this can be a relatively slow operation on large arrays, unlike HISTAT the reported order statistics are accurate, not approximations, irrespective of the distribution of values being analysed.

Usage:

```
stats ndf [comp] [clip] [logfile]
```

Parameters:

CLIP( ) = _REAL (Read)

An optional one-dimensional array of clipping levels to be applied, expressed as standard deviations. If a null value is supplied for this parameter (the default), then no iterative clipping will take place and the statistics computed will include all the valid NDF pixels.

If an array of clipping levels is given, then the routine will first compute statistics using all the available pixels. It will then reject all those pixels whose values lie outside κ standard deviations of the mean (where κ is the first value supplied) and will then re-evaluate the statistics. This rejection iteration is repeated in turn for each value in the CLIP array. A maximum of five values may be supplied, all of which must be positive. `[!]`

COMP = LITERAL (Read)

The name of the NDF array component for which statistics are required: `"Data"`, `"Error"`, `"Quality"` or `"Variance"` (where `"Error"` is the alternative to `"Variance"` and causes the square root of the variance values to be taken before computing the statistics). If `"Quality"` is specified, then the quality values are treated as numerical values (in the range 0 to 255). `["Data"]`

LOGFILE = FILENAME (Write)
A text file into which the results should be logged. If a null value is supplied (the default), then no logging of results will take place. [!]

NDF = NDF (Read)
The NDF data structure to be analysed.

ORDER = _LOGICAL (Read)
Whether or not to calculate order statistics. If set TRUE the median and optionally percentiles are determined and reported. [FALSE]

PERCENTILES(100) = _REAL (Read)
A list of percentiles to be found. None are computed if this parameter is null (!). The percentiles must be in the range 0.0 to 100.0. This parameter is ignored unless ORDER is TRUE. [!]

Results Parameters:

KURTOSIS = _DOUBLE (Write)
The population excess kurtosis of all the valid pixels in the NDF array. This is the normal kurtosis minus 3, such that a Gaussian distribution of values would generate an excess kurtosis of 0.

MAXCOORD( ) = _DOUBLE (Write)
A one-dimensional array of values giving the WCS co-ordinates of the centre of the (first) maximum-valued pixel found in the NDF array. The number of co-ordinates is equal to the number of NDF dimensions.

MAXIMUM = _DOUBLE (Write)
The maximum pixel value found in the NDF array.

MAXPOS( ) = _INTEGER (Write)
A one-dimensional array of pixel indices identifying the (first) maximum-valued pixel found in the NDF array. The number of indices is equal to the number of NDF dimensions.

MAXWCS = LITERAL (Write)
The formatted WCS co-ordinates at the maximum pixel value. The individual axis values are comma separated.

MEAN = _DOUBLE (Write)
The mean value of all the valid pixels in the NDF array.

MEDIAN = _DOUBLE (Write)
The median value of all the valid pixels in the NDF array when ORDER is TRUE.

MINCOORD( ) = _DOUBLE (Write)
A one-dimensional array of values giving the data co-ordinates of the centre of the (first) minimum-valued pixel found in the NDF array. The number of co-ordinates is equal to the number of NDF dimensions.

MINIMUM = _DOUBLE (Write)
The minimum pixel value found in the NDF array.

MINPOS( ) = _INTEGER (Write)
A one-dimensional array of pixel indices identifying the (first) minimum-valued pixel found in the NDF array. The number of indices is equal to the number of NDF dimensions.

MINWCS = LITERAL (Write)
The formatted WCS co-ordinates at the minimum pixel value. The individual axis values are comma separated.

NUMBAD = _INTEGER (Write)
The number of pixels which were either not valid or were rejected from the statistics during iterative κ-sigma clipping.

NUMGOOD = _INTEGER (Write)
The number of NDF pixels which actually contributed to the computed statistics.

NUMPIX = _INTEGER (Write)
The total number of pixels in the NDF (both good and bad).

PERVAL() = _DOUBLE (Write)
The values of the percentiles of the good pixels in the NDF array. This parameter is only written when one or more percentiles have been requested.

SIGMA = _DOUBLE (Write)
The population standard deviation of the pixel values in the NDF array.

SKEWNESS = _DOUBLE (Write)
The population skewness of all the valid pixels in the NDF array.

TOTAL = _DOUBLE (Write)
The sum of the pixel values in the NDF array.

Examples:

```
stats image
```

Computes and displays simple statistics for the data array in the NDF called image.

```
stats image order percentiles=[25,75]
```

As the previous example but it also reports the median, 25 and 75 percentiles.

```
stats ndf=spectrum variance
```

Computes and displays simple statistics for the variance array in the NDF called spectrum.

```
stats spectrum error
```

Computes and displays statistics for the variance array in the NDF called spectrum, but takes the square root of the variance values before doing so.

```
stats halley logfile=stats.dat
```

Computes statistics for the data array in the NDF called halley, and writes the results to a logfile called `stats.dat`.

```
stats ngc1333 clip=[3.0,2.8,2.5]
```

Computes statistics for the data array in the NDF called ngc1333, applying three iterations of κ-sigma clipping. The statistics are first calculated for all the valid pixels in the data array. Those pixels with values lying more than 3.0 standard deviations from the mean are then rejected, and the statistics are re-computed. This process is then repeated twice more, rejecting pixel values lying more than 2.8 and 2.5 standard deviations from the mean. The final statistics are displayed.

Implementation Status:

- This routine correctly processes the AXIS, DATA, VARIANCE, QUALITY, TITLE, and HISTORY components of the NDF.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using double-precision floating point.
- Any number of NDF dimensions is supported.

Related Applications :

KAPPA: HISTAT, NDFTRACE; FIGARO: ISTAT.

SUB
Subtracts one NDF data structure from another

Description:

The routine subtracts one NDF data structure from another pixel-by-pixel to produce a new NDF.

Usage:

```
sub in1 in2 out
```

Parameters:

IN1 = NDF (Read)
First NDF, from which the second NDF is to be subtracted.

IN2 = NDF (Read)
Second NDF, to be subtracted from the first NDF.

OUT = NDF (Write)
Output NDF to contain the difference of the two input NDFs.

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN1 to be used instead. `[!]`

Examples:

```
sub a b c
```

This subtracts the NDF called b from the NDF called a, to make the NDF called c. NDF c inherits its title from a.

```
sub out=c in1=a in2=b title="Background subtracted"
```

This subtracts the NDF called b from the NDF called a, to make the NDF called c. NDF c has the title `"Background subtracted"`.

Notes:

If the two input NDFs have different pixel-index bounds, then they will be trimmed to match before being subtracted. An error will result if they have no pixels in common.

Related Applications :

KAPPA: ADD, CADD, CDIV, CMULT, CSUB, DIV, MATHS, MULT.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.

- The UNITS component is propagated only if it has the same value in both input NDFs.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Huge NDFs are supported.

SUBSTITUTE
Replaces all occurrences of a given value in an NDF array with another value

Description:

This application changes all pixels that have a defined value in an NDF with an alternate value. The number of replacements is reported. Two modes are available.

- A single pair of old and new values can be supplied (see Parameters OLDVAL and NEWVAL). All occurrences of OLDVAL are replaced with NEWVAL. Other values are unchanged.
- A look-up table containing corresponding old and new values can be supplied. By default, each input pixel that equals an old value is replaced by the corresponding new value. Alternatively, all input pixels can be changed to a new value by interpolation between the input values (see Parameters LUT and INTERP).

Usage:

```
substitute in out oldval newval [comp]
```

Parameters:

COMP = LITERAL (Read)

The components whose values are to be substituted. It may be `"Data"`, `"Error"`, `"Variance"`, or `"All"`. The last of the options forces substitution in both the data and variance arrays. This parameter is ignored if the data array is the only array component within the NDF. `["Data"]`

IN = NDF (Read)

Input NDF structure containing the data and/or variance array to have some of its elements substituted.

INTERP = LITERAL (Read)

Determines how the values in the file specified by Parameter LUT are used, from the following options.

- `"None"` – Pixel values that equal an input value are replaced by the corresponding output value. Other values are left unchanged.
- `"Nearest"` – Every pixel value is replaced by the output value corresponding to the nearest input value.
- `"Linear"` – Every pixel value is replaced by an output value determined using linear interpolation between the input values.

If `"Nearest"` or `"Linear"` is used, pixel values that are outside the range of input value covered by the look-up table are set bad in the output. Additionally, an error is reported if the old data values are not montonic increasing. `["None"]`

LUT = FILENAME (Read)
The name of a text file containing a look-up table of old and new data values. If null (!) is supplied for this parameter the old and new data values will instead be obtained using Parameters OLDVAL and NEWVAL. Lines starting with a hash (#) or exclamation mark (!) are ignored, as are blank lines. Other lines should contain an old data value followed by the corresponding new data value. The way in which the values in this table are used is determined by Parameter INTERP. [!]

NEWVAL = _DOUBLE (Read)
The value to replace occurrences of OLDVAL. It must lie within the minimum and maximum values of the data type of the array with higher precision. The new value is converted to data type of the array being converted before the search begins. The suggested default is the current value. This parameter is only accessed if a null value is supplied for Parameter LUT.

OLDVAL = _DOUBLE (Read)
The element value to be replaced. The same value is substituted in both the data and variance arrays when COMP="All". It must lie within the minimum and maximum values of the data type of the array with higher precision. The replacement value is converted to data type of the array being converted before the search begins. The suggested default is the current value. This parameter is only accessed if a null value is supplied for Parameter LUT.

OUT = NDF (Write)
Output NDF structure containing the data and/or variance array that is a copy of the input array, but with replacement values substituted.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (!) propagates the title from the input NDF to the output NDF. [!]

TYPE = LITERAL (Read)
The numeric type for the output NDF. The value given should be one of the following: _DOUBLE, _REAL, _INTEGER, _INT64, _WORD, _UWORD, _BYTE or _UBYTE (note the leading underscore). If a null (!) value is supplied, the output data type equals the input data type. [!]

Examples:

```
substitute aa bb 1 0
```

This copies the NDF called aa to the NDF bb, except that any pixels with value 1 in aa are altered to have value 0 in bb.

```
substitute aa bb oldval=1 newval=0 comp=v
```

As above except the substitution occurs to the variance values.

```
substitute in=saturn out=saturn5 oldval=2.5 newval=5 comp=All
```

This copies the NDF called saturn to the NDF saturn5, except that any elements in the data and variance arrays that have value 2.5 are altered to have value 5 in saturn5.

Notes:

- The comparison for floating-point values tests that the difference between the replacement value and the element value is less than their mean times the precision of the data type.

Related Applications :

KAPPA: CHPIX, FILLBAD, GLITCH, NOMAGIC, SEGMENT, SETMAGIC, ZAPLIN; FIGARO: GOODVAR.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

SURFIT
Fits a polynomial or bi-cubic spline surface to two-dimensional data array

Description:

The background of a two-dimensional data array in the supplied NDF structure is estimated by condensing the array into equally sized rectangular bins, fitting a spline or polynomial surface to the bin values, and finally evaluating the surface for each pixel in the data array.

There is a selection of estimators by which representative values for each bin are determined. There are several options to make the fit more accurate. Values beyond upper and lower thresholds may be excluded from the binning. Bad pixels are also excluded, so prior masking may help to find the background more rapidly. κ-sigma clipping of the fitted bins is available so that the fit is not biased by anomalous bins, such as those entirely within an extended object. If a given bin contains more than a prescribed fraction of bad pixels, it is excluded from the fit.

The data array representing the background is evaluated at each pixel by one of two methods. It is written to the output NDF structure.

The raw binned data, the weights, the fitted binned data and the residuals to the fit may be written to a logfile. This also keeps a record of the input parameters and the rms error of the fit.

Usage:

```
surfit in out [fittype] [estimator] [bindim] [evaluate]
```

Parameters:

BINDIM() = _INTEGER (Read)

The *x*-*y* dimensions of a bin used to estimate the local background. If you supply only one value, it is used for both dimensions. The minimum value is 2. The maximum may be constrained by the number of polynomial terms, such that in each direction there are at least as many bins as terms. If a null (`!`) value is supplied, the value used is such that 32 bins are created along each axis. `[!]`

CLIP() = _REAL (Read)

Array of limits for progressive clipping of pixel values during the binning process in units of standard deviation. A null value means only unclipped statistics are computed and presented. Between one and five values may be supplied. `[2,3]`

ESTIMATOR = LITERAL (Read)

The estimator for the bin. It must be one of the following values: `"Mean"` for the mean value, `"Ksigma"` for the mean with κ-sigma clipping; `"Mode"` for the mode, and `"Median"` for the median. `"Mode"` is only available when there are at least twelve pixels in a bin. If a null (`!`) value is supplied, `"Median"` is used if there are fewer than 6 values in a bin, and `"Mode"` is used otherwise. `[!]`

EVALUATE = LITERAL (Read)
The method by which the resulting data array is to be evaluated from the surface-fit. It must be either "Interpolate" where the values at the corners of the bins are derived first, and then the pixel values are found by linear interpolation within those bins; or "All" where the surface-fit is evaluated for every pixel. The latter is slower, but can produce more-accurate results, unless the surface is well behaved. The default is the current value, which is initially set to "Interpolate". []

FITCLIP() = _REAL (Read)
Array of limits for progressive clipping of the binned array in units of the rms deviation of the fit. A null value (!) means no clipping of the binned array will take place. Between 1 and 5 values may be supplied. The default is the current value, which is ! initially. []

FITTYPE = LITERAL (Read)
The type of fit. It must be either "Polynomial" for a Chebyshev polynomial or "Spline" for a bi-cubic spline. The default is the current value, which initially is "Spline". []

GENVAR = _LOGICAL (Read)
If TRUE, a constant variance array is created in the output NDF assigned to the mean square surface-fit error. [FALSE]

LOGFILE = FILENAME (Read)
Name of the file to log the binned array and errors before and after fitting. If null, there will be no logging. [!]

IN = NDF (Read)
NDF containing the two-dimensional data array to be fitted.

KNOTS(2) = _INTEGER (Read)
The number of interior knots used for the bi-cubic-spline fit along the *x* and *y* axes. These knots are equally spaced within the image. Both values must be in the range 0 to 11. If you supply a single value, it applies to both axes. Thus 1 creates one interior knot, [5,4] gives 5 along the *x* axis and 4 along the *y* direction. Increasing this parameter's values increases the flexibility of the surface. Normally, 4 is a reasonable value. The upper limit of acceptable values will be reduced along each axis when its binned array dimension is fewer than 29. KNOTS is only accessed when FITTYPE="Spline". The default is the current value, which is 4 initially. []

ORDER(2) = _INTEGER (Read)
The orders of the fits along the *x* and *y* directions. Both values must be in the range 0 to 14. If you supply a single single value, it applies to both axes. Thus 0 gives a constant, [3,1] gives a cubic along the *x* direction and a linear fit along the *y* axis. Increasing this parameter's values increases the flexibility of the surface. The upper limit of acceptable values will be reduced along each axis when its binned array dimension is fewer than 29. ORDER is only accessed when FITTYPE="Polynomial". The default is the current value, which is 4 initially. []

OUT = NDF (Write)
NDF to contain the fitted two-dimensional data array.

THRHI = _REAL (Read)
Upper threshold above which values will be excluded from the analysis to derive

representative values for the bins. If it is null (!) there will be no upper threshold. [!]

THRLO = _REAL (Read)
Lower threshold below which values will be excluded from the analysis to derive representative values for the bins. If it is null (!) there will be no lower threshold. [!]

TITLE = LITERAL (Read)
The title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. [!]

WLIM = _REAL (Read)
The minimum fraction of good pixels in a bin that permits the bin to be included in the fit. Here good pixels are ones that participated in the calculation of the bin's representative value. So they exclude both bad pixels and ones rejected during estimation (*e.g.* ones beyond the thresholds or were clipped). [!]

Results Parameters:

RMS = _REAL (Write)
The RMS deviation of the fit from the original data (per pixel).

Notes:

A polynomial surface fit is stored in a SURFACEFIT extension, component FIT of type POLYNOMIAL, variant CHEBYSHEV or BSPLINE.

For further details of the CHEBYSHEV variant see SGP/38. The CHEBYSHEV variant includes the fitting variance for each coefficient.

The BSPLINE variant structure is provisional. It contain the spline coefficients in the two-dimensional DATA_ARRAY component, the knots in XKNOTS and YKNOTS arrays, and a scaling factor to restore the original values in SCALE. All of these components have type _REAL.

Also stored in the SURFACEFIT extension is the r.m.s. deviation to the fit (component RMS); and the co-ordinate system component COSYS, set to "GRID".

Examples:

```
surfit comaB comaB_bg
```

This calculates the surface fit to the two-dimensional NDF called comaB using the current defaults. The evaluated fit is stored in the NDF called comaB_bg.

```
surfit comaB comaB_bg poly median order=5 bindim=[24,30]
```

As above except that 5th-order polynomial fit is chosen, the median is used to derive the representative value for each bin, and the binning size is 24 pixels along the first axis, and 32 pixels along the second.

```
surfit comaB comaB_bg fitclip=[2,3] logfile=comaB_fit.lis
```

As the first example except that the binned array is clipped at 2 then 3 standard deviations to remove outliers before the final fit is computed. The text file comaB_fit.lis records a log of the surface fit.

```
surfit comaB comaB_bg estimator=ksigma clip=[2,2,3]
```

As the first example except that the representative value of each bin is the mean after clipping twice at 2 then once at 3 standard deviations.

```
surfit in=irasorion out=sback evaluate=all fittype=s knots=7
```

This calculates the surface fit to the two-dimensional NDF called irasorion. The fit is evaluated at every pixel and the resulting array stored in the NDF called sback. A spline with seven knots along each axis is used to fit the surface.

Related Applications :

KAPPA: ARDMASK, FITSURFACE, MAKESURFACE, REGIONMASK.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF. Any input VARIANCE is ignored.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single- or double-precision floating point for FITTYPE="Spline" or "Polynomial" respectively. The output NDF's DATA and VARIANCE components have type _REAL (single-precision).

THRESH
Edits an NDF to replace values between or outside given limits with specified constant values

Description:

This application creates an output NDF by copying values from an input NDF, replacing all values within given data ranges by a user-specified constant or by the bad value. Upper and lower thresholds are supplied using Parameters THRLO and THRHI.

If THRLO is less than or equal to THRHI, values between and including the two thresholds are copied from the input to output array. Any values in the input array greater than the upper threshold will be set to the value of Parameter NEWHI, and anything less than the lower threshold will be set to the value of Parameter NEWLO, in the output data array. Thus the output NDF is constrained to lie between the two bounds.

If THRLO is greater than THRHI, values greater than or equal to THRLO are copied from the input to output array, together with values less than or equal to THRHI. Any values between THRLO and THRHI will be set to the value of Parameter NEWLO in the output NDF.

Each replacement value may be the bad-pixel value for masking.

Usage:

```
thresh in out thrlo thrhi newlo newhi [comp]
```

Parameters:

COMP = LITERAL (Read)

The components whose values are to be constrained between thresholds. The options are limited to the arrays within the supplied NDF. In general the value may be `"Data"`, `"Quality"`, `"Error"`, or `"Variance"`. If `"Quality"` is specified, then the quality values are treated as numerical values in the range 0 to 255. `["Data"]`

IN = NDF (Read)

Input NDF structure containing the array to have thresholds applied.

NEWHI = LITERAL (Read)

This gives the value to which all input array-element values greater than the upper threshold are set. If this is set to `"Bad"`, the bad value is substituted. Numerical values of NEWHI must lie in within the minimum and maximum values of the data type of the array being processed. The suggested default is the upper threshold. This parameter is ignored if THRLO is greater than THRHI.

NEWLO = LITERAL (Read)

This gives the value to which all input array-element values less than the lower threshold are set. If this is set to `"Bad"`, the bad value is substituted. Numerical values of NEWLO must lie in within the minimum and maximum values of the data type of the array being processed. The suggested default is the lower threshold.

OUT = NDF (Write)

Output NDF structure containing the thresholded version of the array.

THRHI = _DOUBLE (Read)
The upper threshold value within the input array. It must lie in within the minimum and maximum values of the data type of the array being processed. The suggested default is the current value.

THRLO = _DOUBLE (Read)
The lower threshold value within the input array. It must lie within the minimum and maximum values of the data type of the array being processed. The suggested default is the current value.

TITLE = LITERAL (Read)
Title for the output NDF structure. A null value (!) propagates the title from the input NDF to the output NDF. [!]

Results Parameters:

NUMHI = _INTEGER (Write)
The number of pixels whose values were thresholded as being greater than the THRHI threshold.

NUMLO = _INTEGER (Write)
The number of pixels whose values were thresholded as being less than the THRLO threshold.

NUMRANGE = _INTEGER (Write)
The number of pixels whose values were thresholded as being between the THRLO and THRHI thresholds, if THRLO is greater than THRHI.

NUMSAME = _INTEGER (Write)
The number of unchanged pixels.

Examples:

```
thresh zzcam zzcam2 100 500 0 0
```

This copies the data array in the NDF called zzcam to the NDF called zzcam2. Any data value less than 100 or greater than 500 in zzcam is set to 0 in zzcam2.

```
thresh zzcam zzcam2 500 100 0
```

This copies the data array in the NDF called zzcam to the NDF called zzcam2. Any data value less than 500 and greater than 100 in zzcam is set to 0 in zzcam2.

```
thresh zzcam zzcam2 100 500 0 0 comp=Variance
```

As above except that the data array is copied unchanged and the thresholds apply to the variance array.

```
thresh n253 n253cl thrlo=-0.5 thrhi=10.1 \
```

This copies the data array in the NDF called n253 to the NDF called n253cl. Any data value less than -0.5 in n253 is set to -0.5 in n253cl, and any value greater than 10.1 in n253 becomes 10.1 in n253cl.

```
thresh pavo pavosky -0.02 0.02 bad bad
```

All data values outside the range -0.02 to 0.02 in the NDF called pavo become bad in the NDF called pavosky. All values within this range are copied from pavo to pavosky.

Related Applications :

KAPPA: HISTEQ, MATHS; FIGARO: CLIP, IDIFF, RESCALE.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.
- Any number of NDF dimensions is supported.

TRANDAT
Converts free-format text data into an NDF

Description:

This application takes grid data contained in a free-format text file and stores them in the data array of an NDF . The data file could contain, for example, mapping data or results from simulations which are to be converted into an image for analysis.

There are two modes of operation which depend on whether the text file contains co-ordinate information, or solely data values (determined by Parameter AUTO).

a) **AUTO=FALSE** If the file contains co-ordinate information the format of the data is tabular; the positions and values are arranged in columns and a record may contain information for only a single point. Where data points are duplicated only the last value appears in the NDF. Comment lines can be given, and are indicated by a hash or exclamation mark in the first column. Here is an example file (the vertical ellipses indicate missing lines in the file):

```
# Model 5, phi = 0.25, eta = 1.7
1 -40.0 40.0 1121.9
2 0.0 30.0 56.3
3 100.0 20.0 2983.2
4 120.0 85.0 339.3
.   .    .     .
.   .    .     .
.   .    .     .
<EOF>
```

The records do not need to be ordered (but see the warning in the "Notes"), as the application searches for the maximum and minimum co-ordinates in each dimension so that it can define the size of the output image. Also, each record may contain other data fields (separated by one or more spaces), which need not be all the same data type. In the example above only columns 2, 3 and 4 are required. There are parameters (POSCOLS, VALCOL) which select the co-ordinate and value columns.

The distance between adjacent pixels (given by Parameter PSCALE) defaults to 1, and is in the same units as the read-in co-ordinates. The pixel index of a data value is calculated using the expression

$$\text{index} = \text{FLOOR}((x - xoff)/\text{scale}) + 1$$

where x is the supplied co-ordinate and $xoff$ is the value of the POFFSET parameter (which defaults to the minimum supplied co-ordinate along an axis), *scale* is the value of Parameter PSCALE, and FLOOR is a function that returns the largest integer that is smaller (*i.e.* more negative) than its argument.

You are informed of the number of points found and the maximum and minimum co-ordinate values for each dimension. There is no limit imposed by the application on

the number of points or the maximum output array size, though there may be external constraints. The derived array size is reported in case you have made a typing error in the text file. If you realise that this has indeed occurred just abort (`!!`) when prompted for the output NDF.

b) **AUTO=TRUE** If the text file contains no co-ordinates, the format is quite flexible, however, the data are read into the data array in Fortran order, *i.e.* the first dimension is the most rapidly varying, followed by the second dimension and so on. The number of data values that may appear on a line is variable; data values are separated by at least a space, comma, tab or carriage return. A line can have up to 255 characters. In addition a record may have trailing comments designated by a hash or exclamation mark. Here is an example file, though a more regular format would be clearer for the human reader.

```
# test for the new TRANDAT
23 45.3 !  a comment
50.7,47.5 120.  46.67 47.89 42.4567
.1 23.3 45.2 43.2 56.0 30.9 29.  27.  26.  22.4 20.  18.  -12.  8.
9.2 11.
<EOF>
```

Notice that the shape of the NDF is defined by a parameter rather than explicitly in the file.

Usage:

```
trandat freename out [poscols] [valcol] [pscale] [dtype] [title]
```

Parameters:

AUTO = _LOGICAL (Read)
If `TRUE` the text file does not contain co-ordinate information. `[FALSE]`

BAD = _LOGICAL (Read)
If `TRUE` the output NDF data array is initialised with the bad value, otherwise it is filled with zeroes. `[TRUE]`

DTYPE = LITERAL (Read)
The HDS type of the data values within the text file, and the type of the data array in the output NDF. The options are: `'_REAL'`, `'_DOUBLE'`, `'_INTEGER'`, `'_BYTE'`, `'_UBYTE'`, `'_WORD'`, `'_UWORD'`. (Note the leading underscore.) `['_REAL']`

FREENAME = FILENAME (Read)
Name of the text file containing the free-format data.

LBOUND( ) = _INTEGER (Read)
The lower bounds of the NDF to be created. The number of values must match the number supplied to Parameter SHAPE. It is only accessed in automatic mode. If a null (`!`) value is supplied, the value used is 1 along each axis. `[!]`

POFFSET() = _REAL (Read)
The supplied co-ordinates that correspond to the origin of floating point pixel co-ordinates. It is only used in co-ordinate mode. Its purpose is to permit an offset from some arbitrary units to pixels. If a null (`!`) value is supplied, the value used is the minimum supplied co-ordinate value for each dimension. `[!]`

POSCOLS() = _INTEGER (Read)
Column positions of the co-ordinates in an input record of the text file, starting from x to higher dimensions. It is only used in co-ordinate mode. The columns must be different amongst themselves and also different from the column containing the values. If there is duplication, new values for both POSCOLS and VALCOL will be requested. [1,2]

PSCALE() = _REAL (Read)
Pixel-to-pixel distance in co-ordinate units for each dimension. It is only used in co-ordinate mode. Its purpose is to permit linear scaling from some arbitrary units to pixels. [1.0 in each co-ordinate dimension]

QUANTUM = _INTEGER (Read)
You can safely ignore this parameter. It is used for fine-tuning performance in the co-ordinate mode.

The application obtains work space to store the position-value data before they can be copied into the output NDF so that the array bounds can be computed. Since the number of lines in the text file is unknown, the application obtains chunks of work space whose size is three times this parameter whenever it runs out of storage. (Three because the parameter specifies the number of lines in the file rather than the number of data items.) If you have a large number of points there are efficiency gains if you make this parameter either about 20–30 per cent or slightly greater than or equal to the number of lines your text file. A value slightly less than the number of lines is inefficient as it creates nearly 50 per cent unused space. A value that is too small can cause unnecessary unmapping, expansion and re-mapping of the work space. For most purposes the default should give acceptable performance. It must lie between 32 and 2097152. [2048]

SHAPE( ) = _INTEGER (Read)
The shape of the NDF to be created. For example, [50,30,20] would create 50 columns by 30 lines by 10 bands. It is only accessed in automatic mode.

NDF = NDF (Write)
Output NDF for the generated data array.

TITLE = LITERAL (Read)
Title for the output NDF. ["KAPPA - Trandat"]

VALCOL = _INTEGER (Read)
Column position of the array values in an input record of the text file. It is only used in co-ordinate mode. The column position must be different from those specified for the co-ordinate columns. If there is duplication, new values for both POSCOLS and VALCOL will be requested. [3]

Examples:

```
trandat simdata.dat model
```

Reads the text file simdata.dat and stores the data into the data array of a two-dimensional, _REAL NDF called model. The input file should have the co-ordinates and real values arranged in columns, with the x-y positions in columns 1 and 2 respectively, and the real data in column 3.

```
trandat freename=simdata out=model auto shape=[50,40,9]
```

Reads the text file `simdata` and stores the data into the data array of a three-dimensional, _REAL NDF called model. Its x dimension is 50, y is 40 and z is 9. The input file only contains real values and comments.

```
trandat freename=simdata out=model auto shape=[50,40,9] dtype=_i
```

As the previous example except an _INTEGER NDF is created, and the text file must contain integer data.

```
trandat simdata.dat model [6,3,4] 2
```

Reads the text file `simdata.dat` and stores the data into the data array of a three-dimensional, _REAL NDF called model. The input file should have the co-ordinates and real values arranged in columns, with the x-y-z positions in columns 6, 3 and 4 respectively, and the real data in column 2.

```
trandat spectrum.dat lacertid noauto poscols=2 valcol=4 pscale=2.3
```

Reads the text file `spectrum.dat` and stores the data into the data array of a one-dimensional, _REAL NDF called lacertid. The input file should have the co-ordinate and real values arranged in columns, with its co-ordinates in columns 2, and the real data in column 4. A one-pixel step in the NDF corresponds to 2.3 in units of the supplied co-ordinates.

Notes:

- Bad data values may be represented by the string "BAD" (case insensitive) within the input text file.
- All non-complex numeric data types can be handled. However, byte, unsigned byte, word and unsigned word require data conversion, and therefore involve additional processing. to a vector element (for n-d generality).
- **WARNING:** In non-automatic mode it is strongly advisable for large output NDFs to place the data in Fortran order, *i.e.* the first dimension is the most rapidly varying, followed by the second dimension and so on. This gives optimum performance. The meaning of 'large' will depend on working-set quotas on your system, but a few megabytes gives an idea. If you jump randomly backwards and forwards, or worse, have a text file in reverse-Fortran order, this can have disastrous performance consequences for you and other users.
- In non-automatic mode, the co-ordinates for each dimension are stored in the NDF axis structure. The first centre is at the minimum value found in the list of positions for the dimension plus half of the scale factor. Subsequent centres are incremented by the scale factor.
- The output NDF may have between one and seven dimensions.
- In automatic mode, an error is reported if the shape does not use all the data points in the file.

Related Applications :

CONVERT: ASCII2NDF, NDF2ASCII; FIGARO: ASCIN, ASCOUT.

TRIG
Performs a trigonometric transformation on a NDF

Description:

This routine copies the supplied input NDF , performing a specified trigonometric operation (sine, tangent, *etc.*) on each value in the DATA array. The VARIANCE component, if present, is modified appropriately. Pixels for which the required value is undefined, or outside the numerical range of the NDFs data type, are set bad in the output.

Usage:

```
trig in trigfunc out title
```

Parameters:

IN = NDF (Read)
The input NDF structure.

OUT = NDF (Write)
The output NDF structure.

TRIGFUNC = LITERAL (Read)
Trigonometrical function to be applied. The options are as follows.

- `"ACOS"` — arc-cosine (radians)
- `"ACOSD"` — arc-cosine (degrees)
- `"ASIN"` — arc-sine (radians)
- `"ASIND"` — arc-sine (degrees)
- `"ATAN"` — arc-tangent (radians)
- `"ATAND"` — arc-tangent (degrees)
- `"COS"` — cosine (radians)
- `"COSD"` — cosine (degrees)
- `"SIN"` — sine (radians)
- `"SIND"` — sine (degrees)
- `"TAN"` — tangent (radians)
- `"TAND"` — tangent (degrees)

TITLE = LITERAL (Read)
A title for the output NDF. A null value will cause the title of the NDF supplied for Parameter IN to be used instead. `[!]`

Examples:

```
trig sindata asind data
```

Take the arc-sine of the data values in the NDF called sindata, and store the results (in degrees) in the NDF called data.

```
trig sindata asin data
```

As above, but the output values are stored in radians.

Related Applications :

KAPPA: ADD, CADD, CMULT, CDIV, CSUB, DIV, MATHS, MULT, SUB.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, HISTORY, WCS, and VARIANCE components of an NDF data structure and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single-precision floating point, or double precision, if appropriate, but the numeric type of the input pixels is preserved in the output NDF.

VECPLOT
Plots a two-dimensional vector map

Description:

This application plots vectors defined by the values contained within a pair of two-dimensional NDFs , the first holding the magnitude of the vector quantity at each pixel, and the second holding the corresponding vector orientations. It is assumed that the two NDFs are aligned in pixel co-ordinates. The number of vectors in the plot is kept to a manageable value by only plotting vectors for pixels on a sparse regular matrix. The increment (in pixels) between plotted vectors is given by Parameter STEP. Zero orientation may be fixed at any position angle within the plot by specifying an appropriate value for Parameter ANGROT. Each vector may be represented either by an arrow or by a simple line, as selected by Parameter ARROW.

The plot is produced within the current graphics database picture, and may be aligned with an existing DATA picture if the existing picture contains suitable co-ordinate Frame information (see Parameter CLEAR).

Annotated axes can be produced (see Parameter AXES), and the appearance of these can be controlled in detail using Parameter STYLE. The axes show co-ordinates in the current co-ordinate Frame of NDF1.

A key to the vector scale can be displayed to the right of the vector map (see Parameter KEY). The appearance and position of this key may be controlled using Parameters KEYSTYLE and KEYPOS.

Usage:

```
vecplot ndf1 ndf2 [comp] [step] [vscale] [arrow] [just] [device]
```

Parameters:

ANGROT = _REAL (Read)

A rotation angle in degrees to be added to each vector orientation before plotting the vectors (see Parameter NDF2). It should be in the range 0–360. [0.0]

ARROW = LITERAL (Read)

Vectors are drawn as arrows, with the size of the arrow head specified by this parameter. Simple lines can be drawn by setting the arrow head size to zero. The value should be expressed as a fraction of the largest dimension of the vector map. [current value]

AXES = _LOGICAL (Read)

TRUE if labelled and annotated axes are to be drawn around the vector map. These display co-ordinates in the current co-ordinate Frame NDF1, which may be changed using application WCSFRAME (see also Parameter USEAXIS). The width of the margins left for the annotation may be controlled using Parameter MARGIN. The appearance of the axes (colours, founts, *etc.*) can be controlled using the STYLE parameter. [TRUE]

CLEAR = _LOGICAL (Read)

TRUE if the graphics device is to be cleared before displaying the vector map. If you

want the vector map to be drawn over the top of an existing DATA picture, then set CLEAR to FALSE. The vector map will then be drawn in alignment with the displayed data. If possible, alignment occurs within the current co-ordinate Frame of the NDF. If this is not possible (for instance, if suitable WCS information was not stored with the existing DATA picture), then alignment is attempted in PIXEL co-ordinates. If this is not possible, then alignment is attempted in GRID co-ordinates. If this is not possible, then alignment is attempted in the first suitable Frame found in the NDF irrespective of its domain. A message is displayed indicating the domain in which alignment occurred. If there are no suitable Frames in the NDF then an error is reported. [TRUE]

COMP = LITERAL (Read)

The component of NDF1 which is to be used to define the vector magnitudes. It may be "Data", "Error" or "Variance". The last two are not available if NDF1 does not contain a VARIANCE component. The vector orientations are always defined by the "Data" component of NDF2. ["Data"]

DEVICE = DEVICE (Read)

The plotting device. [Current graphics device]

FILL = _LOGICAL (Read)

The DATA picture containing the vector map is usually produced with the same shape as the data. However, for maps with markedly different dimensions this default behaviour may not give the clearest result. When FILL is TRUE, the smaller dimension of the picture is expanded to produce the largest possible picture within the current picture. [FALSE]

JUST = LITERAL (Read)

The justification for each vector; it can take any of the following values:

- "Centre" — the vectors are drawn centred on the corresponding pixel,
- "Start" — the vectors are drawn starting at the corresponding pixel, and
- "End" — the vectors are drawn ending at the corresponding pixel.

["Centre"]

KEY = _LOGICAL (Read)

TRUE if a key indicating the vector scale is to be produced. [TRUE]

KEYPOS() = _REAL (Read)

Two values giving the position of the key. The first value gives the gap between the right-hand edge of the vector map and the left-hand edge of the key (0.0 for no gap, 1.0 for the largest gap). The second value gives the vertical position of the top of the key (1.0 for the highest position, 0.0 for the lowest). If the second value is not given, the top of the key is placed level with the top of the vector map. Both values should be in the range 0.0 to 1.0. If a key is produced, then the right-hand margin specified by Parameter MARGIN is ignored. [current value]

KEYSTYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the key (see Parameter KEY).

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and

interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the text in the key is controlled using String attributes (*e.g.* Colour(Strings), Font(Strings); the synonym Text can be used in place of Strings). Note, the Size attribute specifies the size of key text relative to the size of the numerical labels on the vector-map axes. Thus a value of 2.0 for Size will result in text which is twice the size of the numerical axis labels. The appearance of the example vector is controlled using Curve attributes (*e.g.* Colour(Curves); the synonym Vector can be used in place of Curves). The numerical scale value is formatted as an axis-1 value (using attributes Format(1), Digits(1), *etc*; the synonym Scale can be used in place of the value 1). The length of the example vector is formatted as an axis-2 value (using attribute Format(2), *etc*; the synonym Vector can be used in place of the value 2). The vertical space between lines in the key can be controlled using attribute TextLabGap. A value of 1.0 is used if no value is set for this attribute, and produces default vertical spacing. Values larger than 1.0 increase the vertical space, and values less than 1.0 decrease the vertical space. [current value]

KEYVEC = _REAL (Read)

Length of the vector to be displayed in the key, in data units. If a null (!) value is supplied, the value used is generated on the basis of the spread of vector lengths in the plot. [!]

MARGIN(4) = _REAL (Read)

The widths of the margins to leave around the vector map for axis annotation. The widths should be given as fractions of the corresponding dimension of the current picture. The actual margins used may be increased to preserve the aspect ratio of the DATA 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. If a null (!) value is supplied, the value used is 0.15 (for all edges) if annotated axes are being produced, and zero otherwise. See also Parameter KEYPOS. [current value]

NDF1 = NDF (Read)

NDF structure containing the two-dimensional image giving the vector magnitudes.

NDF2 = NDF (Read)

NDF structure containing the two-dimensional image giving the vector orientations. The values are considered to be in units of degrees unless the UNITS component of

the NDF has the value "Radians" (case insensitive). The positive *y* pixel axis defines zero orientation, and rotation from the *x* pixel axis to the *y* pixel is considered positive.

STEP = _INTEGER (Read)

The number of pixels between adjacent displayed vectors (along both axes). Increasing this value reduces the number of displayed vectors. If a null (!) value is supplied, the value used gives about thirty vectors along the longest axis of the plot. [!]

STYLE = GROUP (Read)

A group of attribute settings describing the plotting style to use for the vectors and annotated axes.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of the vectors is controlled by the attributes Colour(Curves), Width(Curves), *etc.* (the synonym Vectors may be used in place of Curves). [current value]

VSCALE = _REAL (Read)

The scale to be used for the vectors. The supplied value should give the data value corresponding to a vector length of one centimetre. If a null (!) value is supplied, a default value is used. [!]

USEAXIS = GROUP (Read)

USEAXIS is only accessed if the current co-ordinate Frame of the NDF has more than two axes. A group of two strings should be supplied specifying the two axes which are to be used when annotating and aligning the vector map. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the input NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (!) value is supplied, the axes with the same indices as the two significant NDF pixel axes are used. [!]

Examples:

```
vecplot polint polang
```

Produces a vector map on the current graphics device with vector magnitude taken from the NDF called polint and vector orientation taken from NDF polang. All other settings are defaulted, so for example about 20 vectors are displayed along the longest axis, and a key is plotted.

```
vecplot polint polang angrot=23.4 clear=no
```

Produces a vector map in which the primary axis of the vectors (as defined by the value zero in the NDF polang) is at the position angle 23.4 degrees (measured anti-clockwise from the positive *y* axis) in the displayed map. The map is drawn over the top of the previously drawn DATA picture, aligned in a suitable co-ordinate Frame.

```
vecplot stack(,,2) stack(,,1) arrow=0.1 just=start nokey
```

Produces a vector map in which the vectors are defined by two planes in the three-dimensional NDF called stack. There is no need to copy the two planes into two separate NDFs before running VECPLOT. Each vector is represented by an arrow, starting at the position of the corresponding pixel. No key to the vector scale and justification is produced.

Notes:

- If no Title is specified via the STYLE parameter, then the TITLE component in NDF1 is used as the default title for the annotated axes. Should the NDF not have a TITLE component, then the default title is instead taken from current co-ordinate Frame in NDF1, unless this attribute has not been set explicitly, whereupon the name of NDF1 is used as the default title.
- The application stores a number of pictures in the graphics database in the following order: a FRAME picture containing the annotated axes, vectors, and key; a KEY picture to store the key if present; and a DATA picture containing just the vectors. Note, the FRAME picture is only created if annotated axes or a key has been drawn, or if non-zero margins were specified using Parameter MARGIN. The world co-ordinates in the DATA picture will be pixel co-ordinates. A reference to NDF1, together with a copy of the WCS information in the NDF are stored in the DATA picture. On exit the current database picture for the chosen device reverts to the input picture.

Related Applications :

KAPPA: CALPOL.

Implementation Status:

- Only real data can be processed directly. Other non-complex numeric data types will undergo a type conversion before the vector plot is drawn.
- Bad pixels and quality masking are supported.

WCSADD
Creates a Mapping and optionally adds a new co-ordinate Frame into the WCS component of an NDF

Description:

This application can be used to create a new AST Mapping and optionally use the Mapping to add a new co-ordinate Frame into the WCS component of an NDF (see Parameter NDF). An output text file may also be created holding a textual representation of the Mapping for future use by other applications such as REGRID (see Parameter MAPOUT). A number of different types of Mapping can be used (see Parameter MAPTYPE).

When adding a new Frame to a WCS component, the Mapping is used to connect the new Frame to an existing one (called the *basis* Frame: see Parameter FRAME). The specific type of Frame to add is specified using Parameter FRMTYPE (the default is to simply copy the basis Frame). Optionally (see Parameter TRANSFER), attributes which have been assigned an explicit value in the basis Frame are transferred to the new Frame (but only if they are relevant to the type of the new Frame). The value of the Domain attribute for the new Frame can be specified using Parameter DOMAIN. Other attribute values for the new Frame may be specified using Parameter ATTRS. The new Frame becomes the current co-ordinate Frame in the NDF (unless Parameter RETAIN is set TRUE).

WCSADD will only generate Mappings with the same number of input and output axes; this number is determined by the number of axes in the basis Frame if an NDF is supplied, or by the NAXES parameter otherwise.

Usage:

```
wcsadd ndf frame domain maptype
```

Parameters:

ATTRS = GROUP (Read)

A group of attribute settings to be applied to the new Frame before adding it into the NDF.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character "^". Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of an attribute appropriate to the type of Frame specified by Parameter FRMTYPE (see SUN/210 for a complete description of all attributes), and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes—these defaults are inherited from the basis Frame. Any unrecognised attributes are ignored (no error is reported).

CENTRE(2) = _DOUBLE (Read)
The co-ordinates of the centre of a pincushion distortion. It is only used when MAPTYPE="PINCUSHION". See also DISCO. [0,0]

DIAG( ) = _DOUBLE (Read)
The elements along the diagonal of the linear transformation matrix. There will be as many of these as there are axes in the basis Frame. Each effectively gives the factor by which co-ordinates on the corresponding axis should be multiplied. This parameter is only used when MAPTYPE="DIAG".

DISCO = _DOUBLE (Read)
The distortion coefficient of a pincushion distortion. Used in conjunction with the CENTRE parameter, this defines the forward transformation to be used as follows:

$$XX = X + D * (X - C1) * ((X - C1) * *2 + (Y - C2) * *2)$$
$$YY = Y + D * (Y - C2) * ((X - C1) * *2 + (Y - C2) * *2)$$

where (X,Y) are the input co-ordinates, (XX,YY) the output co-ordinates, D is DISCO, and $C1$ and $C2$ are the two elements of CENTRE. DISCO is only used when MAPTYPE="PINCUSHION".

DOMAIN = LITERAL (Read)
The value for the Domain attribute for the new Frame. Care should be taken to ensure that domain names are used consistently. This will usually mean avoiding any domain names that are already in use within the WCS component, particularly the standard domain names such as GRID, FRACTION, PIXEL, AXIS, and GRAPHICS. The supplied value is stripped of spaces, and converted to upper case before being used.

Note, if Parameter MAPTYPE is set to "REFNDF", then the value supplied for Parameter DOMAIN indicates the Domain of the Frame within the reference NDF that is to be copied (see Parameter REFNDF).

EPOCH = _DOUBLE (Read)
If the basis Frame is specified using a 'Sky Co-ordinate System' specification for a celestial co-ordinate system (see Parameter FRAME), 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. The suggested default is the value stored in the basis Frame.

FOREXP = LITERAL (Read)
A group of expressions to be used for the forward co-ordinate transformations in a MathMap. There must be at least as many expressions as the number of axes of the Mapping, but there may be more if intermediate expressions are to be used. The expressions may be given directly in response to the prompt, or read from a text file, in which case the name of the file should be given, preceded by a "^" character. Individual expression should be separated by commas or, if they are supplied in a file, newlines (see Section 4.13).

The syntax for each expression is Fortran-like; see the "Examples" section below, and Appendix G for details. FOREXP is only used when MAPTYPE="MATH".

FRAME = LITERAL (Read)
A string specifying the basis Frame. If a null value is supplied the current co-ordinate Frame in the NDF is used. The string can be one of the following:

- A domain name such as SKY, AXIS, PIXEL. The two "pseudo-domains" WORLD and DATA may be supplied and will be translated into PIXEL and AXIS respectively, so long as the WCS component of the NDF does not contain Frames with these domains.
- 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).

FRMTYPE = LITERAL (Read)
The type of Frame to add to the NDF. If a null (!) value is supplied, a copy of the basis Frame is used (as modified by Parameters ATTRS and DOMAIN). The allowed values are as follows.

- "FRAME" — A simple Cartesian Frame (the number of axes is equal to the number of outputs from the Mapping)
- "SKYFRAME" — A two-dimensional Frame representing positions on the celestial sphere.
- "SPECFRAME" — A one-dimensional Frame representing positions within an electromagnetic spectrum.
- "TIMEFRAME" — A one-dimensional Frame representing moments in time.

Note, if Parameter MAPTYPE is set to "REFNDF", then Parameter FRMTYPE will not be used—the Frame used will instead always be a copy of the Frame from the reference NDF (as selected by Parameter DOMAIN). [!]

INVEXP = LITERAL (Read)
The expressions to be used for the inverse co-ordinate transformations in a MathMap. See FOREXP. INVEXP is only used when MAPTYPE="MATH".

MAPIN = FILENAME (Read)
The name of a file containing an AST Mapping with which to connect the basis Frame to the new one. The file may be a text file which contains the textual representation of an AST Mapping, or a FITS file which contains the Mapping as an AST object encoded in its headers, or an NDF. If it is an NDF, the Mapping from its base (GRID-domain) to current Frame will be used. Only used when MAPTYPE="FILE".

MAPOUT = FILENAME (Write)
The name of a text file in which to store a textual representation of the Mapping. This can be used, for instance, by the REGRID application. If a null (!) value is supplied, no file is created. [!]

MAPTYPE = LITERAL (Read)
The type of Mapping to be used to connect the new Frame to the basis Frame. It must be one of the following strings, each of which require some additional parameters as indicated.

- "DIAGONAL" — A linear mapping with no translation of off-diagonal coefficients (see Parameter DIAG)
- "FILE" — A mapping defined by an AST Mapping supplied in a separate file (see Parameter MAPIN)
- "LINEAR" — A general linear mapping (see Parameter TR)
- "MATH" — A general algebraically defined mapping (see Parameters FOREXP, INVEXP, SIMPFI, SIMPIF)
- "PINCUSHION" — A pincushion/barrel distortion (see Parameters DISCO and CENTRE)
- "REFNDF" — The Mapping is obtained by aligning the NDF with a second reference NDF (see Parameter REFNDF)
- "SHIFT" — A translation (see Parameter SHIFT)
- "UNIT" — A unit mapping
- "ZOOM" — A uniform expansion/contraction (see Parameter ZOOM)

["LINEAR"]

NAXES = _INTEGER (Read)
The number of input and output axes which the Mapping will have. Only used if a null value is supplied for Parameter NDF.

NDF = NDF (Read and Write)
The NDF in which to store a new co-ordinate Frame. Supply a null (!) value if you do not wish to add a Frame to an NDF (you can still use the MAPOUT parameter to write the Mapping to a text file).

REFNDF = NDF (Read)
A reference NDF from which to obtain the Mapping and Frame. The NDFs specified by Parameters NDF and REFNDF are aligned in a suitable co-ordinate system (usually their current Frames—an error is reported if the two NDFs cannot be aligned). The Mapping from the basis Frame in "NDF" (specified by Parameter FRAME) to the required Frame in "REFNDF" (specified by Parameter DOMAIN) is then found and used. The Frame added into "NDF" is always a copy of the reference Frame—regardless of the setting of Parameter FRMTYPE. Parameter REFNDF is only used when Parameter MAPTYPE is set to "REFNDF", in which case a value must also be supplied for Parameter NDF (an error will be reported otherwise).

RETAIN = _LOGICAL (Read)
Indicates whether the original current Frame should be retained within the WCS FrameSet of the modified NDF (see Parameter NDF). If FALSE, the newly added Frame is the current Frame on exit. Otherwise, the original current Frame is retained on exit. [FALSE]

SHIFT( ) = _DOUBLE (Read)
A vector giving the displacement represented by the translation. There must be one element for each axis. Only used when MAPTYPE="SHIFT".

SIMPFI = _LOGICAL (Read)
The value of the Mapping's SimpFI attribute (whether it is legitimate to simplify the forward followed by the inverse transformation to a unit transformation). This parameter is only used when MAPTYPE="MATH". [TRUE]

SIMPIF = _LOGICAL (Read)
The value of the Mapping's SimpIF attribute (whether it is legitimate to simplify the inverse followed by the forward transformation to a unit transformation). This parameter is only used when MAPTYPE="MATH". [TRUE]

TR( ) = _DOUBLE (Read)
The values of this parameter are the coefficients of a linear transformation from the basis Frame specified by Parameter FRAME to the new Frame. This parameter is only used when MAPTYPE="LINEAR". For instance, if a feature has co-ordinates (X,Y,Z,...) in the basis Frame, and co-ordinates (U,V,W,...) in the new Frame, then the following transformations would be used, depending on how many axes the two Frames have:

- one-dimensional:
$$U = TR(1) + TR(2) * X$$
- two-dimensional:
$$U = TR(1) + TR(2) * X + TR(3) * Y$$
$$V = TR(4) + TR(5) * X + TR(6) * Y$$
- three-dimensional:
$$U = TR(1) + TR(2) * X + TR(3) * Y + TR(4) * Z$$
$$V = TR(5) + TR(6) * X + TR(7) * Y + TR(8) * Z$$
$$W = TR(9) + TR(10) * X + TR(11) * Y + TR(12) * Z$$

The correct number of values must be supplied (that is, $N * (N + 1)$ where N is the number of axes in the new and old Frames). If a null value (!) is given it is assumed that the new Frame and the basis Frame are connected using a unit mapping (*i.e.* corresponding axis values are identical in the two Frames). This parameter is only used when MAPTYPE="LINEAR". [!]

TRANSFER = _LOGICAL (Read)
If TRUE, attributes which have explicitly set values in the basis Frame (specified by Parameter FRAME) are transferred to the new Frame (specified by Parameter FRMTYPE), if they are applicable to the new Frame. If FALSE, no attribute values are transferred. The dynamic default is TRUE if and only if the two Frames are of the same class and have the same value for their Domain attributes. []

ZOOM = _DOUBLE (Read)
The scaling factor for a ZoomMap; every co-ordinate will be multiplied by this factor in the forward transformation. ZOOM is only used when MAPTYPE="ZOOM".

Examples:

```
wcsadd speca axis frmtype=specframe maptype=unit
attrs="'system=wave,unit=Angstrom'"
```

This example assumes the NDF called speca has an Axis structure describing wavelength in Ångstroms. It adds a corresponding SpecFrame into the WCS component of the NDF. The SpecFrame is connected to the Frame describing the NDF Axis structure using a unit Mapping. Subsequently, WCSATTRIB can be used to modify the SpecFrame so that it describes the spectral-axis value in some other system (frequency, velocities of various forms, energy, wave number, *etc.*).

```
wcsadd ngc5128 pixel old_pixel unit
```

This adds a new co-ordinate Frame into the WCS component of the NDF called ngc5128. The new Frame is given the domain OLD_PIXEL and is a copy of the existing PIXEL Frame. This OLD_PIXEL Frame will be retained through further processing and can be used as a record of the original pixel co-ordinate Frame.

```
wcsadd my_data dist-lum dist(au)-lum linear tr=[0,2.0626E5,0,0,0,1]
```

This adds a new co-ordinate Frame into the WCS component of the NDF called my_data. The new Frame is given the domain DIST(AU)-LUM and is a copy of an existing Frame with domain DIST-LUM. The first axis in the new Frame is derived from the first axis in the basis Frame but is in different units (AU instead of parsecs). This change of units is achieved by multiplying the old Frame Axis 1 values by 2.0626E5. The values on the second axis are copied without change. You could then use application WCSATTRIB to set the Unit attribute for Axis 1 of the new Frame to "AU".

```
wcsadd my_data dist-lum dist(au)-lum diag diag=[2.0626E5,1]
```

This does exactly the same as the previous example.

```
wcsadd ax322 !  shrunk zoom zoom=0.25 mapout=zoom.ast
```

This adds a new Frame to the WCS component of ax322 which is a one-quarter-scale copy of its current co-ordinate Frame. The Mapping is also stored in the text file zoom.ast.

```
wcsadd cube grid slid shift shift=[0,0,1024]
```

This adds a new Frame to the WCS component of the NDF cube which matches the GRID-domain co-ordinates in the first two axes, but is translated by 1024 pixels on the third axis.

```
wcsadd plane pixel polar math simpif simpfi
forexp="'r=sqrt(x*x+y*y),theta=atan2(y,x)'"
invexp="'x=r*cos(theta),y=r*sin(theta)'"
```

A new Frame is added which gives pixel positions in polar co-ordinates. Fortran-like expressions are supplied which define both the forward and inverse transformations of the Mapping. The symbols *x* and *y* are used to represent the two input Cartesian pixel co-ordinate axes, and the symbols *r* and *theta* are used to represent the output polar co-ordinates. Note, the single quotes are needed when running from the UNIX shell in order to prevent the shell interpreting the parentheses and commas within the expressions.

```
wcsadd plane pixel polar math simpif simpfi forexp=^ft invexp=^it
```

As above, but the expressions defining the transformations are supplied in two text files called `ft` and `it`, instead of being supplied directly. Each file could contain the two expression on two separate lines.

```
wcsadd ndf=!  naxes=2 mapout=pcd.ast maptype=pincushion disco=5.3e-10
```

This constructs a pincushion-type distortion Mapping centred on the origin with a distortion coefficient of 5.3e-10, and writes out the Mapping as a text file called pcd.ast. This file could then be used by REGRID to resample the pixels of an NDF according to this transformation. No NDF is accessed.

```
wcsadd qmosaic frame=grid domain=polanal maptype=refndf refndf=imosaic
```

This adds a new co-ordinate Frame into the WCS component of the NDF called qmosaic. The new Frame has domain "POLANAL" and is copied from the NDF called imosaic (an error is reported if there is no such Frame with imosaic). The new co-ordinate Frame is attached to the base Frame (*i.e.* GRID co-ordinates) within qmosaic using a Mapping that produces alignment between qmosaic and imosaic.

Notes:

- The new Frame has the same number of axes as the basis Frame.
- An error is reported if the transformation supplied using Parameter TR is singular.

Related Applications :

KAPPA: NDFTRACE, REGRID, WCSATTRIB, WCSFRAME, WCSREMOVE; CCDPACK: WCSEDIT.

WCSALIGN
Aligns a group of NDFs using World Co-ordinate System information

Description:

This application resamples or rebins a group of input NDFs , producing corresponding output NDFs which are aligned pixel-for-pixel with a specified reference NDF, or POLPACK catalogue (see Parameter REFCAT).

If an input NDF has more pixel axes than the reference NDF, then the extra pixel axes are retained unchanged in the output NDF. Thus, for instance, if an input RA/Dec/velocity cube is aligned with a reference two-dimensional galactic-longitude/latitude image, the output NDF will be a galactic-longitude/latitude/velocity cube.

The transformations needed to produce alignment are derived from the co-ordinate system information stored in the WCS components of the supplied NDFs. For each input NDF, alignment is first attempted in the current co-ordinate Frame of the reference NDF. If this fails, alignment is attempted in the current co-ordinate Frame of the input NDF. If this fails, alignment occurs in the pixel co-ordinate Frame. A message indicating which Frame alignment was achieved in is displayed.

Two algorithms are available for determining the output pixel values: resampling and rebinning (the method used is determined by the REBIN parameter).

Two methods exist for determining the bounds of the output NDFs. First you can give values for Parameters LBND and UBND which are then used as the pixel index bounds for all output NDFs. Second, if a null value is given for LBND or UBND, default values are generated separately for each output NDF so that the output NDF just encloses the entire area covered by the corresponding input NDF. Using the first method will ensure that all output NDFs have the same pixel origin, and so the resulting NDFs can be directly compared. However, this may result in the output NDFs being larger than necessary. In general, the second method results in smaller NDFs being produced, in less time. However, the output NDFs will have differing pixel origins which need to be taken into account when comparing the aligned NDFs.

Usage:

```
wcsalign in out lbnd ubnd ref
```

Parameters:

ABORT = _LOGICAL (Read)

This controls what happens if an error occurs whilst processing one of the input NDFs. If a FALSE value is supplied for ABORT, then the error message will be displayed, but the application will attempt to process any remaining input NDFs. If a TRUE value is supplied for ABORT, then the error message will be displayed, and the application will abort. [FALSE]

ACC = _REAL (Read)

The positional accuracy required, as a number of pixels. For highly non-linear projections, a recursive algorithm is used in which successively smaller regions of the projection are fitted with a least-squares linear transformation. If such a

transformation results in a maximum positional error greater than the value supplied for ACC (in pixels), then a smaller region is used. High accuracy is paid for by larger run times. [0.5]

ALIGNREF = _LOGICAL (Read)

Determines the co-ordinate system in which each input NDF is aligned with the reference NDF. If TRUE, alignment is performed in the co-ordinate system described by the current Frame of the WCS FrameSet in the reference NDF. If FALSE, alignment is performed in the co-ordinate system specified by the following set of WCS attributes in the reference NDF: AlignSystem, AlignStdOfRest, AlignOffset, AlignSpecOffset, AlignSideBand, AlignTimeScale. The AST library provides fixed defaults for all these. So for instance, AlignSystem defaults to ICRS for celestial axes and Wavelength for spectral axes, meaning that celestial axes will be aligned in ICRS and spectral axes in wavelength, by default. Similarly, AlignStdOfRest defaults to Heliocentric, meaning that by default spectral axes will be aligned in the Heliocentric rest frame.

As an example, if you are aligning two spectra which both use radio velocity as the current WCS, but which have different rest frequencies, then setting ALIGNREF to TRUE will cause alignment to be performed in radio velocity, meaning that the differences in rest frequency are ignored. That is, a channel with 10 Km/s in the input is mapping onto the channel with 10 km/s in the output. If ALIGNREF is FALSE (and no value has been set for the AlignSystem attribute in the reference WCS), then alignment will be performed in wavelength, meaning that the different rest frequencies cause an additional shift. That is, a channel with 10 Km/s in the input will be mapping onto which ever output channel has the same wavelength, taking into account the different rest frequencies.

As another example, consider aligning two maps which both have (azimuth,elevation) axes. If ALIGNREF is TRUE, then any given (az,el) values in one image will be mapped onto the exact same (az,el) values in the other image, regardless of whether the two images were taken at the same time. But if ALIGNREF is FALSE, then a given (az,el) value in one image will be mapped onto pixel that has the same ICRS co-ordinates in the other image (since AlignSystem default to ICRS for celestial axes). Thus any different in the observation time of the two images will result in an additional shift.

As yet another example, consider aligning two spectra which are both in frequency with respect to the LSRK, but which refer to different points on the sky. If ALIGNREF is TRUE, then a given LSRK frequency in one spectrum will be mapped onto the exact same LSRK frequency in the other image, regardless of the different sky positions. But if ALIGNREF is FALSE, then a given input frequency will first be converted to Heliocentric frequency (the default value for AlignStdOfRest is "Heliocentric"), and will be mapped onto the output channel that has the same Heliocentric frequency. Thus the differecen in sky positions will result in an additional shift. [FALSE]

CONSERVE = _LOGICAL (Read)

If set TRUE, then the output pixel values will be scaled in such a way as to preserve the total data value in a feature on the sky. The scaling factor is the ratio of the output pixel size to the input pixel size. This option can only be used if the Mapping is successfully approximated by one or more linear transformations. Thus an error will be reported if it used when the ACC parameter is set to zero (which stops the use of linear approximations), or if the Mapping is too non-linear to be approximated by a piece-wise linear transformation. The ratio of output to input pixel size is evaluated

once for each panel of the piece-wise linear approximation to the Mapping, and is assumed to be constant for all output pixels in the panel. The dynamic default is TRUE if rebinning, and FALSE if resampling (see Parameter REBIN). []

IN = NDF (Read)
A group of input NDFs (of any dimensionality). This should be given as a comma-separated list, in which each list element can be:

- an NDF name, optionally containing wild-cards and/or regular expressions ("*", "?", "[a-z]" *etc.*).
- the name of a text file, preceded by an up-arrow character "^". Each line in the text file should contain a comma-separated list of elements, each of which can in turn be an NDF name (with optional wild-cards, *etc.*), or another file specification (preceded by an up-arrow). Comments can be included in the file by commencing lines with a hash character "#".

If the value supplied for this parameter ends with a minus sign "-", then you are re-prompted for further input until a value is given which does not end with a hyphen. All the NDFs given in this way are concatenated into a single group.

INSITU = _LOGICAL (Read)
If INSITU is set to TRUE, then no output NDFs are created. Instead, the pixel origin of each input NDF is modified in order to align the input NDFs with the reference NDF (which is a much faster operation than a full resampling). This can only be done if the mapping from input pixel co-ordinates to reference pixel co-ordinates is a simple integer pixel shift of origin. If this is not the case an error will be reported when the input is processed (what happens then is controlled by the ABORT parameter). Also, in-situ alignment is only possible if null values are supplied for LBND and UBND. [FALSE]

LBND() = _INTEGER (Read)
An array of values giving the lower pixel-index bound on each axis for the output NDFs. The number of values supplied should equal the number of axes in the reference NDF. The given values are used for all output NDFs. If a null value (!) is given for this parameter or for Parameter UBND, then separate default values are calculated for each output NDF which result in the output NDF just encompassing the corresponding input NDF. The suggested defaults are the lower pixel-index bounds from the reference NDF, if supplied (see Parameter REF).

MAXPIX = _INTEGER (Read)
A value which specifies an initial scale size in pixels for the adaptive algorithm which approximates non-linear Mappings with piece-wise linear transformations. If MAXPIX is larger than any dimension of the region of the output grid being used, a first attempt will be made to approximate the Mapping by a linear transformation over the entire output region. If a smaller value is used, the output region will first be divided into subregions whose size does not exceed MAXPIX pixels in any dimension, and then attempts will be made at approximation. [1000]

METHOD = LITERAL (Read)
The method to use when sampling the input pixel values (if resampling), or dividing an input pixel value up between a group of neighbouring output pixels (if rebinning).

For details of these schemes, see the descriptions of routines AST_RESAMPLEx and AST_REBINx in SUN/210. METHOD can take the following values.

- `"Bilinear"` — When resampling, the output pixel values are calculated by bi-linear interpolation among the four nearest pixels values in the input NDF. When rebinning, the input pixel value is divided up bi-linearly between the four nearest output pixels. Produces smoother output NDFs than the nearest-neighbour scheme, but is marginally slower.
- `"Nearest"` — When resampling, the output pixel values are assigned the value of the single nearest input pixel. When rebinning, the input pixel value is assigned completely to the single nearest output pixel.
- `"Sinc"` — Uses the $\mathrm{sinc}(\pi x)$ kernel, where x is the pixel offset from the interpolation point (resampling) or transformed input pixel centre (rebinning), and $\mathrm{sinc}(z) = \sin(z)/z$. Use of this scheme is not recommended.
- `"SincSinc"` — Uses the $\mathrm{sinc}(\pi x)\mathrm{sinc}(k\pi x)$ kernel. A valuable general-purpose scheme, intermediate in its visual effect on NDFs between the bi-linear and nearest-neighbour schemes.
- `"SincCos"` — Uses the $\mathrm{sinc}(\pi x)\cos(k\pi x)$ kernel. Gives similar results to the `"Sincsinc"` scheme.
- `"SincGauss"` — Uses the $\mathrm{sinc}(\pi x)e^{-kx^2}$ kernel. Good results can be obtained by matching the FWHM of the envelope function to the point-spread function of the input data (see Parameter PARAMS).
- `"Somb"` — Uses the $\mathrm{somb}(\pi x)$ kernel, where x is the pixel offset from the interpolation point (resampling) or transformed input pixel centre (rebinning), and $\mathrm{somb}(z) = 2 * J_1(z)/z$. J_1 is the first-order Bessel function of the first kind. This scheme is similar to the `"Sinc"` scheme.
- `"SombCos"` — Uses the $\mathrm{somb}(\pi x)\cos(k\pi x)$ kernel. This scheme is similar to the `"SincCos"` scheme.
- `"Gauss"` — Uses the e^{-kx^2} kernel. The FWHM of the Gaussian is given by Parameter PARAMS(2), and the point at which to truncate the Gaussian to zero is given by Parameter PARAMS(1).

All methods propagate variances from input to output, but the variance estimates produced by interpolation schemes other than nearest neighbour need to be treated with care since the spatial smoothing produced by these methods introduces correlations in the variance estimates. Also, the degree of smoothing produced varies across the NDF. This is because a sample taken at a pixel centre will have no contributions from the neighbouring pixels, whereas a sample taken at the corner of a pixel will have equal contributions from all four neighbouring pixels, resulting in greater smoothing and lower noise. This effect can produce complex Moiré patterns in the output variance estimates, resulting from the interference of the spatial frequencies in the sample positions and in the pixel centre positions. For these reasons, if you want to use the output variances, you are generally safer using nearest-neighbour interpolation. The initial default is `"SincSinc"`. [current value]

OUT = NDF (Write)

A group of output NDFs corresponding one-for-one with the list of input NDFs given for Parameter IN. This should be given as a comma-separated list, in which each list element can be:

- an NDF name. If the name contains an asterisk character "*", the name of the corresponding input NDF (without directory or file suffix) is substituted for the asterisk (for instance, "*_al" causes the output NDF name to be formed by appending the string "_al" to the corresponding input NDF name). Input NDF names can also be edited by including original and replacement strings between vertical bars after the NDF name (for instance, *_al|b4|B1| causes any occurrence of the string "B4" in the input NDF name to be replaced by the string "B1" before appending the string "_al" to the result).
- the name of a text file, preceded by an up-arrow character "^". Each line in the text file should contain a comma-separated list of elements, each of which can in turn be an NDF name (with optional editing, *etc.*), or another file specification (preceded by an up-arrow). Comments can be included in the file by commencing lines with a hash character "#".

If the value supplied for this parameter ends with a hyphen "-", then you are re-prompted for further input until a value is given which does not end with hyphen. All the NDFs given in this way are concatenated into a single group.

This parameter is only accessed if the INSITU parameter is FALSE.

PARAMS(2) = _DOUBLE (Read)

An optional array which consists of additional parameters required by the Sinc, SincSinc, SincCos, SincGauss, Somb, SombCos, and Gauss methods.

PARAMS(1) is required by all the above schemes. It is used to specify how many pixels are to contribute to the interpolated result on either side of the interpolation or binning point in each dimension. Typically, a value of 2 is appropriate and the minimum allowed value is 1 (*i.e.* one pixel on each side). A value of zero or fewer indicates that a suitable number of pixels should be calculated automatically. [0]

PARAMS(2) is required only by the Gauss, SombCos, SincSinc, SincCos, and SincGauss schemes. For the SombCos, SincSinc and SincCos schemes, it specifies the number of pixels at which the envelope of the function goes to zero. The minimum value is 1.0, and the run-time default value is 2.0. For the Gauss and SincGauss schemes, it specifies the full-width at half-maximum (FWHM) of the Gaussian envelope measured in output pixels. The minimum value is 0.1, and the run-time default is 1.0. On astronomical NDFs and spectra, good results are often obtained by approximately matching the FWHM of the envelope function, given by PARAMS(2), to the point-spread function of the input data. []

REBIN = _LOGICAL (Read)

Determines the algorithm used to calculate the output pixel values. If a TRUE value is given, a rebinning algorithm is used. Otherwise, a resampling algorithm is used. See the "Choice of Algorithm" topic below. The initial default is FALSE. [current value]

REF = NDF (Read)

The NDF to which all the input NDFs are to be aligned. If a null value is supplied for this parameter, the first NDF supplied for Parameter IN is used. This parameter is only used if no catalogue is supplied for Parameter REFCAT.

REFCAT = FILENAME (Read)

A POLPACK catalogue defining the WCS to which all the input NDFs are to be aligned. If a null value is supplied for this parameter, the WCS will be obtained from an NDF using Parameter REF. [!]

UBND() = _INTEGER (Read)
An array of values giving the upper pixel-index bound on each axis for the output NDFs. The number of values supplied should equal the number of axes in the reference NDF. The given values are used for all output NDFs. If a null value (!) is given for this parameter or for Parameter LBND, then separate default values are calculated for each output NDF which result in the output NDF just encompassing the corresponding input NDF. The suggested defaults are the upper pixel-index bounds from the reference NDF, if supplied (see Parameter REF).

WLIM = _REAL (Read)
This parameter is only used if REBIN is set TRUE. It specifies the minimum number of good pixels which must contribute to an output pixel for the output pixel to be valid. Note, fractional values are allowed. A null (!) value causes a very small positive value to be used resulting in output pixels being set bad only if they receive no significant contribution from any input pixel. [!]

Examples:

```
wcsalign image1 image1_al ref=image2 accept
```

This example resamples the NDF called image1 so that it is aligned with the NDF call image2, putting the output in image1_al. The output image has the same pixel-index bounds as image2 and inherits WCS information from image2.

```
wcsalign m51* *_al lbnd=!  accept
```

This example resamples all the NDFs with names starting with the string "m51" in the current directory so that they are aligned with the first input NDF. The output NDFs have the same names as the input NDFs, but extended with the string "_al". Each output NDF is just big enough to contain all the pixels in the corresponding input NDF.

```
wcsalign ^in.lis ^out.lis lbnd=!  accept
```

This example is like the previous example, except that the names of the input NDFs are read from the text file in.lis, and the names of the corresponding output NDFs are read from text file out.lis.

Choice of Algorithm :

The algorithm used to produce the output image is determined by the REBIN parameter, and is based either on resampling the output image or rebinning the input image.

The resampling algorithm steps through every pixel in the output image, sampling the input image at the corresponding position and storing the sampled input value in the output pixel. The method used for sampling the input image is determined by the METHOD parameter. The rebinning algorithm steps through every pixel in the input image, dividing the input pixel value between a group of neighbouring output pixels, incrementing these output pixel values by their allocated share of the input pixel value, and finally normalising each output value by the total number of contributing input values. The way in which the input sample is divided between the output pixels is determined by the METHOD parameter.

Both algorithms produce an output in which the each pixel value is the weighted mean of the near-by input values, and so do not alter the mean pixel values associated with a source, even if the pixel size changes. Thus the total data sum in a source will change if the input and output pixel sizes differ. However, if the CONSERVE parameter is set TRUE, the output values are scaled by the ratio of the output to input pixel size, so that the total data sum in a source is preserved.

A difference between resampling and rebinning is that resampling guarantees to fill the output image with good pixel values (assuming the input image is filled with good input pixel values), whereas holes can be left by the rebinning algorithm if the output image has smaller pixels than the input image. Such holes occur at output pixels which receive no contributions from any input pixels, and will be filled with the value zero in the output image. If this problem occurs the solution is probably to change the width of the pixel spreading function by assigning a larger value to PARAMS(1) and/or PARAMS(2) (depending on the specific METHOD value being used).

Both algorithms have the capability to introduce artefacts into the output image. These have various causes described below.

- Particularly sharp features in the input can cause rings around the corresponding features in the output image. This can be minimised by suitable settings for the METHOD and PARAMS parameters. In general such rings can be minimised by using a wider interpolation kernel (if resampling) or spreading function (if rebinning), at the cost of degraded resolution.
- The approximation of the Mapping using a piece-wise linear transformation (controlled by Parameter ACC) can produce artefacts at the joints between the panels of the approximation. They are caused by the discontinuities between the adjacent panels of the approximation, and can be minimised by reducing the value assigned to the ACC parameter.

Notes:

- WCS information (including the current co-ordinate Frame) is propagated from the reference NDF to all output NDFs.
- QUALITY is propagated from input to output only if Parameter METHOD is set to "Nearest" and REBIN is set to FALSE.

Related Applications :

KAPPA: WCSFRAME, REGRID; CCDPACK: TRANNDF.

Implementation Status:

- This routine correctly processes the DATA, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDFs (see the METHOD parameter for notes on the interpretation of output variances).
- Processing of bad pixels and automatic quality masking are supported.

- All non-complex numeric data types can be handled. If REBIN is `TRUE`, the data type will be converted to one of _INTEGER, _DOUBLE or _REAL for processing.

WCSATTRIB
Manages attribute values associated with the WCS component of an NDF

Description:

This application can be used to manage the values of attributes associated with the current co-ordinate Frame of an NDF (title, axis labels, axis units, *etc.*).

Each attribute has a name, a value, and a state. This application accesses all attribute values as character strings, converting to and from other data types as necessary. The attribute state is a Boolean flag (*i.e.* TRUE or FALSE) indicating whether or not a value has been assigned to the attribute. If no value has been assigned to an attribute, then it adopts a default value until an explicit value is assigned to it. An attribute value can be cleared, causing the attribute to revert to its default value.

The operation performed by this application is controlled by Parameter MODE, and can:

- display the value of an attribute;
- set a new value for an attribute;
- set new values for a list of attributes;
- clear an attribute value; and
- test the state of an attribute.

Note, the attributes of the PIXEL, GRID and AXIS Frames are managed internally by the NDF library. They may be examined using this application, but an error is reported if any attempt is made to change them. The exception to this is that the Domain attribute may be changed, resulting in a copy of the Frame being added to the WCS component of the NDF with the new Domain name. The AXIS Frame is derived from the AXIS structures within the NDF, so the AXLABEL and AXUNITS commands may be used to change the axis label or units string for the AXIS Frame.

Usage:

```
wcsattrib ndf mode name newval
```

Parameters:

MODE = LITERAL (Read)

The operation to be performed on the attribute specified by Parameter NAME. It can be one of the following options.

- `"Clear"` — Clears the current attribute value, causing it to revert to its default value.
- `"Get"` — The current value of the attribute is displayed on the screen and written to output Parameter VALUE. If the attribute has not yet been assigned a value (or has been cleared), then the default value will be displayed.

- "MSet" — Assigns new values to multiple attributes. The attribute names and values are obtained using Parameter SETTING.
- "Set" — Assigns a new value, given by Parameter NEWVAL, to the attribute.
- "Test" — Displays "TRUE" if the attribute has been assigned a value, and "FALSE" otherwise (in which case the attribute will adopt its default value). This flag is written to the output Parameter STATE.

The initial suggested default is "Get".

NAME = LITERAL (Read)
The attribute name. It is not used if MODE is "MSet".

NDF = NDF (Read and Write)
The NDF to be modified. When MODE="Get", the access is Read only.

NEWVAL = LITERAL (Read)
The new value to assign to the attribute. It is only used if MODE is "Set".

REMAP = _LOGICAL (Read)
Only accessed if MODE is "Set" or "Clear". If REMAP is TRUE, then the Mappings which connect the current Frame to the other Frames within the WCS FrameSet will be modified (if necessary) to maintain the FrameSet integrity. For instance, if the current Frame of the NDF represents FK5 RA and DEC, and you change System from "FK5" to "Galactic", then the Mappings which connect the SKY Frame to the other Frames (*e.g.* PIXEL, AXIS) will be modified so that each pixel corresponds to the correct Galactic co-ordinates. If REMAP is FALSE, then the Mappings will not be changed. This can be useful if the FrameSet has incorrect attribute values for some reason, which need to be corrected without altering the Mappings to take account of the change. [TRUE]

SETTING = LITERAL (Read)
This is only accessed if MODE is set to "MSet". It should hold a comma-separated list of "<attribute>=<value>" strings, where <attribute> is the name of an attribute and <value> is the value to assign to the attribute.

Results Parameters:

STATE = _LOGICAL (Write)
On exit, this holds the state of the attribute on entry to this application. It is not used if MODE is "MSet".

VALUE = LITERAL (Write)
On exit, this holds the value of the attribute on entry to this application. It is not used if MODE is "MSet".

Examples:

```
wcsattrib my_spec set System freq
```

This sets the System attribute of the current co-ordinate Frame in the NDF called my_Spec so that the Frame represents frequency (this assumes the current Frame is a SpecFrame). The Mappings between the current Frame and the other Frames are modified to take account of the change of system.

```
wcsattrib my_spec mset setting='unit(1)=km/s,system(1)=vrad'
```

This sets new values of "km/s" and "vrad" simultaneously for the Unit and System attributes for the first axis of the NDF called my_spec.

```
wcsattrib ngc5128 set title "Polarization map of Centaurus-A"
```

This sets the Title attribute of the current co-ordinate Frame in the NDF called ngc5128 to the string `"Polarization map of Centaurus-A"`.

```
wcsattrib my_data set domain saved_pixel
```

This sets the Domain attribute of the current co-ordinate Frame in the NDF called my_data to the string SAVED_PIXEL.

```
wcsattrib my_data set format(1) "%10.5G"
```

This sets the Format attribute for Axis 1 in the current co-ordinate Frame in the NDF called my_data, so that axis values are formatted as floating-point values using a minimum field width of ten characters, and displaying five significant figures. An exponent is used if necessary.

```
wcsattrib ngc5128 set format(2) bdms.2
```

This sets the Format attribute for Axis 2 in the current co-ordinate Frame in the NDF called ngc5128, so that axis values are formatted as separate degrees, minutes and seconds fields, separated by blanks. The seconds field has two decimal places. This assumes the current co-ordinate Frame in the NDF is a celestial co-ordinate Frame.

```
wcsattrib my_data get label(1)
```

This displays the label associated with the first axis of the current co-ordinate Frame in the NDF called my_data. A default label is displayed if no value has been set for this attribute.

```
wcsattrib my_data test label(1)
```

This displays `"TRUE"` if a value has been set for the Label attribute for the first axis of the current co-ordinate Frame in the NDF called my_data, and `"FALSE"` otherwise.

```
wcsattrib my_data clear label(1)
```

This clears the Label attribute for the first axis of the current co-ordinate Frame in the NDF called my_data. It reverts to its default value.

```
wcsattrib my_data set equinox J2000 remap=no
```

This assumes that the Equinox attribute for the current co-ordinate Frame within NDF `"my_data"` has been set to some incorrect value, which needs to be corrected to `"J2000"`. The REMAP parameter is set `FALSE`, which prevents the inter-Frame Mappings from being altered to take account of the new Equinox value. This means that each pixel in the NDF will retain its original RA and DEC values (but they will now be interpreted as J2000). If REMAP had been left at its default value of `TRUE`, then the RA and DEC associated with each pixel would have been modified in order to precess them from the original (incorrect) equinox to J2000.

Notes:

- An error is reported if an attempt is made to set or clear the Base Frame in the WCS component.
- The Domain names GRID, FRACTION, AXIS, and PIXEL are reserved for use by the NDF library and an error will be reported if an attempt is made to assign one of these values to any Frame.

Related Applications :

KAPPA: AXLABEL, AXUNITS, NDFTRACE, WCSADD, WCSFRAME, WCSREMOVE, WCSCOPY.

WCSCOPY
Copies WCS information from one NDF to another

Description:

This application copies the WCS component from one NDF to another, optionally modifying it to take account of a linear mapping between the pixel co-ordinates in the two NDFs. It can be used, for instance, to rectify the loss of WCS information produced by older applications which do not propagate the WCS component.

Usage:

```
wcscopy ndf like [tr] [confirm]
```

Parameters:

CONFIRM = _LOGICAL (Read)

If TRUE, the user is asked for confirmation before replacing any existing WCS component within the input NDF. No confirmation is required if there is no WCS component in the input NDF. [TRUE]

LIKE = NDF (Read)

The reference NDF data structure from which WCS information is to be copied.

NDF = NDF (Read and Write)

The input NDF data structure in which the WCS information is to be stored. Any existing WCS component is over-written (see Parameter CONFIRM).

OK = _LOGICAL (Read)

This parameter is used to get a confirmation that an existing WCS component within the input NDF can be over-written.

TR( ) = _DOUBLE (Read)

The values of this parameter are the coefficients of a linear transformation from pixel co-ordinates in the reference NDF given for Parameter LIKE, to pixel co-ordinates in the input NDF given for Parameter NDF. For instance, if a feature has pixel co-ordinates (X,Y,Z,...) in the reference NDF, and pixel co-ordinates (U,V,W,...) in the input NDF, then the following transformations would be used, depending on how many axes each NDF has:

- one-dimensional:

$$U = TR(1) + TR(2) * X$$

- two-dimensional:

$$U = TR(1) + TR(2) * X + TR(3) * Y$$
$$V = TR(4) + TR(5) * X + TR(6) * Y$$

- three-dimensional:

$$U = TR(1) + TR(2) * X + TR(3) * Y + TR(4) * Z$$
$$V = TR(5) + TR(6) * X + TR(7) * Y + TR(8) * Z$$
$$W = TR(9) + TR(10) * X + TR(11) * Y + TR(12) * Z$$

If a null value (!) is given it is assumed that the pixel co-ordinates of a given feature are identical in the two NDFs. [!]

Examples:

```
wcscopy m51_sim m51
```

This copies the WCS component from the NDF called m51 to the NDF called m51_sim, which may hold the results of a numerical simulation for instance. It is assumed that the two NDFs are aligned (*i.e.* the pixel co-ordinates of any feature are the same in both NDFs).

```
wcscopy m51_sqorst m51 [125,0.5,0.0,125,0.0,0.5]
```

This example assumes that an application similar to SQORST has previously been used to change the size of a two-dimensional NDF called m51, producing a new NDF called m51_sqorst. It is assumed that this SQORST-like application does not propagate WCS and also resets the pixel origin to [1, 1]. In fact, this is what SQORST actually did, prior to KAPPA version 1.0. This example shows how WCSCOPY can be used to rectify this by copying the WCS component from the original NDF m51 to the squashed NDF m51_sqorst, modifying it in the process to take account of both the squashing and the resetting of the pixel origin produced by SQORST. To do this, you need to work out the transformation in pixel co-ordinates produced by SQORST, and specify this when running WCSCOPY using the TR parameter. Let's assume the first axis of NDF m51 has pixel-index bounds of I1:I2 (these values can be found using NDFTRACE). If the first axis in the squashed NDF m51_sqorst spans$M$ pixels (where M is the value assigned to SQORST Parameter XDIM), then it will have pixel-index bounds of 1:M. Note, the lower bound is 1 since the pixel origin has been reset by SQORST. The squashing factor for the first axis is then:

$$FX = M/(I2 - I1 + 1)$$

and the shift in the pixel origin is:

$$SX = FX * (1 - I1)$$

Likewise, if the bounds of the second axis in m51 are J1:J2, and SQORST Parameter YDIM is set to N, then the squashing factor for the second axis is:

$$FY = N/(J2 - J1 + 1)$$

and the shift in the pixel origin is:

$$SY = FY * (1 - J1)$$

You would then use the following values for Parameter TR when running WCSCOPY:

$$TR = [SX, FX, 0.0, SY, 0.0, FY]$$

Note, the zero terms indicate that the axes are independent (*i.e.* there is no rotation of the image). The numerical values in the example are for an image with pixel-index bounds of 52:251 on both axes which was squashed by SQORST to produce an image with 100 pixels on each axis.

Notes:

- An error is reported if the transformation supplied using Parameter TR is singular.
- The pixel with pixel index *I* spans a range of pixel co-ordinate from $(I - 1.0)$ to *I*.
- The pixel indices of the bottom-left pixel in an NDF is called the *pixel origin* of the NDF, and can take any value. The pixel origin can be examined using application NDFTRACE and set using application SETORIGIN. WCSCOPY takes account of the pixel origins in the two NDFs when modifying the WCS component. Thus, if a null value is given for Parameter TR, the supplied WCS component may still be modified if the two NDFs have different pixel origins.

Related Applications :

KAPPA: NDFTRACE, WCSADD, WCSATTRIB, WCSFRAME, WCSREMOVE.

WCSFRAME
Changes the current co-ordinate Frame in the WCS component of an NDF

Description:

This application displays the current co-ordinate Frame associated with an NDF and then allows the user to specify a new Frame. The current co-ordinate Frame determines the co-ordinate system in which positions within the NDF will be expressed when communicating with the user.

Having selected a new current co-ordinate Frame, its attributes (such the specific system it uses to represents points within its Domain, its units, *etc.*) can be changed using KAPPA command WCSATTRIB.

Usage:

```
wcsframe ndf frame epoch
```

Parameters:

EPOCH = _DOUBLE (Read)

If a *Sky Co-ordinate System* specification is supplied (using Parameter FRAME) for a celestial co-ordinate system, 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.

FRAME = LITERAL (Read)

A string specifying the new co-ordinate Frame. If a null parameter value is supplied, then the current Frame is left unchanged. The suggested default is the Domain (or index if the Domain is not set) of the current Frame. The string can be one of the following:

- A domain name such as SKY, SPECTRUM, AXIS, PIXEL. The two 'pseudo-domains' WORLD and DATA may be supplied and will be translated into PIXEL and AXIS respectively, so long as the WCS component of the NDF does not contain Frames with these domains.
- 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). Using an SCS value is equivalent to specifying `"SKY"` for this parameter and then setting the System attribute (to `"FK5"`, `"Galactic"`, *etc.*) using KAPPA command WCSATTRIB. The specific system used to describe positions in other Domains (SPECTRUM, for instance) must be set using WCSATTRIB.

NDF = NDF (Read and Write)

The NDF data structure in which the current co-ordinate Frame is to be modified.

Examples:

```
wcsframe m51 pixel
```

This chooses pixel co-ordinates for the current co-ordinate Frame in the NDF m51.

```
wcsframe m51 sky
```

This chooses celestial co-ordinates for the current co-ordinate Frame in the NDF m51 (if available). The specific celestial co-ordinate system (FK5, Galactic, *etc.*) will depend on the contents of the WCS component of the NDF, but may be changed by setting a new value for the System attribute using the WCSATTRIB command.

```
wcsframe m51 spectral
```

This chooses spectral co-ordinates for the current co-ordinate Frame in the NDF m51 (if available). The specific spectral co-ordinate system (wavelength, frequency, *etc.*) will depend on the contents of the WCS component of the NDF, but may be changed by setting a new value for the System attribute using the WCSATTRIB command.

```
wcsframe m51 equ(J2000) epoch=1998.2
```

This chooses equatorial (RA/DEC) co-ordinates referred to the equinox at Julian epoch 2000.0 for the current co-ordinate Frame in the NDF m51. The positions were determined at the Julian epoch 1998.2 (this is needed to correct positions for the fictitious proper motions which may be introduced when converting between different celestial co-ordinate systems).

```
wcsframe m51 2
```

This chooses the second co-ordinate Frame in the WCS component of the NDF.

```
wcsframe m51 data
```

This chooses a co-ordinate Frame with domain DATA if one exists, or the AXIS co-ordinate Frame otherwise.

```
wcsframe m51 world
```

This chooses a co-ordinate Frame with domain WORLD if one exists, or the PIXEL co-ordinate Frame otherwise.

Notes:

- The current co-ordinate Frame in the supplied NDF is not displayed if a value is assigned to Parameter FRAME on the command line.

- This routine may add a new co-ordinate Frame into the WCS component of the NDF.
- The NDFTRACE command can be used to examine the co-ordinate Frames in the WCS component of an NDF.

Related Applications :

KAPPA: NDFTRACE, WCSATTRIB, WCSCOPY, WCSREMOVE.

WCSMOSAIC
Tiles a group of NDFs using World Co-ordinate System information

Description:

This application aligns and rebins a group of input NDFs into a single output NDF. It differs from WCSALIGN in both the algorithm used, and in the requirements placed on the input NDFs. WCSMOSAIC requires that the transformation from pixel to WCS co-ordinates be defined in each input NDF, but (unlike WCSALIGN) the inverse transformation from WCS to pixel co-ordinates need not be defined. For instance, this means that WCSMOSAIC can process data in which the WCS position of each input pixel is defined via a look-up table rather than an analytical expression. Note however, that the WCS information in the reference NDF (see Parameter REF) must have a defined inverse transformation.

The WCSMOSAIC algorithm proceeds as follows. First, the output NDF is filled with zeros. An associated array of weights (one weight for each output pixel) is created and is also filled with zeros. Each input NDF is then processed in turn. For each pixel in the current input NDF, the corresponding transformed position in the output NDF is found (based on the WCS information in both NDFs). The input pixel value is then divided up between a small group of output pixels centred on this central output position. The method used for choosing the fraction of the input pixel value assigned to each output pixel is determined by the METHOD and PARAMS parameters. Each of the affected output pixel values is then incremented by its allocated fraction of the input pixel value. The corresponding weight values are incremented by the fractions used (that is, if 0.25 of an input pixel is assigned to an output pixel, the weight for the output pixel is incremented by 0.25). Once all pixels in the current input NDF have been rebinned into the output NDF in this way, the algorithm proceeds to rebin the next input NDF in the same way. Once all input NDFs have been processed, output pixels which have a weight less than the value given by Parameter WLIM are set bad. The output NDF may then optionally (see Parameter NORM) be normalised by dividing it by the weights array. This normalisation of the output NDF takes account of any difference in the number of pixels contributing to each output pixel, and also removes artefacts which may be produced by aliasing between the input and output pixel grids. Thus each output pixel value is a weighted mean of the input pixel values from which it receives contributions. This means that the units of the output NDF are the same as the input NDF. In particular, any difference between the input and output pixel sizes is ignored, resulting in the total input data sum being preserved only if the input and output NDFs have equal pixel sizes. However, an option exists to scale the input values before use so that the total data sum in each input NDF is preserved even if the input and output pixel sizes differ (see Parameter CONSERVE).

If the input NDFs contain variances, then these are propagated to the output. Alternatively, output variances can be generated from the spread of input values contributing to each output pixel (see Parameter GENVAR). Any input variances can also be used to weight the input data (see Parameter VARIANCE). By default, all input data are given equal weight. An additional weight for each NDF can be specified using Parameter WEIGHTS.

The transformations needed to produce alignment are derived from the co-ordinate system information stored in the WCS components of the supplied NDFs. For each input NDF,

alignment is first attempted in the current co-ordinate Frame of the reference NDF. If this fails, alignment is attempted in the current co-ordinate Frame of the input NDF. If this fails, alignment occurs in the pixel co-ordinate Frame. A message indicating which Frame alignment was achieved in is displayed.

Usage:

```
wcsmosaic in out lbnd ubnd ref
```

Parameters:

ACC = _REAL (Read)

The positional accuracy required, as a number of pixels. For highly non-linear projections, a recursive algorithm is used in which successively smaller regions of the projection are fitted with a least-squares linear transformation. If such a transformation results in a maximum positional error greater than the value supplied for ACC (in pixels), then a smaller region is used. High accuracy is paid for by longer run times. [0.05]

ALIGNREF = _LOGICAL (Read)

Determines the co-ordinate system in which each input NDF is aligned with the reference NDF. If TRUE, alignment is performed in the co-ordinate system described by the current Frame of the WCS FrameSet in the reference NDF. If FALSE, alignment is performed in the co-ordinate system specified by the following set of WCS attributes in the reference NDF: AlignSystem AlignStdOfRest, AlignOffset, AlignSpecOffset, AlignSideBand, AlignTimeScale. The AST library provides fixed defaults for all these. So for instance, AlignSystem defaults to ICRS for celestial axes and Wavelength for spectral axes, meaning that celestial axes will be aligned in ICRS and spectral axes in wavelength, by default. Similarly, AlignStdOfRest defaults to Heliocentric, meaning that by default spectral axes will be aligned in the Heliocentric rest frame.

As an example, if you are mosaicing two spectra which both use radio velocity as the current WCS, but which have different rest frequencies, then setting ALIGNREF to TRUE will cause alignment to be performed in radio velocity, meaning that the differences in rest frequency are ignored. That is, a channel with 10 Km/s in the input is mapping onto the channel with 10 km/s in the output. If ALIGNREF is FALSE (and no value has been set for the AlignSystem attribute in the reference WCS), then alignment will be performed in wavelength, meaning that the different rest frequencies cause an additional shift. That is, a channel with 10 Km/s in the input will be mapping onto which ever output channel has the same wavelength, taking into account the different rest frequencies.

As another example, consider mosaicing two maps which both have (azimuth,elevation) axes. If ALIGNREF is TRUE, then any given (az,el) values in one image will be mapped onto the exact same (az,el) values in the other image, regardless of whether the two images were taken at the same time. But if ALIGNREF is FALSE, then a given (az,el) value in one image will be mapped onto pixel that has the same ICRS co-ordinates in the other image (since AlignSystem default to ICRS for celestial axes). Thus any different in the observation time of the two images will result in an additional shift.

As yet another example, consider mosaicking two spectra which are both in frequency with respect to the LSRK, but which refer to different points on the sky. If ALIGNREF is TRUE, then a given LSRK frequency in one spectrum will be mapped onto the exact same LSRK frequency in the other image, regardless of the different sky positions.

But if ALIGNREF is FALSE, then a given input frequency will first be converted to Heliocentric frequency (the default value for AlignStdOfRest is Heliocentric"), and will be mapped onto the output channel that has the same Heliocentric frequency. Thus the difference in sky positions will result in an additional shift. [FALSE]

CONSERVE = _LOGICAL (Read)

If set TRUE, then the output pixel values will be scaled in such a way as to preserve the total data value in a feature on the sky. The scaling factor is the ratio of the output pixel size to the input pixel size. This option can only be used if the Mapping is successfully approximated by one or more linear transformations. Thus an error will be reported if it used when the ACC parameter is set to zero (which stops the use of linear approximations), or if the Mapping is too non-linear to be approximated by a piece-wise linear transformation. The ratio of output to input pixel size is evaluated once for each panel of the piece-wise linear approximation to the Mapping, and is assumed to be constant for all output pixels in the panel. This parameter is ignored if the NORM parameter is set FALSE. [TRUE]

GENVAR = _LOGICAL (Read)

If TRUE, output variances are generated based on the spread of input pixel values contributing to each output pixel. Any input variances then have no effect on the output variances (although input variances will still be used to weight the input data if the VARIANCE parameter is set TRUE). If GENVAR is set FALSE, the output variances are based on the variances in the input NDFs, so long as all input NDFs contain variances (otherwise the output NDF will not contain any variances). If a null (!) value is supplied, then a value of FALSE is adopted if and only if all the input NDFs have variance components (TRUE is used otherwise). [FALSE]

IN = NDF (Read)

A group of input NDFs (of any dimensionality). This should be given as a comma-separated list, in which each list element can be one of the following options.

- An NDF name, optionally containing wild-cards and/or regular expressions ("*", "?", "[a-z]" *etc.*).
- The name of a text file, preceded by an up-arrow character "^". Each line in the text file should contain a comma-separated list of elements, each of which can in turn be an NDF name (with optional wild-cards, *etc.*), or another file specification (preceded by an up-arrow). Comments can be included in the file by commencing lines with a hash character "#".

If the value supplied for this parameter ends with a hyphen, then you are re-prompted for further input until a value is given which does not end with a hyphen. All the NDFs given in this way are concatenated into a single group.

LBND() = _INTEGER (Read)

An array of values giving the lower pixel-index bound on each axis for the output NDF. The suggested default values just encompass all the input data. A null value (!) also results in these same defaults being used. [!]

MAXPIX = _INTEGER (Read)

A value which specifies an initial scale size in pixels for the adaptive algorithm which approximates non-linear Mappings with piece-wise linear transformations. If MAXPIX is larger than sny dimension of the region of the output grid being used, a

first attempt will be made to approximate the Mapping by a linear transformation over the entire output region. If a smaller value is used, the output region will first be divided into subregions whose size does not exceed MAXPIX pixels in any dimension, and then attempts will be made at approximation. [1000]

METHOD = LITERAL (Read)

The method to use when dividing an input pixel value between a group of neighbouring output pixels. For details on these schemes, see the description of AST_REBINx in SUN/210. METHOD can take the following values.

- "Bilinear" — The input pixel value is divided bi-linearly between the four nearest output pixels. This produces smoother output NDFs than the nearest-neighbour scheme, but is marginally slower.
- "Nearest" — The input pixel value is assigned completely to the single nearest output pixel.
- "Sinc" — Uses the $\mathrm{sinc}(\pi x)$ kernel, where x is the pixel offset from the transformed input pixel centre, and $\mathrm{sinc}(z) = \sin(z)/z$. Use of this scheme is not recommended.
- "SincSinc" — Uses the $\mathrm{sinc}(\pi x)\mathrm{sinc}(k\pi x)$ kernel. This is a valuable general-purpose scheme, intermediate in its visual effect on NDFs between the bilinear and nearest-neighbour schemes.
- "SincCos" — Uses the $\mathrm{sinc}(\pi x)\cos(k\pi x)$ kernel. It gives similar results to the "Sincsinc" scheme.
- "SincGauss" — Uses the $\mathrm{sinc}(\pi x)e^{-kx^2}$ kernel. Good results can be obtained by matching the FWHM of the envelope function to the point-spread function of the input data (see Parameter PARAMS).
- "Somb" — Uses the $\mathrm{somb}(\pi x)$ kernel, where $\mathrm{somb}(z) = 2 * J_1(z)/z$. J_1 is the first-order Bessel function of the first kind. This scheme is similar to the "Sinc" scheme.
- "SombCos" — Uses the $\mathrm{somb}(\pi x)\cos(k\pi x)$ kernel. This scheme is similar to the "SincCos" scheme.
- "Gauss" — Uses the e^{-kx^2} kernel. The FWHM of the Gaussian is given by Parameter PARAMS(2), and the point at which to truncate the Gaussian to zero is given by Parameter PARAMS(1).

All methods propagate variances from input to output, but the variance estimates produced by schemes other than nearest neighbour need to be treated with care since the spatial smoothing produced by these methods introduces correlations in the variance estimates. Also, the degree of smoothing produced varies across the NDF. This is because a sample taken at a pixel centre will have no contributions from the neighbouring pixels, whereas a sample taken at the corner of a pixel will have equal contributions from all four neighbouring pixels, resulting in greater smoothing and lower noise. This effect can produce complex Moiré patterns in the output variance estimates, resulting from the interference of the spatial frequencies in the sample positions and in the pixel-centre positions. For these reasons, if you want to use the output variances, you are generally safer using nearest-neighbour interpolation. The initial default is "SincSinc". [current value]

NORM = _LOGICAL (Read)

In general, each output pixel contains contributions from multiple input pixel values, and the number of input pixels contributing to each output pixel will vary from pixel to pixel. If NORM is set TRUE (the default), then each output value is normalised by dividing it by the number of contributing input pixels, resulting in each output value being the weighted mean of the contibuting input values. However, if NORM is set FALSE, this normalisation is not applied. See also Parameter CONSERVE. Setting NORM to FALSE and VARIANCE to TRUE results in an error being reported. [TRUE]

OUT = NDF (Write)

The output NDF. If a null (!) value is supplied, WCSMOSAIC will terminate early without creating an output cube, but without reporting an error. Note, the pixel bounds which the output cube would have had will still be written to output Parameters LBOUND and UBOUND, even if a null value is supplied for OUT.

PARAMS(2) = _DOUBLE (Read)

An optional array which consists of additional parameters required by the Sinc, SincSinc, SincCos, SincGauss, Somb, SombCos, and Gauss methods.

PARAMS(1) is required by all the above schemes. It is used to specify how many output pixels on either side of the central output pixel are to receive contribution from the corresponding input pixel. Typically, a value of 2 is appropriate and the minimum allowed value is 1 (*i.e.* one pixel on each side). A value of zero or fewer indicates that a suitable number of pixels should be calculated automatically. [0]

PARAMS(2) is required only by the Gauss, SombCos, SincSinc, SincCos, and SincGauss schemes. For the SombCos, SincSinc and SincCos schemes, it specifies the number of output pixels at which the envelope of the function goes to zero. The minimum value is 1.0, and the run-time default value is 2.0. For the Gauss and SincGauss schemes, it specifies the full-width at half-maximum (FWHM) of the Gaussian envelope measured in output pixels. The minimum value is 0.1, and the run-time default is 1.0. []

REF = NDF (Read)

The NDF to which all the input NDFs are to be aligned. If a null value is supplied for this parameter, the first NDF supplied for Parameter IN is used. The WCS information in this NDF must have a defined inverse transformation (from WCS co-ordinates to pixel co-ordinates). [!]

UBND() = _INTEGER (Read)

An array of values giving the upper pixel-index bound on each axis for the output NDF. The suggested default values just encompass all the input data. A null value (!) also results in these same defaults being used. [!]

VARIANCE = _LOGICAL (Read)

If TRUE, then any input VARIANCE components in the input NDFs are used to weight the input data (the weight used for each data value is the reciprocal of the variance). If FALSE, all input data is given equal weight. Note, some applications (such as CCDPACK:MAKEMOS) use a parameter named USEVAR to determine both whether input variances are used to weights input data values, and also how to calculate output variances. However, WCSMOSAIC uses the VARIANCE parameter only for the first of these purposes (determining whether to weight the input data). The second purpose (determining how to create output variances) is fulfilled by the GENVAR parameter. [FALSE]

WEIGHTS = LITERAL (Read)
An optional group of numerical weights, one for each of the input NDFs specified by Parameter IN. If VARIANCE is `TRUE`, the weight assigned to each input pixel is the value supplied in this group correspoinding to the appropriate input NDF, divided by the variance of the pixel value. An error is reported if the number of supplied weights does not equal the number of supplied input NDFs. `[!]`

WLIM = _REAL (Read)
This parameter specifies the minimum number of good pixels that must contribute to an output pixel for the output pixel to be valid. Note, fractional values are allowed. If a value less than 1.0E-10 is supplied, a value of `1.0E-10` is used. `[1.0E-10]`

Results Parameters:

FLBND( ) = _DOUBLE (Write)
The lower bounds of the bounding box enclosing the output NDF in the current WCS Frame. The number of elements in this parameter is equal to the number of axes in the current WCS Frame. Celestial axis values will be in units of radians.

FUBND( ) = _DOUBLE (Write)
The upper bounds of the bounding box enclosing the output NDF in the current WCS Frame. The number of elements in this parameter is equal to the number of axes in the current WCS Frame. Celestial axis values will be in units of radians.

LBOUND() = _INTEGER (Write)
The lower pixel bounds of the output NDF. Note, values will be written to this output parameter even if a null value is supplied for Parameter OUT.

UBOUND() = _INTEGER (Write)
The upper pixel bounds of the output NDF. Note, values will be written to this output parameter even if a null value is supplied for Parameter OUT.

Examples:

```
wcsmosaic m51* mosaic lbnd=!  accept
```

This example rebins all the NDFs with names starting with the string `"m51"` in the current directory so that they are aligned with the first input NDF, and combines them all into a single output NDF called mosaic. The output NDF is just big enough to contain all the pixels in all the input NDFs.

Notes:

- WCS information (including the current co-ordinate Frame) is propagated from the reference NDF to the output NDF.
- QUALITY is not propagated from input to output.
- There are different facts reported, their verbosity depending on the current message-reporting level set by environment variable MSG_FILTER. If this is set to `QUIET`, no information will be displayed while the command is executing. When the filtering level is at least as verbose as `NORMAL`, the interpolation method being used will be displayed. If set to `VERBOSE`, the name of each input NDF will also be displayed as it is processed.

Related Applications :

KAPPA: WCSFRAME, WCSALIGN, REGRID; CCDPACK: TRANNDF.

Implementation Status:

- This routine correctly processes the DATA, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDFs (see the METHOD parameter for notes on the interpretation of output variances).
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled, but the data type will be converted to one of _INTEGER, _DOUBLE or _REAL for processing.

WCSREMOVE
Remove co-ordinate Frames from the WCS component of an NDF

Description:

This application allows you to remove one or more co-ordinate Frames from the WCS component in an NDF . The indices of any remaining Frames are 'shuffled down' to fill the gaps left by the removed Frames.

Usage:

```
wcsremove ndf frames
```

Parameters:

FRAMES() = LITERAL (Read

Specifies the Frame(s) to be removed. It can be a list of indices (within the WCS component of the supplied NDF) or list of Domain names. If one or more Domain name are specified, any WCS Frames which have a matching Domain are removed. If a list of indices is supplied, any indices outside the range of the available Frames are ignored. Single Frames or a set of adjacent Frames may be specified, *e.g.*. typing `[4,6-9,12,14-16]` will remove Frames 4,6,7,8,9,12,14,15,16. (Note that the brackets are required to distinguish this array of characters from a single string including commas. The brackets are unnecessary when there only one item.) If you wish to remove all the files enter the wildcard ∗. `5-∗` will remove from 5 to the last Frame.

NDF = NDF (Read and Write)

The NDF data structure.

Examples:

```
wcsremove m51 "3-5"
```

This removes Frames 3, 4 and 5 from the NDF called m51. Any remaining Frames with indices higher than 5 will be re-numbered to fill the gaps left by the removed Frames (*i.e.* the original Frame 6 will become Frame 3, *etc.*).

Notes:

- The Frames within the WCS component of an NDF may be examined using application NDFTRACE.

Related Applications :

KAPPA: NDFTRACE, WCSADD, WCSATTRIB, WCSCOPY, WCSFRAME.

WCSSHOW
Examines the internal structure of a WCS description.

Description:

This application allows you to examine WCS information (represented by an AST Object) stored in a specified NDF or HDS object, or a catalogue. The structure can be dumped to a text file, or a Graphical User Interface can be used to navigate through the structure (see Parameter LOGFILE). A new FrameSet can also be stored in the WCS component of an NDF (see Parameter NEWWCS). This allows an NDF WCS component to be dumped to a text file, edited, and then restored to the NDF.

The GUI main window contains the attribute values of the supplied AST Object. Only those associated with the Object's class are displayed initially, but attributes of the Objects parents classes can be displayed by clicking one of the class button to the top left of the window.

If the Object contains attributes which are themselves AST Objects (such as the Frames within a FrameSet), then new windows can be created to examine these attributes by clicking over the attribute name.

Usage:

```
wcsshow ndf object logfile newwcs full quiet
```

Parameters:

CAT = FILENAME (Read)

A catalogue containing a positions list such as produced by applications LISTMAKE, CURSOR. If supplied, the WCS Information in the catalogue is displayed. If a null (`!`) is supplied, the WCS information in the NDF specified by Parameter NDF is displayed. `[!]`

FULL = _INTEGER (Read)

This parameter is a three-state flag and takes values of `-1`, `0`, or `+1`. It controls the amount of information included in the output generated by this application. If FULL is zero, then a modest amount of non-essential but useful information will be included in the output. If FULL is negative, all non-essential information will be suppressed to minimise the amount of output, while if it is positive, the output will include the maximum amount of detailed information about the Object being examined. [current value]

LOGFILE = FILENAME (Write)

The name of the text file in which to store a dump of the specified AST Object. If a null (`!`) value is supplied, no log file is created. If a log file is given, the Tk browser window is not produced. `[!]`

NDF = NDF (Read or Update)

If an NDF is supplied, then its WCS FrameSet is displayed. If a null (`!`) value is supplied, then the Parameter OBJECT is used to specify the AST Object to display. Update access is required to the NDF if a value is given for Parameter NEWWCS.

Otherwise, only read access is required. Only accessed if a null (!) value is supplied for CAT.

NEWWCS = GROUP (Read)
A group expression giving a dump of an AST FrameSet which is to be stored as the WCS component in the NDF given by Parameter NDF. The existing WCS component is unchanged if a null value is supplied. The value supplied for this parameter is ignored if a null value is supplied for Parameter NDF. The Base Frame in the FrameSet is assumed to be the GRID Frame. If a value is given for this parameter, then the log file or Tk browser will display the new FrameSet (after being stored in the NDF and retrieved). [!]

OBJECT = LITERAL (Read)
The HDS object containing the AST Object to display. Only accessed if Parameters NDF and CAT are null. It must have an HDS type of WCS, must be scalar, and must contain a single one-dimensional array component with name DATA and type _CHAR.

QUIET = _LOGICAL (Read)
If TRUE, then the structure of the AST Object is not displayed (using the Tk GUI). Other functions are unaffected. If a null (!) value is supplied, the value used is TRUE if a non-null value is supplied for Parameter LOGFILE or Parameter NEWWCS, and FALSE otherwise. [!]

Examples:

```
wcsshow m51
```

Displays the WCS component of the NDF m51 in a Tk GUI.

```
wcsshow m51 logfile=m51.ast
```

Dumps the WCS component of the NDF m51 to text file m51.ast.

```
wcsshow m51 newwcs=^m51.ast
```

Reads a FrameSet from the text file m51.ast and stores it in the WCS component of the NDF m51. For instance, the text file m51.ast could be an edited version of the text file created in the previous example.

```
wcsshow object="~/agi_starprog.agi_3800_1.picture(4).more.ast_plot"
```

Displays the AST Plot stored in the AGI database with X windows picture number 4.

WCSSLIDE
Applies a translational correction to the WCS in an NDF

Description:

This application modifies the WCS information in an NDF so that the WCS position of a given pixel is moved by specified amount along each WCS axis. The shifts to use are specified either by an absolute offset vector given by the ABS parameter or by the difference between a fiducial point and a standard object given by the FID and OBJ parameters respectively. In each case the co-ordinates are specified in the NDF's current WCS co-ordinate Frame.

Usage:

```
wcsslide ndf abs
```

Parameters:

ABS( ) = _DOUBLE (Read)
Absolute shift for each WCS axis. The number of values supplied must match the number of WCS axes in the NDF. It is only used if STYPE="Absolute". Offsets for celestial longitude and latitude axes should be specified in arcseconds. Offsets for all other types of axes should be given directly in the units of the axis.

FID = LITERAL (Read)
A comma-separated list of formatted axis values giving the position of the fiducial point in WCS co-ordinates. The number of values supplied must match the number of WCS axes in the NDF. It is only used if STYPE="Relative".

NDF = NDF (Update)
The NDF to be translated.

OBJ = LITERAL (Read)
A comma-separated list of formatted axis values giving the position of the standard object in WCS co-ordinates. The number of values supplied must match the number of WCS axes in the NDF. It is only used if STYPE="Relative".

STYPE = LITERAL (Read)
The sort of shift to be used. The choice is "Relative" or "Absolute". ["Absolute"]

Examples:

```
wcsslide m31 [32,23]
```

The (RA,Dec) axes in the NDF m31 are shifted by 32 arcseconds in right ascension and 23 arcseconds in declination.

```
wcsslide speca stype=rel fid=211.2 obj=211.7
```

The spectral axis in the NDF speca (which measures frequency in GHz), is shifted by 0.5 GHz (*i.e.* 211.7–211.2).

```
wcsslide speca stype=abs abs=0.5
```

This does just the same as the previous example.

Notes:

- The correction is affected by translating pixel co-ordinates by a constant amount before projection them into WCS co-ordinates. Therefore, whilst the translation will be constant across the array in pixel co-ordinates, it may vary in WCS co-ordinates depending on the nature of the pixel→WCS transformation. The size of the translation in pixel co-ordinates is chosen in order to produce the required shift in WCS co-ordinates at the OBJ position (if STYPE is `"Relative"`), or at the array centre (if STYPE is `"Absolute"`).

Related Applications :

KAPPA: SLIDE.

Implementation Status:

- There can be an arbitrary number of NDF dimensions.

WCSTRAN
Transform a position from one NDF co-ordinate Frame to another

Description:

This application transforms a position from one NDF co-ordinate Frame to another. The input and output Frames may be chosen freely from the Frames available in the WCS component of the supplied NDF. The transformed position is formatted for display and written to the screen and also to an output parameter.

Usage:

```
wcstran ndf posin framein [frameout]
```

Parameters:

EPOCHIN = _DOUBLE (Read)

If a 'Sky Co-ordinate System' specification is supplied (using Parameter FRAMEIN) for a celestial co-ordinate system, then an epoch value is needed to qualify it. This is the epoch at which the supplied sky position was 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.

EPOCHOUT = _DOUBLE (Read)

If a 'Sky Co-ordinate System' specification is supplied (using Parameter FRAMEOUT) for a celestial co-ordinate system, then an epoch value is needed to qualify it. This is the epoch at which the transformed sky position is required. 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.

FRAMEIN = LITERAL (Read)

A string specifying the co-ordinate Frame in which the input position is supplied (see Parameter POSIN). If a null parameter value is supplied, then the current Frame in the NDF is used. The string can be one of the following:

- A domain name such as SKY, AXIS, PIXEL. The two 'pseudo-domains' WORLD and DATA may be supplied and will be translated into PIXEL and AXIS respectively, so long as the WCS component of the NDF does not contain Frames with these domains.
- 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).

FRAMEOUT = LITERAL (Read)

A string specifying the co-ordinate Frame in which the transformed position is required. If a null parameter value is supplied, then the current Frame in the NDF is used. The string can be one of the following:

- A domain name such as SKY, AXIS, PIXEL. The two 'pseudo-domains' WORLD and DATA may be supplied and will be translated into PIXEL and AXIS respectively, so long as the WCS component of the NDF does not contain Frames with these domains.
- 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).

NDF = NDF (Read and Write)
The NDF data structure containing the required co-ordinate Frames.

POSIN = LITERAL (Read)
The co-ordinates of the position to be transformed, in the co-ordinate Frame specified by Parameter FRAMEIN (supplying a colon `":"` will display details of the required co-ordinate Frame). The position should be supplied as a list of formatted axis values separated by spaces or commas.

QUIET = _LOGICAL (Read)
If `TRUE`, the transformed position is not written to the screen (it is still written to the output Parameter POSOUT). `[FALSE]`

SKYDEG = _INTEGER (Read)
If greater than zero, the values for any celestial longitude or latitude axes are formatted as decimal degrees, irrespective of the Format attributes in the NDF WCS component. The supplied integer value indicates the number of decimal places required. If the SKYDEG value is less than or equal to zero, the formats specified by the Format attributes in the WCS component are honoured. `[0]`

Results Parameters:

POSOUT = LITERAL (Write)
The formatted co-ordinates of the transformed position, in the co-ordinate Frame specified by Parameter FRAMEOUT. The position will be stored as a list of formatted axis values separated by spaces or commas.

Examples:

```
wcstran m51 "100.1 21.5" pixel
```

This transforms the pixel position `"100.1 21.5"` into the current co-ordinate Frame of the NDF m51. The results are displayed on the screen and written to the output Parameter POSOUT.

```
wcstran m51 "1:00:00 -12:30" equ(B1950) pixel
```

This transforms the RA/DEC position `"1:00:00 -12:30"` (referred to the J2000 equinox) into pixel co-ordinates within the NDF m51. The results are written to the output Parameter POSOUT.

```
wcstran m51 "1:00:00 -12:30" equ(B1950) equ(j2000)
```

This is like the previous example except that the position is transformed into RA/DEC referred to the B1950 equinox, instead of pixel co-ordinates.

Notes:

- The transformed position is not written to the screen when the message filter environment variable MSG_FILTER is set to `QUIET`. The creation of the output Parameter POSOUT is unaffected by MSG_FILTER.

Related Applications :

KAPPA: LISTMAKE, LISTSHOW, NDFTRACE, WCSATTRIB, WCSFRAME.

WIENER
Applies a Wiener filter to a one- or two-dimensional array

Description:

This application filters the supplied one- or two-dimensional array using a Wiener filter. It takes an array holding observed data and another holding a Point-Spread Function as input and produces an output restored array with potentially higher resolution and lower noise. Generally superior results can be obtained using applications MEM2D or LUCY, but at the cost of much more processing time.

The Wiener filter attempts to minimise the mean squared difference between the undegraded image and the restored image. To do this it needs to know the power spectrum of the undegraded image (*i.e.* the power at each spatial frequency before the instrumental blurring and the addition of noise). Obviously, this is not usually available, and instead the power spectrum of some other image must be used (the 'model' image). The idea is that a model image should be chosen for which there is some a priori reason for believing it to have a power spectrum similar to the undegraded image. Many different suggestions have been made for the best way to make this choice and the literature should be consulted for a detailed discussion (for instance, see the paper *Wiener Restoration of HST Images: Signal Models and Photometric Behavior* by I.C. Busko in the proceedings of the first Annual Conference on Astronomical Data Analysis Software and Systems, Tucson). By default, this application uses a 'white' model image, *i.e.* one in which there is equal power at all spatial frequencies. The default value for this constant power is the mean power per pixel in the input image. There is also an option to use the power spectrum of a supplied model image.

The filter also depends on a model of the noise in the supplied image. This application assumes that the noise is 'white' and is constant across the image. You can specify the noise power to use. If a noise power of zero is supplied, then the Wiener filter just becomes a normal inverse filter which will tend to amplify noise in the supplied image.

The filtering is done by multiplying the Fourier transform of the supplied image by the Fourier transform of the filter function. The output image is then created by taking the inverse Fourier transform of the product. The Fourier transform of the filter function is given by:

$$\frac{H^*}{|H|^2 + \frac{P_n}{P_g}}$$

where H is the Fourier transform of the supplied Point-Spread Function, P_n is the noise power, P_g is the power in the model image, and H^* is the complex conjugate of H. If the supplied model includes noise (as indicated by Parameter QUIET) then P_n is subtracted from P_g before evaluating the above expression.

Usage:

```
wiener in psf out xcentre ycentre
```

Parameters:

IN = NDF (Read)
The input NDF containing the observed data. This image may contain bad values, in which case the bad values will be replaced by zero before applying the filter. The resulting filtered image is normalised by dividing each pixel value by the corresponding weight of the good input pixels. These weights are found by filtering a mask image which holds the value one at every good input pixel, and zero at every bad input pixel.

MODEL = NDF (Read)
An NDF containing an image to use as the model for the power spectrum of the restored image. Any bad values in this image are replaced by the mean of the good values. If a null value is supplied then the model power spectrum is taken to be uniform with a value specified by Parameter PMODEL. [!]

OUT = NDF (Write)
The restored output array. An extension named WIENER is added to the output NDF to indicate that the image was created by this application (see Parameter QUIET).

PMODEL = _REAL (Read)
The mean power per pixel in the model image. This parameter is only accessed if a null value is supplied for Parameter MODEL. If a value is obtained for PMODEL then the model image is assumed to have the specified constant power at all spatial frequencies. If a null (!) value is supplied, the value used is the mean power per pixel in the input image. [!]

PNOISE = _REAL (Read)
The mean noise power per pixel in the observed data. For Gaussian noise this is equal to the variance. If a null (!) value is supplied, the value used is an estimate of the noise variance based on the difference between adjacent pixel values in the observed data. [!]

PSF = NDF (Read)
An NDF holding an estimate of the Point-Spread Function (PSF) of the input array. This could, for instance, be produced using the KAPPA application PSF. There should be no bad pixels in the PSF otherwise an error will be reported. The PSF can be centred anywhere within the array, but the location of the centre must be specified using Parameters XCENTRE and YCENTRE. The PSF is assumed to have the value zero outside the supplied NDF.

QUIET = _LOGICAL (Read)
This specifies whether or not the image given for Parameter MODEL (or the value given for Parameter PMODEL), includes noise. If the model does not include any noise then a TRUE value should be supplied for QUIET. If there is any noise in the model then QUIET should be supplied FALSE. If a null (!) value is supplied, the value used is FALSE, unless the image given for Parameter MODEL was created by a previous run of WIENER (as indicated by the presence of a WIENER extension in the NDF), in which case the run time default is TRUE (*i.e.* the previous run of WIENER is assumed to have removed the noise). [!]

THRESH = _REAL (Read)
The fraction of the PSF peak amplitude at which the extents of the PSF are determined. These extents are used to derive the size of the margins that pad the supplied input

array. Lower values of THRESH will result in larger margins being used. THRESH must be positive and less than 0.5. [0.0625]

TITLE = LITERAL (Read)
A title for the output NDF. A null (!) value means using the title of the input NDF. [!]

WLIM = _REAL (Read)
If the input array contains bad values, then this parameter may be used to determine the minimum weight of good input values required to create a good output value. It can be used, for example, to prevent output pixels from being generated in regions where there are relatively few good input values to contribute to the restored result. It can also be used to 'fill in' small areas (*i.e.* smaller than the PSF) of bad pixels.

The numerical value given for WLIM specifies the minimum total weight associated with the good pixels in a smoothing box required to generate a good output pixel (weights for each pixel are defined by the normalised PSF). If this specified minimum weight is not present, then a bad output pixel will result, otherwise a smoothed output value will be calculated. The value of this parameter should lie between 0.0 and 1.0. WLIM=0 causes a good output value to be created even if there is only one good input value, whereas WLIM=1 causes a good output value to be created only if all input values are good. [0.001]

XCENTRE = _INTEGER (Read)
The x pixel index of the centre of the PSF within the supplied PSF array. The suggested default is the middle pixel (rounded down if there are an even number of pixels per line).

YCENTRE = _INTEGER (Read)
The y pixel index of the centre of the PSF within the supplied PSF array. The suggested default is the middle line (rounded down if there are an even number of lines).

Examples:

```
wiener cenA star cenA_hires 11 13
```

This example deconvolves the array in the NDF called cenA, putting the resulting array in the NDF called cenA_hires. The PSF is defined by the array in NDF star, and the centre of the PSF is at pixel (11, 13).

```
wiener cenA star cenA_hires 11 13 pnoise=0
```

This example performs the same function as the previous example, except that the noise power is given as zero. This causes the Wiener filter to reduce to a standard inverse filter, which will result in more high frequencies being present in the restored image.

```
wiener cenA star cenA_hires 11 13 model=theory quiet
```

This example performs the same function as the first example, except that the power spectrum of the restored image is modelled on that of NDF theory, which may for instance contain a theoretical model of the object in NDF cenA, together with a simulated

star field. The Parameter QUIET is set to a TRUE value to indicate that the theoretical model contains no noise.

Notes:

- The convolutions required by the Wiener filter are performed by the multiplication of Fourier transforms. The supplied input array is extended by a margin along each edge to avoid problems of wrap-around between opposite edges of the array. The width of this margin is about equal to the width of the significant part of the PSF (as determined by Parameter THRESH). The application displays the width of these margins. The margins are filled by replicating the edge pixels from the supplied input NDFs.

Related Applications :

KAPPA: FOURIER, LUCY, MEM2D.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled. Arithmetic is performed using single-precision floating point.

ZAPLIN
Replaces regions in a two-dimensional NDF by bad values or by linear interpolation

Description:

This routine replaces selected areas within a two-dimensional input NDF (specified by Parameter IN), either by filling the areas with bad values, or by linear interpolation between neighbouring data values (see Parameter ZAPTYPE). Each area to be replaced can be either a range of pixel columns extending the full height of the image, a range of pixel lines extending the full width of the image, or a rectangular region with edges parallel to the pixel axes (see Parameter LINCOL).

The bounds of the area to be replaced can be specified either by using a graphics cursor, or directly in response to parameter prompts, or by supplying a text file containing the bounds (see Parameter MODE). In the first two modes the application loops asking for new areas to zap, until told to quit or an error is encountered. In the last mode processing stops when the end of file is found. An output text file may be produced containing a description of the areas replaced (see Parameter COLOUT). This file may be used to specify the regions to be replaced in a subsequent invocation of ZAPLIN.

Usage:

```
                          { lincol=?
zaplin in out [title]     { columns=?  lines=?
                          { colin=?
                        mode
```

Parameters:

COLIN = FILENAME (Read)

The name of a text file containing the bounds of the areas to be replaced. This parameter is only accessed if Parameter MODE is set to `"File"`. Each record in the file must be either a blank line, a comment (indicated by a `"!"` or `"#"` in column 1), or a definition of an area to be replaced, consisting of three or four space-separated fields. If a range of columns is to be replaced, each of the first two fields should be a formatted value for the first axis of the current co-ordinate Frame of the input NDF, and the third field should be the single character `"C"`. If a range of lines is to be replaced, each of the first two fields should be a formatted value for the second axis of the current co-ordinate Frame, and the third field should be the single character `"L"`. If a rectangular region is to be replaced, the first two fields should give the formatted values on axes 1 and 2 at one corner of the box, and the second two fields should give the formatted values on axes 1 and 2 at the opposite corner of the box.

COLOUT = FILENAME (Read)

The name of an output text file in which to store descriptions of the areas replaced by the current invocation of this application. It has the same format as the input file accessed using Parameter COLIN, and so may be used as input on a subsequent

invocation. This parameter is not accessed if Parameter MODE is set to "File". If COLOUT is null (!), no file will be created. [!]

COLUMNS = LITERAL (Read)
A pair of *x* values indicating the range of columns to be replaced. All columns between the supplied values will be replaced. This parameter is only accessed if Parameter LINCOL is set to "Columns" or "Region", and Parameter MODE is set to "Interface". Each *x* value should be given as a formatted value for Axis 1 of the current co-ordinate Frame of the input NDF. The two values should be separated by a comma, or by one or more spaces.

DEVICE = DEVICE (Read)
The graphics device to use if Parameter MODE is set to "Cursor". [Current graphics device]

IN = NDF (Read)
The input image.

LINCOL = LITERAL (Read)
The type of area is to be replaced. This parameter is only accessed if Parameter MODE is set to "Cursor" or "Interface". The options are as follows.

- "Lines" — Replaces lines of pixels between the *y* values specified by Parameter LINES. Each replaced line extends the full width of the image.
- "Columns" — Replaces columns of pixels between the *x* values specified by Parameter COLUMNS. Each replaced column extends the full height of the image.
- "Region" — Replaces the rectangular region of pixels within the *x* and *y* bounds specified by Parameters COLUMNS and LINES. The edges of the box are parallel to the pixel axes.

If this parameter is specified on the command line, and Parameter MODE is set to "Interface", only one area will be replaced; otherwise a series of areas will be replaced until a null (!) value is supplied for this parameter.

LINES = LITERAL (Read)
A pair of *y* values indicating the range of lines to be replaced. All lines between the supplied values will be replaced. This parameter is only accessed if Parameter LINCOL is set to "Lines" or "Region", and Parameter MODE is set to "Interface". Each *y* value should be given as a formatted value for Axis 2 of the current co-ordinate Frame of the input NDF. The two values should be separated by a comma, or by one or more spaces.

MARKER = INTEGER (Read)
This parameter is only accessed if Parameter PLOT is set to "Mark". It specifies the type of marker with which each cursor position should be marked, and should be given as an integer PGPLOT marker type. For instance, 0 gives a box, 1 gives a dot, 2 gives a cross, 3 gives an asterisk, 7 gives a triangle. The value must be larger than or equal to −31. [current value]

MODE = LITERAL (Read)
The method used to obtain the bounds of the areas to be replaced. The supplied string can be one of the following options.

- `"Interface"` — Bounds are obtained using Parameters COLUMNS and LINES. The type of area to be replaced is specified using Parameter LINCOL.
- `"Cursor"` — Bounds are obtained using the graphics cursor of the device specified by Parameter DEVICE. The type of area to be replaced is specified using Parameter LINCOL. The WCS information stored with the picture in the graphics database is used to map the supplied cursor positions into the pixel co-ordinate Frame of the input NDF. A message is displayed indicating the co-ordinate Frame in which the picture and the output NDF were aligned. Graphics may be drawn over the image indicating the region to be replaced (see Parameter PLOT).
- `"File"` — The bounds and type of each area to be replaced are supplied in the text file specified by Parameter COLIN.

[current value]

NOISE = _LOGICAL (Read)

This parameter is only accessed if Parameter ZAPTYPE is set to `"Linear"`. If a `TRUE` value is supplied, gaussian noise is added to each interpolated pixel value. The variance of the noise is equal to the variance of the data value being replaced. If the data variance is bad, no noise is added. If the input NDF has no VARIANCE component, variances equal to the absolute data value are used. This facility is provided for cosmetic use. `[FALSE]`

OUT = NDF (Write)

The output image.

PLOT = LITERAL (Read)

The type of graphics to be used to mark each cursor position. The appearance of these graphics (colour, size, *etc.*) is controlled by the STYLE parameter. PLOT can take any of the following values.

- `"Adapt"` — Causes `"Box"` to be used if a region is being replaced, `"Vline"` is a range of columns is being replaced, and `"Hline"` if a range of lines is being replaced.
- `"Box"` — A rectangular box with edges parallel to the edges of the graphics device is drawn with the two specified positions at opposite corners.
- `"Mark"` — Each position is marked by the symbol specified by Parameter MARKER.
- `"None"` — No graphics are produced.
- `"Vline"` — A vertial line is drawn through each specified position, extending the entire height of the selected picture.
- `"Hline"` — A horizontal line is drawn through each specified position, extending the entire width of the selected picture.

[current value]

STYLE = GROUP (Read)

A group of attribute settings describing the style to use when drawing the graphics specified by Parameter PLOT.

A comma-separated list of strings should be given in which each string is either an attribute setting, or the name of a text file preceded by an up-arrow character `"^"`. Such text files should contain further comma-separated lists which will be read and interpreted in the same manner. Attribute settings are applied in the order in which

they occur within the list, with later settings overriding any earlier settings given for the same attribute.

Each individual attribute setting should be of the form:

<name>=<value>

where <name> is the name of a plotting attribute, and <value> is the value to assign to the attribute. Default values will be used for any unspecified attributes. All attributes will be defaulted if a null value (!)—the initial default—is supplied. To apply changes of style to only the current invocation, begin these attributes with a plus sign. A mixture of persistent and temporary style changes is achieved by listing all the persistent attributes followed by a plus sign then the list of temporary attributes.

See Section E for a description of the available attributes. Any unrecognised attributes are ignored (no error is reported).

The appearance of vertical and horizontal lines is controlled by the attributes Colour(Curves), Width(Curves), *etc.* (the synonym Lines may be used in place of Curves). The appearance of boxes is controlled by the attributes Colour(Border), Size(Border), *etc.* (the synonym Box may be used in place of Border). The appearance of markers is controlled by attributes Colour(Markers), Size(Markers), *etc*. [current value]

TITLE = LITERAL (Read)

Title for the output image. A null value (!) propagates the title from the input image to the output image. [!]

USEAXIS = GROUP (Read)

USEAXIS is only accessed if the current co-ordinate Frame of the input NDF has more than two axes. A group of two strings should be supplied specifying the two axes spanning the plane containing the areas to be replaced. Each axis can be specified using one of the following options.

- Its integer index within the current Frame of the output NDF (in the range 1 to the number of axes in the current Frame).
- Its Symbol string such as "RA" or "VRAD".
- A generic option where "SPEC" requests the spectral axis, "TIME" selects the time axis, "SKYLON" and "SKYLAT" picks the sky longitude and latitude axes respectively. Only those axis domains present are available as options.

A list of acceptable values is displayed if an illegal value is supplied. If a null (!) value is supplied, the axes with the same indices as the first two significant NDF pixel axes are used. [!]

ZAPTYPE = LITERAL (Read)

The method used to choose the replacement pixel values. It should be one of the options below.

- "Bad" — Replace the selected pixels by bad values.
- "Linear" — Replace the selected pixels using linear interpolation. If a range of lines is replaced, then the interpolation is performed vertically between the first non-bad pixels above and below the selected lines. If a range of columns is replaced, then the interpolation is performed horizontally between the first

non-bad pixels to the left and right of the selected columns. If a rectangular region is replaced, then the interpolation is bi-linear between the nearest non-bad pixels on all four edges of the selected region. If interpolation is not possible (for instance, if the selected pixels are at the edge of the array) then the pixels are replaced with bad values. `["Linear"]`

Examples:

```
zaplin out=cleaned colout=fudge.dat
```

Assuming the current value of Parameter MODE is `"Cursor"`, this will copy the NDF associated with the last DATA picture to an NDF called cleaned, interactively replacing areas using the current graphics device. Linear interpolation is used to obtain the replacement values. A record of the areas replaced will be stored in a text file named `fudge.dat`.

```
zaplin grubby cleaned i lincol=r columns="188 190" lines="15 16"
```

This replaces a region from pixel (188, 15) to (190, 16) within the NDF called grubby and stores the result in the NDF called cleaned. The current co-ordinate Frame in the input NDF should be set to PIXEL first (using WCSFRAME). The replacement is performed using linear interpolation.

```
zaplin grubby(6,,) cleaned i lincol=r columns="188 190"
```

This replaces columns 188 to 190 in the 6th y-z plane region within the NDF called grubby and stores the result in the NDF called cleaned. The current co-ordinate Frame in the input NDF should be set to PIXEL first (using WCSFRAME). The replacement is performed using linear interpolation.

```
zaplin m42 m42c f colin=aaoccd1.dat zaptype=b
```

This flags with bad values the regions in the NDF called m42 defined in the text file called `aaoccd1.dat`, and stores the result in an NDF called m42c.

```
zaplin m42 m42c f colin=aaoccd1.dat noise
```

As above except that linear interpolation plus cosmetic noise are used to replace the areas to be cleaned rather than bad pixels.

Notes:

- Bounds supplied in Interface and File mode are transformed into the PIXEL Frame of the input NDF before being used.
- Complicated results arise if the axes of the current Frame of the input NDF are not parallel to the pixel axes. In these cases it is usually better to switch to the PIXEL Frame (using WCSFRAME) prior to using ZAPLIN. Roughly speaking, the range of

pixel lines and/or columns which are replaced will include any which intersect the specified range on the current-Frame axis.

- When using input files care should be taken to ensure that the co-ordinate system used in the file matches the current co-ordinate Frame of the input NDF.
- If the input NDF is a section of an NDF with a higher dimensionality, the `"lines"` and `"columns"` are with respect to the two-dimensional section, and do not necessarily refer to the first and second dimensions of the NDF as a whole. See the `"Examples"`.

Related Applications :

KAPPA: ARDMASK, CHPIX, FILLBAD, GLITCH, NOMAGIC, REGIONMASK, SEGMENT, SETMAGIC; FIGARO: CSET, ICSET, NCSET, TIPPEX.

Implementation Status:

- This routine correctly processes the AXIS, DATA, QUALITY, VARIANCE, LABEL, TITLE, UNITS, WCS, and HISTORY components of the input NDF and propagates all extensions.
- Processing of bad pixels and automatic quality masking are supported.
- All non-complex numeric data types can be handled.

D Descriptions of Frame Attributes

Each co-ordinate Frame has several *attributes* that determine its properties. This section lists the most important of these. See SUN/210 for a complete description of the properties of Frames and their attributes. The application WCSATTRIB can be used to examine attribute values associated with the current Frame of an NDF, and change the values of those that are not read-only.

Digits/Digits(axis)
Number of digits of precision

Description:

This attribute specifies how many digits of precision are required by default when a co-ordinate value is formatted for a Frame axis (*e.g.* when producing annotated plot axes). Its value may be set either for a Frame as a whole, or (by subscripting the attribute name with the number of an axis) for each axis individually. Any value set for an individual axis will override the value for the Frame as a whole.

Note that the Digits value acts only as a means of determining a default value for the Format attribute. Its effects are overridden if a Format string is set explicitly for an axis.

The default Digits value for a Frame is 7. If a value fewer than 1 is supplied, then 1 is used instead.

If the Frame is actually a SkyFrame (*e.g.* describes celestial longitude and latitude), then the Digits value specifies the total number of digits in the formatted axis value—that is, the sum of the hours (or degrees), minutes and seconds digits.

Examples:

```
wcsattrib m51 set digits 4
```

This sets the Digits attribute to the value 4 for all axes in the current Frame of the NDF `m51`. This results in axis values being formatted with four digits of precision when they are displayed by any application, so long as no value has been set for the Format attribute.

```
wcsattrib m51 set digits(1) 4
```

This is like the previous example, except that the Digits value is only set for the first axis in the current Frame of the NDF.

Domain
Co-ordinate system domain

Description:

This attribute contains a string that identifies the physical domain of the co-ordinate system that a Frame describes.

The Domain attribute also controls how Frames align with each other. If the Domain value in a Frame is set, then only Frames with the same Domain value can be aligned with it.

Some Frames are given standard Domain values when they are created (*e.g.* GRID, FRACTION, PIXEL, AXIS, SKY, SPECTRUM, CURPIC, NDC, BASEPIC, GRAPHICS). Frames created by the user (for instance, using WCSADD) can have any Domain value, but the standard Domain names should be avoided unless the standard meanings are appropriate for the Frame being created.

Examples:

```
wcsattrib m51 set domain oldpixel
```

If the current co-ordinate Frame in the NDF `m51` is the PIXEL Frame, then this command takes a 'snap-shot' of the PIXEL Frame and stores it as a new Frame with Domain OLDPIXEL in the WCS component. Subsequent changes to the PIXEL Frame (for instance, produced by applications that rotate, or move the contents of the NDF) will not effect the OLDPIXEL Frame, which will thus provide a 'frozen' record of the original PIXEL Frame.

Notes:

- All Domain values are converted to uppercase and white space is removed before use.

DSBCentre
The central position of interest in a dual-sideband spectrum (DSBSpecFrames only)

Description:

This attribute specifies the central position of interest in a dual-sideband spectrum. Its sole use is to determine the local oscillator frequency (the frequency which marks the boundary between the lower and upper sidebands). See the description of the IF (intermediate frequency) attribute for details of how the local oscillator frequency is calculated. The sideband containing this central position is referred to as the "observed" sideband, and the other sideband as the "image" sideband.

The value is accessed as a position in the spectral system specified by the System attribute, but is stored internally as topocentric frequency. Thus, if the System attribute of the DSBSpecFrame is set to "VRAD", the Unit attribute set to "m/s" and the StdOfRest attribute set

to "LSRK", then values for the DSBCentre attribute should be supplied as radio velocity in units of "m/s" relative to the kinematic LSR (alternative units may be used by appending a suitable units string to the end of the value). This value is then converted to topocentric frequency and stored. If (say) the Unit attribute is subsequently changed to "km/s" before retrieving the current value of the DSBCentre attribute, the stored topocentric frequency will be converted back to LSRK radio velocity, this time in units of "km/s", before being returned.

The default value for this attribute is 30 GHz.

Notes:

- The attributes which define the transformation to or from topocentric frequency should be assigned their correct values before accessing this attribute. These potentially include System, Unit, StdOfRest, ObsLon, ObsLat, ObsAlt, Epoch, RefRA, RefDec, and RestFreq.

Epoch
Epoch of observation

Description:

This attribute is used to qualify the co-ordinate system described by a Frame, by giving the moment in time when the co-ordinates are known to be correct. Often, this will be the date of observation.

The Epoch value is important in cases where the co-ordinate system changes with time. For instance, when considering celestial co-ordinate systems, possible reasons for change include: (i) changing aberration of light caused by the observer's velocity (*e.g.* due to the Earth's motion around the Sun), (ii) changing gravitational deflection by the Sun due to changes in the observer's position with time, (iii) fictitious motion due to rotation of non-inertial co-ordinate systems (*e.g.* the old FK4 system), and (iv) proper motion of the source itself (although this last effect is not handled by the SkyFrame class because it affects individual sources rather than the co-ordinate system as a whole).

The Epoch attribute is stored as a Modified Julian Date, and is not usually changed.

Notes:

- Care must be taken to distinguish the Epoch value, which relates to motion (or apparent motion) of the source, from the superficially similar Equinox value. The latter is used to qualify a co-ordinate system which is itself in motion in a (notionally) predictable way as a result of being referred to a slowly moving reference plane (*e.g.* the equator).
- See the description of the System attribute for details of which qualifying attributes apply to each celestial co-ordinate system.

Input Formats :

The formats accepted when setting an Epoch value are listed below. They are all case-insensitive and are generally tolerant of extra white space and alternative field delimiters.

- Besselian Epoch: Expressed in decimal years, with or without decimal places (`"B1950"` or `"B1976.13"` for example).
- Julian Epoch: Expressed in decimal years, with or without decimal places (`"J2000"` or `"J2100.9"` for example).
- Year: Decimal years, with or without decimal places (`"1996.8"` for example). Such values are interpreted as a Besselian epoch (see above) if less than 1984.0 and as a Julian epoch otherwise.
- Julian Date: With or without decimal places (`"JD 2454321.9"` for example).
- Modified Julian Date: With or without decimal places (`"MJD 54321.4"` for example).
- Gregorian Calendar Date: With the month expressed either as an integer or a 3-character abbreviation, and with optional decimal places to represent a fraction of a day (`"1996-10-2"` or `"1996-Oct-2.6"` for example). If no fractional part of a day is given, the time refers to the start of the day (zero hours).
- Gregorian Date and Time: Any calendar date (as above) but with a fraction of a day expressed as hours, minutes and seconds (`"1996-Oct-2 12:13:56.985"` for example).

Output Format :

When enquiring Epoch values, the format used is the "Year" format described under "Input Formats". This is a value in decimal years, which will be a Besselian epoch if less than 1984.0, and a Julian epoch otherwise.

Equinox
Epoch of the mean equinox (SkyFrames only)

Description:

This attribute is used to qualify those celestial co-ordinate systems described by a SkyFrame that are notionally based on the ecliptic (the plane of the Earth's orbit around the Sun) and/or the Earth's equator.

Both of these planes are in motion and their positions are difficult to specify precisely. In practice, therefore, a model ecliptic and/or equator are used instead. These, together with the point on the sky that defines the co-ordinate origin (the intersection of the two planes termed the 'mean equinox') move with time according to some models that remove the more-rapid fluctuations. The SkyFrame class supports both the old FK4 and the current FK5 models.

The position of a fixed source expressed in any of these co-ordinate systems will appear to change with time due to movement of the co-ordinate system itself (rather than motion of the source). Such co-ordinate systems must therefore be qualified by a moment in time

(the 'epoch of the mean equinox' or 'equinox' for short) which allows the position of the model co-ordinate system on the sky to be determined. This is the rôle of the Equinox attribute.

The Equinox attribute is stored as a Modified Julian Date, but when setting or getting its value you may use the same formats as for the Epoch attribute (q.v.).

Notes:

- Care must be taken to distinguish the Equinox value, which relates to the definition of a time-dependent co-ordinate system (based on solar-system reference planes which are in motion), from the superficially similar Epoch value. The latter is used to qualify co-ordinate systems where the positions of sources change with time (or appear to do so) for a variety of other reasons, such as aberration of light caused by the observer's motion, *etc.*
- See the description of the System attribute for details of which qualifying attributes apply to each celestial co-ordinate system.

Format(axis)
Format specification for axis values

Description:

This attribute specifies the format to be used when displaying co-ordinate values associated with a particular Frame axis (*i.e.* to convert values from binary to character form).

If no Format value is set for a Frame axis, a default value is supplied instead. This is based on the value of the Digits, or Digits(axis) attribute and is chosen so that it displays the requested number of digits of precision.

The interpretation of this string depends on whether or not the Frame is a SkyFrame. If it is not, the string is interpreted as a format-specification string to be passed to the C "`printf`" function (*e.g.* `"%1.7G"`) in order to format a single co-ordinate value (supplied as a double-precision number).

For SkyFrames, the syntax and default value of the Format string is re-defined to allow the formatting of sexagesimal values as appropriate for the particular celestial co-ordinate system being represented. The syntax of SkyFrame Format strings is described (below) in the "SkyFrame Formats" section.

SkyFrame Formats :

The Format string supplied for a SkyFrame should contain zero or more of the following characters. These may occur in any order, but the following is recommended for clarity.

- `"+"`: Indicates that a plus sign should be prefixed to positive values. By default, no plus sign is used.

- `"z"`: Indicates that leading zeros should be prefixed to the value so that the first field is of constant width, as would be required in a fixed-width table (leading zeros are always prefixed to any fields that follow). By default, no leading zeros are added.
- `"i"`: Use the standard ISO field separator (a colon) between fields. This is the default behaviour.
- `"b"`: Use a blank to separate fields.
- `"l"`: Use a letter (`"h"`/`"d"`, `"m"` or `"s"` as appropriate) to separate fields.
- `"g"`: This is the same as `"l"`, except that the separator characters are displayed as small superscripts when drawn on a graphical device.
- `"d"`: Include a degrees field. Expressing the angle purely in degrees is also the default if none of `"h"`, `"m"`, `"s"` or `"t"` are given.
- `"h"`: Express the angle as a time and include an hours field (where 24 hours correspond to 360 degrees). Expressing the angle purely in hours is also the default if `"t"` is given without either `"m"` or `"s"`.
- `"m"`: Include a minutes field. By default this is not included.
- `"s"`: Include a seconds field. By default this is not included. This request is ignored if `"d"` or `"h"` is given, unless a minutes field is also included.
- `"t"`: Express the angle as a time (where 24 hours correspond to 360 degrees). This option is ignored if either `"d"` or `"h"` is given and is intended for use where the value is to be expressed purely in minutes and/or seconds of time (with no hours field). If `"t"` is given without `"d"`, `"h"`, `"m"` or `"s"` being present, then it is equivalent to `"h"`.
- `"."`: Indicates that decimal places are to be given for the final field in the formatted string (whichever field this is). The `"."` should be followed immediately by an unsigned integer which gives the number of decimal places required. By default, no decimal places are given.

All of the above format specifiers are case-insensitive. If several characters make conflicting requests (*e.g.* if both `"i"` and `"b"` appear), then the character occurring last takes precedence, except that `"d"` and `"h"` always override `"t"`.

Examples:

```
wcsattrib my_data set format(1) %10.5G
```

This sets the Format attribute for Axis 1 in the current co-ordinate Frame in the NDF called `my_data`, so that axis values are formatted as floating-point values using a minimum field width of ten characters, and displaying five significant figures. An exponent is used if necessary.

```
wcsattrib ngc5128 set format(2) bdms.2
```

This sets the Format attribute for Axis 2 in the current co-ordinate Frame in the NDF called `ngc5128`, so that axis values are formatted as separate degrees, minutes and seconds field, separated by blanks. The seconds field has two decimal places. This assumes the current co-ordinate Frame in the NDF is a celestial co-ordinate Frame (*i.e.* a SkyFrame).

Notes:

- When specifying this attribute by name, it should be subscripted with the number of the Frame axis to which it applies.

IF
The intermediate frequency in a dual-sideband spectrum (DSBSpecFrames only)

Description:

This attribute specifies the (topocentric) intermediate frequency in a dual-sideband spectrum. Its sole use is to determine the local oscillator (LO) frequency (the frequency which marks the boundary between the lower and upper sidebands). The LO frequency is equal to the sum of the centre frequency and the intermediate frequency. Here, the "centre frequency" is the topocentric frequency in Hz corresponding to the current value of the DSBCentre attribute. The value of the IF attribute may be positive or negative: a positive value results in the LO frequency being above the central frequency, whilst a negative aattIF value results in the LO frequency being below the central frequency. The sign of the IF attribute value determines the default value for the SideBand attribute.

When setting a new value for this attribute, the units in which the frequency value is supplied may be indicated by appending a suitable string to the end of the formatted value. If the units are not specified, then the supplied value is assumed to be in units of GHz. For instance, the following strings all result in an IF of 4 GHz being used: "4.0", "4.0 GHz", "4.0E9 Hz", *etc.*

When getting the value of this attribute, the returned value is always in units of GHz. The default value for this attribute is 4 GHz.

ImagFreq
The image sideband equivalent of the rest frequency (DSBSpecFrames only)

Description:

This is a read-only attribute of a dual-sideband spectrum that gives the frequency corresponding to the rest frequency, but in the opposite sideband.

The value is calculated by first transforming the rest frequency (given by the RestFreq attribute) from the standard of rest of the source (given by the SourceVel and SourceVRF attributes) to the standard of rest of the observer (*i.e.* the topocentric standard of rest). The resulting topocentric frequency is assumed to be in the same sideband as the value given for the DSBCentre attribute (the "observed" sideband), and is transformed to the other

sideband (the "image" sideband). The new frequency is converted back to the standard of rest of the source, and the resulting value is returned as the attribute value, in units of GHz.

Label(axis)
Axis label

Description:

This attribute specifies a label to be attached to each axis of a Frame when it is represented (*e.g.*) in graphical output.

If a Label value has not been set for a Frame axis, then a suitable default is supplied, depending on whether or not the Frame is a SkyFrame.

The default for simple Frames is the string "Axis <n>", where <n> is 1, 2, *etc.* for each successive axis.

The default labels for specialised Frames (SkyFrames, SpecFrames, *etc.*) depend on the particular co-ordinate system represented by the Frame (*e.g.* "Right ascension", "Galactic latitude", "Frequency", "Wavelength in air", *etc.*).

Examples:

```
wcsattrib my_data set label(2) "IRAS data (marked in white)"
```

This sets the Label for Axis 2 in the current Frame in the NDF called my_data, to the string "IRAS data (marked in white)".

Notes:

- Axis labels are intended purely for interpretation by human readers and not by software.
- When specifying this attribute by name, it should be subscripted with the number of the Frame axis to which it applies.

LTOffset
The offset from UTC to Local Time, in hours (TimeFrames only)

Description:

This specifies the offset from UTC to Local Time, in hours, for a TimeFrame. Fractional hours can be supplied. It is positive for time zones east of Greenwich. AST uses the figure as given, without making any attempt to correct for daylight saving. The default value is zero.

Naxes
Number of Frame axes

Description:

This is a read-only attribute giving the number of axes in a Frame (*i.e.* the number of dimensions of the co-ordinate space that the Frame describes). This value is determined when the Frame is created.

Examples:

```
wcsattrib my_data get naxes
```

This displays the number of axes in the current Frame of the NDF called `my_data`.

ObsLat
The geodetic latitude of the observer (SpecFrames only)

Description:

This attribute specifies the geodetic latitude of the observer, in degrees. Together with the ObsLon, Epoch, RefRA, and RefDec attributes, it defines the Doppler shift introduced by the observers diurnal motion around the Earth's axis, which is needed when converting SpecFrames to or from the topocentric standard of rest. The maximum velocity error, which can be caused by an incorrect value, is 0.5 km/s. The default value for the attribute is zero.

The value is stored internally in radians, but is converted to and from a degrees string for access. Some example input formats are: `"22:19:23.2"`, `"22 19 23.2"`, `"22:19.387"`, `"22.32311"`, `"N22.32311"`, `"-45.6"`, `"S45.6"`. As indicated, the sign of the latitude can optionally be indicated using characters `"N"` and `"S"` in place of the usual `"+"` and `"-"`. When converting the stored value to a string, the format `"[s]dd:mm:ss.s"` is used, when `"[s]"` is `"N"` or `"S"`.

ObsLon
The geodetic longitude of the observer (SpecFrames only)

Description:

This attribute specifies the geodetic (or equivalently, geocentric) longitude of the observer, in degrees, measured positive eastwards. See also attribute ObsLat. The default value is zero.

The value is stored internally in radians, but is converted to and from a degrees string for access. Some example input formats are: `"155:19:23.2"`, `"155 19 23.2"`, `"155:19.387"`, `"155.32311"`, `"E155.32311"`, `"-204.67689"`, `"W204.67689"`. As indicated, the sign of the longitude can optionally be indicated using characters `"E"` and `"W"` in place of the usual `"+"` and `"-"`. When converting the stored value to a string, the format `"[s]ddd:mm:ss.s"` is used, when `"[s]"` is `"E"` or `"W"` and the numerical value is chosen to be less than 180 degrees.

RefDec
The declination of the reference point (SpecFrames only)

Description:

This attribute specifies the FK5 J2000.0 declination of a reference point on the sky. See the description of attribute RefRA for details. This attribute has a default value of zero.

RefRA
The right ascension of the reference point (SpecFrames only)

Description:

This attribute, together with the RefDec attribute, specifies the FK5 J2000.0 co-ordinates of a reference point on the sky. For one-dimensional spectra, this should normally be the position of the source. For spectral data with spatial coverage (spectral cubes, *etc.*), this should be close to centre of the spatial coverage. It is used to define the correction for Doppler shift to be applied when converting between different standards of rest.

The RefRA and RefDec attributes are stored internally in radians, but are converted to and from a string for access. The format "hh:mm:ss.ss" is used for RefRA, and "dd:mm:ss.s" is used for RefDec.

RefRA has a default value of zero.

RestFreq
The rest frequency (SpecFrames only)

Description:

This attribute specifies the frequency corresponding to zero velocity. It is used when converting between between velocity-based spectral co-ordinate systems and and other co-ordinate systems (such as frequency, wavelength, energy, *etc.*). The default value is 1.0E5 GHz.

When setting a new value for this attribute, the new value can be supplied either directly as a frequency, or indirectly as a wavelength or energy, in which case the supplied value is converted to a frequency before being stored. The nature of the supplied value is indicated by appending text to the end of the numerical value indicating the units in which the value is supplied. If the units are not specified, then the supplied value is assumed to be a frequency in units of GHz. If the supplied unit is a unit of frequency, the supplied value is assumed to be a frequency in the given units. If the supplied unit is a unit of length, the supplied value is assumed to be a (vacuum) wavelength. If the supplied unit is a unit of energy, the supplied value is assumed to be an energy. For instance, the following strings all result in a rest frequency of around 1.4E14 Hz being used: `"1.4E5"`, `"1.4E14 Hz"`, `"1.4E14 s**-1"`, `"1.4E5 GHz"`, `"2.14E-6 m"`, `"21400 Angstrom"`, `"9.28E-20 J"`, `"9.28E-13 erg"`, `"0.58 eV"`, *etc.*

When getting the value of this attribute, the returned value is always a frequency in units of GHz.

SideBand
Indicates which sideband a dual sideband spectrum represents (DSBSpecFrames only)

Description:

This attribute indicates whether a dual-sideband spectrum currently represents its lower or upper sideband, or an offset from the local oscillator frequency. When querying the current value, the returned string is always one of "usb" (for upper sideband), "lsb" (for lower sideband), or "lo" (for offset from the local oscillator frequency). When setting a new value, any of the strings "lsb", "usb", "observed", "image" or "lo" may be supplied (case insensitive). The "observed" sideband is which ever sideband (upper or lower) contains the central spectral position given by attribute DSBCentre, and the "image" sideband is the other sideband. It is the sign of the IF attribute which determines if the observed sideband is the upper or lower sideband. The default value for SideBand is the observed sideband.

SourceVel
The source velocity (SpecFrames only)

Description:

This attribute (together with SourceVRF, RefRA, and RefDec) defines the 'Source' standard of rest (see attribute StdOfRest). This is a rest frame that is moving towards the position given by RefRA and RefDec at a velocity given by SourceVel (in km/s). When setting a value for SourceVel using WCSATTRIB, the velocity should be supplied in the rest frame specified by the SourceVRF attribute. Likewise, when getting the value of SourceVel, it will be returned in the rest frame specified by the SourceVRF attribute.

The default value is zero.

SourceVRF
Rest frame in which the source velocity is stored (SpecFrames only)

Description:

This attribute identifies the rest frame in which the source velocity is stored (the source velocity is accessed using attribute SourceVel). When setting a new value for the SourceVel attribute, the source velocity should be supplied in the rest frame indicated by this attribute. Likewise, when getting the value of the SourceVel attribute, the velocity will be returned in this rest frame.

If the value of SourceVRF is changed, the value stored for SourceVel will be converted from the old to the new rest frame.

The values that can be supplied are the same as for the StdOfRest attribute (except that SourceVRF cannot be set to `"Source"`). The default value is `"Helio"`.

StdOfRest
Standard of rest (SpecFrames only)

Description:

This attribute identifies the standard of rest to which the spectral axis values of a SpecFrame refer, and may take any of the values listed in the "Standards of Rest" section (below).

The default StdOfRest value is `"Helio"`.

Standards of Rest :

The SpecFrame class supports the following StdOfRest values (all are case-insensitive).

- `"Topocentric"`, `"Topocent"` or `"Topo"`: The observers rest-frame (assumed to be on the surface of the earth). Spectra recorded in this standard of rest suffer a Doppler shift which varies over the course of a day because of the rotation of the observer around the axis of the earth. This standard of rest must be qualified using the ObsLat, ObsLon, Epoch, RefRA, and RefDec attributes.
- `"Geocentric"`, `"Geocentr"` or `"Geo"`: The rest-frame of the earth centre. Spectra recorded in this standard of rest suffer a Doppler shift which varies over the course of a year because of the rotation of the earth around the Sun. This standard of rest must be qualified using the Epoch, RefRA, and RefDec attributes.
- `"Barycentric"`, `"Barycent"` or `"Bary"`: The rest-frame of the solar-system barycentre. Spectra recorded in this standard of rest suffer a Doppler shift which depends both on the velocity of the Sun through the Local Standard of Rest, and on the movement of the planets through the solar system. This standard of rest must be qualified using the Epoch, RefRA, and RefDec attributes.

- `"Heliocentric"`, `"Heliocen"` or `"Helio"`: The rest-frame of the Sun. Spectra recorded in this standard of rest suffer a Doppler shift which depends on the velocity of the Sun through the Local Standard of Rest. This standard of rest must be qualified using the RefRA and RefDec attributes.
- `"LSR"`, `"LSRK"`: The rest-frame of the kinematical Local Standard of Rest. Spectra recorded in this standard of rest suffer a Doppler shift which depends on the velocity of the kinematical Local Standard of Rest through the galaxy. This standard of rest must be qualified using the RefRA and RefDec attributes.
- `"LSRD"`: The rest-frame of the dynamical Local Standard of Rest. Spectra recorded in this standard of rest suffer a Doppler shift which depends on the velocity of the dynamical Local Standard of Rest through the galaxy. This standard of rest must be qualified using the RefRA and RefDec attributes.
- `"Galactic"`, `"Galactoc"` or `"Gal"`: The rest-frame of the galactic centre. Spectra recorded in this standard of rest suffer a Doppler shift which depends on the velocity of the galactic centre through the local group. This standard of rest must be qualified using the RefRA and RefDec attributes.
- `"Local_group"`, `"Localgrp"` or `"LG"`: The rest-frame of the local group. This standard of rest must be qualified using the RefRA and RefDec attributes.
- `"Source"`, or `"src"`: The rest-frame of the source. This standard of rest must be qualified using the RefRA, RefDec, and SourceVel attributes.

Where more than one alternative System value is shown above, the first of these will be returned when an enquiry is made.

Symbol(axis)
Axis symbol

Description:

This attribute specifies a short-form symbol to be used to represent co-ordinate values for a particular axis of a Frame. This might be used (*e.g.*) in algebraic expressions where a full description of the axis would be inappropriate. Examples include `"RA"` and `"Dec"` (for right ascension and declination).

If a Symbol value has not been set for a Frame axis, then a suitable default is supplied.

The default Symbol value supplied for simple Frames is the string "<Domain><n>", where <n> is 1, 2, *etc.* for successive axes, and <Domain> is the value of the Frame's Domain attribute (truncated if necessary so that the final string does not exceed 15 characters). If no Domain value has been set, `"x"` is used as the <Domain> value in constructing this default string.

Specialised Frames (SkyFrame, SpecFrame,*etc.*) re-define the default Symbol value to be appropriate for the particular co-ordinate system being represented.

Examples:

```
wcsattrib my_data set symbol(2) AR
```

This sets the Symbol for Axis 2 in the current Frame in the NDF called `my_data`, to the string `"AR"`.

Notes:

- When specifying this attribute by name, it should be subscripted with the number of the Frame axis to which it applies.

System
Co-ordinate system used to describe positions within the domain

---

Description:

In general it is possible for positions within a given physical domain to be described using one of several different co-ordinate systems. For instance, the SkyFrame class can use galactic co-ordinates, equatorial co-ordinates *etc.* to describe positions on the sky. As another example, the SpecFrame class can use frequency, wavelength, velocity *etc.* to describe a position within an electromagnetic spectrum. The System attribute identifies the particular co-ordinate system represented by a Frame. Each class of Frame defines a set of acceptable values for this attribute, as listed below (all are case insensitive). Where more than one alternative System value is shown, the first of will be returned when an enquiry is made.

Applicability:

Frame

The System attribute for a basic Frame always equals `"Cartesian"`, and may not be altered.

SkyFrame

The SkyFrame class supports the following System values and associated celestial co-ordinate systems.

- `"FK4"`: The old FK4 (barycentric) equatorial co-ordinate system, which should be qualified by an Equinox value. The underlying model on which this is based is non-inertial and rotates slowly with time, so for accurate work FK4 co-ordinate systems should also be qualified by an Epoch value.
- `"FK4-NO-E"` or `"FK4_NO_E"`: The old FK4 (barycentric) equatorial system but without the *E-terms of aberration* (*e.g.* some radio catalogues). This co-ordinate system should also be qualified by both an Equinox and an Epoch value.
- `"FK5"` or `"EQUATORIAL"`: The modern FK5 (barycentric) equatorial co-ordinate system. This should be qualified by an Equinox value.

- `"GAPPT"`, `"GEOCENTRIC"` or `"APPARENT"`: The geocentric apparent equatorial co-ordinate system, which gives the apparent positions of sources relative to the true plane of the Earth's equator and the equinox (the co-ordinate origin) at a time specified by the qualifying Epoch value. (Note that no Equinox is needed to qualify this co-ordinate system because no model 'mean equinox' is involved.) These co-ordinates give the apparent right ascension and declination of a source for a specified date of observation, and therefore form an approximate basis for pointing a telescope. Note, however, that they are applicable to a fictitious observer at the Earth's centre, and therefore ignore such effects as atmospheric refraction and the (normally much smaller) aberration of light due to the rotational velocity of the Earth's surface. Geocentric apparent co-ordinates are derived from the standard FK5 (J2000.0) barycentric co-ordinates by taking account of the gravitational deflection of light by the Sun (usually small), the aberration of light caused by the motion of the Earth's centre with respect to the barycentre (larger), and the precession and nutation of the Earth's spin axis (normally larger still).
- `"ECLIPTIC"`: Ecliptic co-ordinates (IAU 1980), referred to the ecliptic and mean equinox specified by the qualifying Equinox value.
- `"GALACTIC"`: Galactic co-ordinates (IAU 1958).
- `"SUPERGALACTIC"`: De Vaucouleurs Supergalactic co-ordinates.
- `"UNKNOWN"`: Any other general spherical co-ordinate system. No Mapping can be created between a pair of SkyFrames if either of the SkyFrames has System set to `"Unknown"`.

Currently, the default System value is `"FK5"`.

SpecFrame

The SpecFrame DSBSpecFrame classes supports the following System values and associated spectral co-ordinate systems (the default is `"WAVE"`—wavelength):

- `"FREQ"`: Frequency (Hz)
- `"ENER"` or `"ENERGY"`: Energy (J)
- `"WAVN"` or `"WAVENUM"`: Wave-number (1/m)
- `"WAVE"` or `"WAVELEN"`: Vacuum wavelength (m)
- `"AWAV"` or `"AIRWAVE"`: Wave-length in air (m)
- `"VRAD"` or `"VRADIO"`: Radio velocity (m/s)
- `"VOPT"` or `"VOPTICAL"`: Optical velocity (m/s)
- `"ZOPT"` or `"REDSHIFT"`: Redshift (dimensionless)
- `"BETA"`: Beta factor (dimensionless)
- `"VELO"` or `"VREL"`: Relativistic velocity (m/s)

The default value for the Unit attribute for each system is shown in parentheses. Note, changes to the Unit attribute for a SpecFrame will result in the Mapping from pixel to spectral co-ordinates being modified in order to reflect the change in units.

TimeFrame

The TimeFrame class supports the following System values and associated co-ordinate systems (the default is "MJD"):

- "MJD": Modified Julian Date (d)
- "JD": Julian Date (d)
- "JEPOCH": Julian epoch (yr)
- "BEPOCH": Besselian (yr)

The default value for the Unit attribute for each system is shown in parentheses. Strictly, these systems should not allow changes to be made to the units. For instance, the usual definition of "MJD" and "JD" include the statement that the values will be in units of days. However, AST does allow the use of other units with all the above supported systems (except BEPOCH), on the understanding that conversion to the "correct" units involves nothing more than a simple scaling (1 yr = 365.25 d, 1 d = 24 h, 1 h = 60 min, 1 min = 60 s). Besselian epoch values are defined in terms of tropical years of 365.2422 days, rather than the usual Julian year of 365.25 days. Therefore, to avoid any confusion, the Unit attribute is automatically cleared to "yr" when a System value of BEPOCH System is selected, and an error is reported if any attempt is subsequently made to change the Unit attribute.

TimeOrigin
The zero point for TimeFrame axis values (TimeFrames only)

Description:

This specifies the origin from which all time values are measured within a TimeFrame. The default value (zero) results in the TimeFrame describing absolute time values in the system given by the System attribute (*e.g.* MJD, Julian epoch, *etc*). If a TimeFrame is to be used to describe elapsed time since some origin, the TimeOrigin attribute should be set to hold the required origin value. The TimeOrigin value stored inside the TimeFrame structure is modified whenever TimeFrame attribute values are changed so that it refers to the original moment in time.

Input Formats :

The formats accepted when setting a TimeOrigin value are listed below. They are all case-insensitive and are generally tolerant of extra white space and alternative field delimiters:

- Besselian Epoch: Expressed in decimal years, with or without decimal places ("B1950" or "B1976.13" for example).
- Julian Epoch: Expressed in decimal years, with or without decimal places ("J2000" or "J2100.9" for example).
- Units: An unqualified decimal value is interpreted as a value in the system specified by the TimeFrame's System attribute, in the units given by the TimeFrame's

Unit attribute. Alternatively, an appropriate unit string can be appended to the end of the floating point value ("123.4 d" for example), in which case the supplied value is scaled into the units specified by the Unit attribute.

- Julian Date: With or without decimal places ("JD 2454321.9" for example).
- Modified Julian Date: With or without decimal places ("MJD 54321.4" for example).
- Gregorian Calendar Date: With the month expressed either as an integer or a three-character abbreviation, and with optional decimal places to represent a fraction of a day ("1996-10-2" or "1996-Oct-2.6" for example). If no fractional part of a day is given, the time refers to the start of the day (zero hours).
- Gregorian Date and Time: Any calendar date (as above) but with a fraction of a day expressed as hours, minutes and seconds ("1996-Oct-2 12:13:56.985" for example). The date and time can be separated by a space or by a "T" (as used by ISO8601 format).

Output Format :

When enquiring TimeOrigin values, the returned formatted floating point value represents a value in the TimeFrame's System, in the unit specified by the TimeFrame's Unit attribute.

TimeScale
Time scale (TimeFrames only)

Description:

This attribute identifies the time scale to which the time axis values of a TimeFrame refer, and may take any of the values listed in the "Time Scales" section (below).

The default TimeScale value depends on the current System value; if the current TimeFrame system is "Besselian epoch" the default is "TT", otherwise it is "TAI". Note, if the System attribute is set so that the TimeFrame represents Besselian Epoch, then an error will be reported if an attempt is made to set the TimeScale to anything other than TT.

Note, the supported time scales fall into two groups. The first group containing UT1, GMST, LAST and LMST define time in terms of the orientation of the earth. The second group (containing all the remaining time scales) define time in terms of an atomic process. Since the rate of rotation of the earth varies in an unpredictable way, conversion between two timescales in different groups relies on a value being supplied for the Dut1 attribute. This attribute specifies the difference between the UT1 and UTC time scales, in seconds, and defaults to zero. See the documentation for the Dut1 attribute in SUN/210 for further details.

Time Scales :

The TimeFrame class supports the following TimeScale values (all are case-insensitive):

- "TAI" – International Atomic Time
- "UTC" – Coordinated Universal Time

- "UT1" – Universal Time
- "GMST" – Greenwich Mean Sidereal Time
- "LAST" – Local Apparent Sidereal Time
- "LMST" – Local Mean Sidereal Time
- "TT" – Terrestrial Time
- "TDB" – Barycentric Dynamical Time
- "TCB" – Barycentric Coordinate Time
- "TCG" – Geocentric Coordinate Time
- "LT" – Local Time (the offset from UTC is given by attribute LTOffset)

An very informative description of these and other time scales is available at http://www.ucolick.org/∼sla/leapsecs/timescales.htm.

UTC Warnings :

UTC should ideally be expressed using separate hours, minutes and seconds fields (or at least in seconds for a given date) if leap seconds are to be taken into account. Since the TimeFrame class represents each moment in time using a single floating point number (the axis value) there will be an ambiguity during a leap second. Thus an error of up to 1 second can result when using AST to convert a UTC time to another time scale if the time occurs within a leap second. Leap seconds occur at most twice a year, and are introduced to take account of variation in the rotation of the earth. The most recent leap second occurred on 1st January 1999. Although in the vast majority of cases leap second ambiguities won't matter, there are potential problems in on-line data acquisition systems and in critical applications involving taking the difference between two times.

Title
Frame title

Description:

This attribute holds a string that is used as a title in (*e.g.*) graphical output to describe the co-ordinate system that a Frame represents. Examples might be "Detector Co-ordinates" or "Galactic Co-ordinates".

If a Title value has not been set for a Frame, then a suitable default is supplied.

The default supplied by the Frame class is "<n>-d co-ordinate system", where <n> is the number of Frame axes (Naxes attribute).

Specialised Frames (SkyFrame, SpecFrame, *etc.*) re-define the default Title value to be appropriate to the particular co-ordinate system being represented.

Examples:

```
wcsattrib my_data set Title "My own data"
```

This sets the Title for the current Frame in the NDF called my_data, to the string "My own data".

Notes:

- A Frame's Title is intended purely for interpretation by human readers and not by software.

Unit(axis)
Axis physical units

Description:

This attribute contains a textual representation of the physical units used to represent co-ordinate values on a particular axis of a Frame.

Specialised Frames (SkyFrame, SpecFrame, *etc.*) re-define the default Unit values to be appropriate to the particular co-ordinate system being represented.

For most classes, the Unit attribute is a purely descriptive comment intended for human readers and makes no difference to the operation of the software. However, there are some classes that have *active* Unit attributes. Changing the Unit attribute for such classes will result in the Mappings within the WCS FrameSet being modified in order to reflect the change in units. By default, only SpecFrames have an active Unit attribute.

In general, the syntax of the Unit attribute should follow the recommendations made in the FITS standard (see the paper "Representation of world coordinates in FITS" by Greisen & Calabretta (available at `http://www.cv.nrao.edu/fits/documents/wcs/wcs.html`).

Notes:

- When specifying this attribute by name, it should be subscripted with the number of the Frame axis to which it applies (unless the Frame has only one axis).

E Descriptions of Plotting Attributes

This appendix lists the available plotting attributes (see Section 7). Note, the default value included in each description is the value that will be used if no other default value is supplied (for instance within a defaults file). In particular, the standard KAPPA default style files contained in `$KAPPA_DIR` include alternative default values for several attributes which will be used in preference to those described below.

Individual applications may override the normal behaviour of particular attributes, in which case the description of the application will include details.

It is often useful to specify values for some of the attributes of the current Frame when giving a plotting style (title, axis labels, *etc.*). These additional attributes are described in Appendix D.

Border
Draw a border around valid regions?

Description:

This attribute controls the appearance of annotated axes by determining whether a border is drawn around regions corresponding to the valid co-ordinates.

If the Border value of a plot is non-zero, then this border will be drawn as part of the axes. Otherwise, the border is not drawn (although axis labels and tick marks will still appear, unless other relevant attributes indicate that they should not). The default behaviour is to draw the border if tick marks and numerical labels will be drawn around the edges of the plotting area (see the Labelling attribute), but to omit it otherwise.

Colour(element)
Colour index for a plot element

Description:

This attribute determines the colour index used when drawing each element of of a plot. It takes a separate value for each graphical element so that, for instance, the setting `"Colour(title)=2"` causes the title to be drawn using colour index 2. Standard X colour names, space-separated floating-point RGB triples, or HTML colour codes (optionally using a `"@"` inplace of a `"#"` to avoid the code being interpreted as a comment within a style file) can also be specified, and the nearest available colour will be used if the requested colour is not currently in the palette.

The default behaviour is for all graphical elements to be drawn using Pen 1.

Notes:

- For a list of the graphical elements available, see Section 7.1.
- If no graphical element is specified, (*e.g.* Colour instead of Colour(title)), then all graphical elements are affected.

DrawAxes
Draw axes?

Description:

This attribute controls the appearance of annotated axes by determining whether curves representing co-ordinate axes should be drawn.

If drawn, these axis lines will pass through any tick marks associated with numerical labels drawn to mark values on the axes. The location of these tick marks and labels (and hence the axis lines) is determined by the LabelAt(axis) attribute.

If the DrawAxes value is non-zero (the default), then axis lines will be drawn, otherwise they will be omitted.

Notes:

- Axis lines are drawn independently of any co-ordinate grid lines (see the Grid attribute) so grid lines may be used to substitute for axis lines if required.
- In some circumstances, numerical labels and tick marks are drawn around the edges of the plotting area (see the Labelling attribute). In this case, the value of the DrawAxes attribute is ignored.

DrawDSB
Annotate both sidebands if the spectral axis represents a dual sideband spectrum?

Description:

This attribute controls the appearance of annotated axes by determining what to draw for a spectral axis representing a dual sideband spectrum. The sideband selected using the SideBand attribute will always be annotated on the edge of the Plot selected using the Edge(axis) attribute. In addition, if the DrawDSB attribute is non-zero (the default) the other sideband will be annotated on the opposite edge of the Plot. If DrawDSB is zero, then the other sideband will not be annotated.

DrawTitle
Draw a title?

Description:

This attribute controls the appearance of annotated axes by determining whether a title is drawn.

If the DrawTitle value is non-zero (the default), then the title will be drawn, otherwise it will be omitted.

Notes:

- The text used for the title is obtained from the Title attribute.
- The vertical placement of the title can be controlled using the TitleGap attribute.

Edge(axis)
Which edges to label

Description:

This attribute controls the appearance of annotated axes by determining which edges are used for displaying numerical and descriptive axis labels. It takes a separate value for each physical axis so that, for instance, the setting `Edge(2)=left` specifies which edge to use to display labels for the second axis.

The values `left`, `top`, `right` and `bottom` (or any abbreviation) can be supplied for this attribute. The default is usually `bottom` for the first axis and `left` for the second axis. However, if exterior labelling was requested (see the Labelling attribute) but cannot be produced using these default Edge values, then the default values will be swapped if this enables exterior labelling to be produced.

Notes:

- In some circumstances, numerical labels will be drawn along internal grid lines instead of at the edges of the plotting area (see the Labelling attribute). In this case, the Edge attribute only affects the placement of the descriptive labels (these are drawn at the edges of the plotting area, rather than along the axis lines).

FileInTitle
Include the NDF name as a second line in the title?

Description:

This attribute controls the appearance of annotated axes by determining whether or not the title at the top of the plot will include a second line giving the NDF name. The default value of zero results in the title containing only the text given by the Title attribute. A non-zero value will result in a second line containing the NDF name being added to the title.

Font(element)
Character fount for a plot element

Description:

This attribute determines the character fount index used when drawing each element of a plot. It takes a separate value for graphical element so that, for instance, the setting `Font(title)=2` causes the title to be drawn using fount number 2.

The range of integer fount indices available and the appearance of the resulting text is determined by the PGPLOT graphics package, but include:

(1) A simple single-stroke fount (the default)

(2) A roman fount

(3) An italic fount

(4) A script fount

Notes:

- For a list of the graphical elements available, see Section 7.1.
- If no graphical element is specified, (*e.g.* Font instead of Font(title)), then all graphical elements are affected.

Gap(axis)
Interval between major axis values

Description:

This attribute controls the appearance of annotated axes by determining the interval between the major axis values at which (for example) major tick marks are drawn. It takes a

separate value for each physical axis so that, for instance, the setting `Gap(2)=3.0` specifies the interval between major values along the second axis.

The Gap value supplied will usually be rounded to the nearest 'nice' value, suitable (*e.g.*) for generating axis labels, before use. To avoid this 'nicing' you should set an explicit format for the axis using the Format(axis) or Digits/Digits(axis) attribute. The default behaviour is for the application to generate its own Gap value when required, based on the range of axis values to be represented.

Notes:

- The Gap value should use the same units as are used internally for storing co-ordinate values on the corresponding axis. For example, with a celestial co-ordinate system, the Gap value should be in radians, not hours or degrees.
- If no axis is specified, (*e.g.* Gap instead of Gap(2)), then both axes are affected.

Grid
Draw grid lines?

Description:

This attribute controls the appearance of annotated axes by determining whether grid lines (a grid of curves marking the 'major' values on each axis) are drawn across the plotting area.

If the Grid attribute is non-zero, then grid lines will be drawn. Otherwise, short tick marks on the axes are used to mark the major axis values. The default behaviour is to use tick marks if the entire plotting area is filled by valid co-ordinates, but to draw grid lines otherwise.

Notes:

- The spacing between major axis values, which determines the spacing of grid lines, may be set using the Gap(axis) attribute.

LabelAt(axis)
Where to place numerical labels for a Plot

Description:

This attribute controls the appearance of annotated axes by determining where numerical axis labels and associated tick marks are placed. It takes a separate value for each physical axis so that, for instance, the setting `"LabelAt(2)=10.0"` specifies where the numerical labels and tick marks for the second axis should be drawn.

For each axis, the LabelAt value gives the value on the other axis at which numerical labels and tick marks should be placed. For example, in a celestial (right ascension, declination) co-ordinate system, LabelAt(1) gives a declination value that defines a line (of constant declination) along which the numerical right-ascension labels and their associated tick marks will be drawn. Similarly, LabelAt(2) gives the right-ascension value at which the declination labels and ticks will be drawn.

The default bahaviour is for the application to generate its own position for numerical labels and tick marks.

Notes:

- The LabelAt value should use the same units as are used internally for storing co-ordinate values on the appropriate axis. For example, with a celestial co-ordinate system, the LabelAt value should be in radians, not hours or degrees.
- Normally, the LabelAt value also determines where the lines representing co-ordinate axes will be drawn, so that the tick marks will lie on these lines (but also see the DrawAxes attribute).
- In some circumstances, numerical labels and tick marks are drawn around the edges of the plotting area (see the Labelling attribute). In this case, the value of the LabelAt attribute is ignored.

LabelUnits(axis)
Include unit descriptions in axis labels?

Description:

This attribute controls the appearance of annotated axes by determining whether the descriptive labels drawn for each axis should include a description of the units being used on the axis. It takes a separate value for each physical axis so that, for instance, the setting `"LabelUnits(2)=1"` specifies that a unit description should be included in the label for the second axis.

If the LabelUnits value axis is non-zero, a unit description will be included in the descriptive label for that axis, otherwise it will be omitted. The default behaviour is to include a unit description unless the Frame being annotated is a SkyFrame (*i.e.* a celestial co-ordinate system), in which case it is omitted.

Notes:

- The text used for the unit description is obtained from Unit(axis) attribute.
- If no axis is specified, (*e.g.* LabelUnits instead of LabelUnits(2)), then both axes are affected.

LabelUp(axis)
Draw numerical axis labels upright?

Description:

This attribute controls the appearance of annotated axes by determining whether the numerical labels for each axis should be drawn upright or not. It takes a separate value for each physical axis so that, for instance, the setting `"LabelUp(2)=1"` specifies that numerical labels for the second axis should be drawn upright.

If the LabelUp value axis is non-zero, it causes numerical labels for that axis to be plotted upright (*i.e.* as normal, horizontal text), otherwise (the default) these labels rotate to follow the axis to which they apply.

Notes:

- In some circumstances, numerical labels and tick marks are drawn around the edges of the plotting area (see the Labelling attribute). In this case, the value of the LabelUp attribute is ignored.
- If no axis is specified, (*e.g.* LabelUp instead of LabelUp(2)), then both axes are affected.

Labelling
Label and tick placement option

Description:

This attribute controls the appearance of annotated axes by determining the strategy for placing numerical labels and tick marks.

If the Labelling value is `"exterior"` (the default), then numerical labels and their associated tick marks are placed around the edges of the plotting area, if possible. If this is not possible, or if the Labelling value is `"interior"`, then they are placed along grid lines inside the plotting area.

Notes:

- The LabelAt(axis) attribute may be used to determine the exact placement of labels and tick marks that are drawn inside the plotting area.

MajTickLen
Length of major tick marks

Description:

This attribute controls the appearance of annotated axes by determining the length of the major tick marks drawn on the axes.

The MajTickLen value should be given as a fraction of the minimum dimension of the plotting area. Negative values cause major tick marks to be placed on the outside of the corresponding grid line or axis (but subject to any clipping imposed by PGPLOT), while positive values cause them to be placed on the inside.

The default behaviour depends on whether a co-ordinate grid is drawn inside the plotting area (see the Grid attribute). If so, the default MajTickLen value is zero (so that major ticks are not drawn), otherwise the default is $+0.015$.

MinTick(axis)
Density of minor tick marks

Description:

This attribute controls the appearance of annotated axes by determining the density of minor tick marks which appear between the major axis values. It takes a separate value for each physical axis so that, for instance, the setting `"MinTick(2)=2"` specifies the density of minor tick marks along the second axis.

The value supplied should be the number of minor divisions required between each pair of major axis values, this being one more than the number of minor tick marks to be drawn. By default, a value is chosen that depends on the gap between major axis values and the nature of the axis.

Notes:

- If no axis is specified, (*e.g.* MinTick instead of MinTick(2)), then both axes are affected.

MinTickLen
Length of minor tick marks

Description:

This attribute controls the appearance of annotated axes by determining the length of the minor tick marks drawn on the axes.

The MinTickLen value should be given as a fraction of the minimum dimension of the plotting area. Negative values cause minor tick marks to be placed on the outside of the corresponding grid line or axis (but subject to any clipping imposed by the underlying graphics system), while positive values cause them to be placed on the inside.

The default value is +0.007.

Notes:

- The number of minor tick marks drawn is determined by the MinTick(axis) attribute.

NumLab(axis)
Draw numerical axis labels?

Description:

This attribute controls the appearance of annotated axes by determining whether labels should be drawn to represent the numerical values along each axis. It takes a separate value for each physical axis so that, for instance, the setting `"NumLab(2)=1"` specifies that numerical labels should be drawn for the second axis.

If the NumLab value for an axis is non-zero (the default), then numerical labels will be drawn for that axis, otherwise they will be omitted.

Notes:

- The drawing of associated descriptive axis labels (describing the quantity being plotted along each axis) is controlled by the TextLab(axis) attribute.
- If no axis is specified, (*e.g.* NumLab instead of NumLab(2)), then both axes are affected.

NumLabGap(axis)
Spacing of numerical labels

Description:

This attribute controls the appearance of annotated axes by determining where numerical axis labels are placed relative to the axes they describe. It takes a separate value for each physical axis so that, for instance, the setting `"NumLabGap(2)=-0.01"` specifies where the numerical label for the second axis should be drawn.

For each axis, the NumLabGap value gives the spacing between the axis line (or edge of the plotting area, if appropriate) and the nearest edge of the corresponding numerical axis labels. Positive values cause the descriptive label to be placed on the opposite side of the line to the default tick marks, while negative values cause it to be placed on the same side.

The NumLabGap value should be given as a fraction of the minimum dimension of the plotting area, the default value being +0.01.

Notes:

- If no axis is specified, (*e.g.* NumLabGap instead of NumLabGap(2)), then both axes are affected.

Size(element)
Character size for a plot element

Description:

This attribute determines the character size used when drawing each element of graphical output. It takes a separate value for each graphical element so that, for instance, the setting `"Size(title)=2.0"` causes the plot title to be drawn using twice the default character size.

Notes:

- For a list of the graphical elements available, see Section 7.1.
- If no graphical element is specified, (*e.g.* Size instead of Size(title)), then all graphical elements are affected.

Style(element)
Line style for a plot element

Description:

This attribute determines the line style used when drawing each element of graphical output. It takes a separate value for each graphical element so that, for instance, the setting `"Style(border)=2"` causes the border to be drawn using line Style 2 (which results in a dashed line).

The range of integer line styles available and their appearance is determined by the PGPLOT graphics system, but includes the following:

(1) Full line (the default)
(2) Dashed
(3) Dot-dash-dot-dash
(4) Dotted
(5) Dash-dot-dot-dot

Notes:

- For a list of the graphical elements available, see Section 7.1.
- If no graphical element is specified, (*e.g.* Style instead of Style(border)), then all graphical elements are affected.

TextBackColour
The background colour to use when drawing text

Description:

This attribute determines the background colour index to use when drawing textual strings—the foreground colour is determined by the Colour(strings) attribute. Standard X colour names can also be specified, and the nearest colour will be used if the requested colour is not currently in the palette.

If the value `"-1"` or `"clear"` is given, the background will be transparent.

TextLab(axis)
Draw descriptive axis labels?

Description:

This attribute controls the appearance of annotated axes by determining whether textual labels should be drawn to describe the quantity being represented on each axis. It takes a separate value for each physical axis so that, for instance, the setting `"TextLab(2)=1"` specifies that descriptive labels should be drawn for the second axis.

If the TextLab value of an axis is non-zero, then descriptive labels will be drawn for that axis, otherwise they will be omitted. The default behaviour is to draw descriptive labels if tick marks and numerical labels are being drawn around the edges of the plotting area (see the Labelling attribute), but to omit them otherwise.

Notes:

- The text used for the descriptive labels is derived from the Label(axis) attribute, together with its Unit(axis) attribute if appropriate (see the LabelUnits(axis) attribute).
- The drawing of numerical axis labels (which indicate values on the axis) is controlled by the NumLab(axis) attribute.
- If no axis is specified, (*e.g.* TextLab instead of TextLab(2)), then both axes are affected.

TextLabGap(axis)
Spacing of descriptive axis labels

Description:

This attribute controls the appearance of annotated axes by determining where descriptive axis labels are placed relative to the axes they describe. It takes a separate value for each physical axis so that, for instance, the setting `"TextLabGap(2)=0.01"` specifies where the descriptive label for the second axis should be drawn.

For each axis, the TextLabGap value gives the spacing between the descriptive label and the edge of a box enclosing all other parts of the annotated grid (excluding other descriptive labels). The gap is measured to the nearest edge of the label (*i.e.* the top or the bottom). Positive values cause the descriptive label to be placed outside the bounding box, while negative values cause it to be placed inside.

The TextLabGap value should be given as a fraction of the minimum dimension of the plotting area, the default value being +0.01.

Notes:

- If drawn, descriptive labels are always placed at the edges of the plotting area, even although the corresponding numerical labels may be drawn along axis lines in the interior of the plotting area (see the Labelling attribute).
- If no axis is specified, (*e.g.* TextLabGap instead of TextLabGap(2)), then both axes are affected.

TextMargin
The width of the margin to clear when drawing text

Description:

This attribute determines the width of the margin that is cleared around the edges of each drawn text string, in units of the text height. That is, if TEXTMARGIN is set to 0.5, the margin will be half the height of the text. The default is zero.

TickAll
Draw tick marks on all edges?

Description:

This attribute controls the appearance of annotated axes by determining whether tick marks should be drawn on all edges of the plot.

If the TickAll value is non-zero (the default), then tick marks will be drawn on all edges of the plot. Otherwise, they will be drawn only on those edges where the numerical and descriptive axis labels are drawn (see the Edge(axis) attribute).

Notes:

- In some circumstances, numerical labels and tick marks are drawn along grid lines inside the plotting area, rather than around its edges (see the Labelling attribute). In this case, the value of the TickAll attribute is ignored.

TitleGap
Vertical spacing for the title

Description:

This attribute controls the appearance of annotated axes by determining where the title is drawn.

Its value gives the spacing between the bottom edge of the title and the top edge of a bounding box containing all the other parts of the annotated grid. Positive values cause the title to be drawn outside the box, while negative values cause it to be drawn inside.

The TitleGap value should be given as a fraction of the minimum dimension of the plotting area, the default value being $+0.05$.

Notes:

- The text used for the title is obtained from the Title attribute.

Tol
Plotting tolerance

Description:

This attribute specifies the plotting tolerance (or resolution) to be used for curves. Smaller values will result in smoother and more accurate curves being drawn (particularly in the presence of discontinuities in the mapping between graphics co-ordinates and physical co-ordinates), but may slow down the plotting process. Conversely, larger values may speed up the plotting process in cases where high resolution is not required.

The Tol value should be given as a fraction of the minimum dimension of the plotting area, and should lie in the range from $1.0E{-}7$ to 1.0. By default, a value of 0.001 is used.

Width(element)
Line width for a plot element

Description:

This attribute determines the line width used when drawing each element of graphical output. It takes a separate value for each graphical element so that, for instance, the setting `"Width(border)=2.0"` causes the border to be drawn using a line width of 2.0. A value of 1.0 results in a line thickness that is approximately 0.0005 times the length of the diagonal of the entire display surface.

Notes:

- For a list of the graphical elements available, see Section 7.1.
- If no graphical element is specified, (*e.g.* Width instead of Width(title)), then all graphical elements are affected.

F Standard Named Colours

The standard set of named colours recognised by KAPPA is tabulated below together with their red, green, and blue relative intensities. It is the X-windows standard colour set so don't blame KAPPA if you think some of them are anomalous. In addition to those tabulated, there are grey levels at each percentage between "Black" and "White". These are called "Grey1", "Grey2", . . . , "Grey99". All the names containing "Grey" have synonyms spelt with "Gray".

Standard Colour Set							
Name	R	G	B	Name	R	G	B
AliceBlue	0.941	0.973	1.000	AntiqueWhite	0.980	0.922	0.843
AntiqueWhite1	1.000	0.937	0.859	AntiqueWhite2	0.933	0.875	0.800
AntiqueWhite3	0.804	0.753	0.690	AntiqueWhite4	0.545	0.514	0.471
Aquamarine	0.498	1.000	0.831	Aquamarine1	0.498	1.000	0.831
Aquamarine2	0.463	0.933	0.776	Aquamarine3	0.400	0.804	0.667
Aquamarine4	0.271	0.545	0.455	Azure	0.941	1.000	1.000
Azure1	0.941	1.000	1.000	Azure2	0.878	0.933	0.933
Azure3	0.757	0.804	0.804	Azure4	0.514	0.545	0.545
Beige	0.961	0.961	0.863	Bisque	1.000	0.894	0.769
Bisque1	1.000	0.894	0.769	Bisque2	0.933	0.835	0.718
Bisque3	0.804	0.718	0.620	Bisque4	0.545	0.490	0.420
Black	0.000	0.000	0.000	BlanchedAlmond	1.000	0.922	0.804
Blue	0.000	0.000	1.000	Blue1	0.000	0.000	1.000
Blue2	0.000	0.000	0.933	Blue3	0.000	0.000	0.804
Blue4	0.000	0.000	0.545	BlueViolet	0.541	0.169	0.886
Brown	0.647	0.165	0.165	Brown1	1.000	0.251	0.251
Brown2	0.933	0.231	0.231	Brown3	0.804	0.200	0.200
Brown4	0.545	0.137	0.137	Burlywood	0.871	0.722	0.529
Burlywood1	1.000	0.827	0.608	Burlywood2	0.933	0.773	0.569
Burlywood3	0.804	0.667	0.490	Burlywood4	0.545	0.451	0.333
CadetBlue	0.373	0.620	0.627	CadetBlue1	0.596	0.961	1.000
CadetBlue2	0.557	0.898	0.933	CadetBlue3	0.478	0.773	0.804
CadetBlue4	0.325	0.525	0.545	Chartreuse	0.498	1.000	0.000
Chartreuse1	0.498	1.000	0.000	Chartreuse2	0.463	0.933	0.000
Chartreuse3	0.400	0.804	0.000	Chartreuse4	0.271	0.545	0.000
Chocolate	0.824	0.412	0.118	Chocolate1	1.000	0.498	0.141
Chocolate2	0.933	0.463	0.129	Chocolate3	0.804	0.400	0.114
Chocolate4	0.545	0.271	0.075	Coral	1.000	0.498	0.314
Coral1	1.000	0.447	0.337	Coral2	0.933	0.416	0.314
Coral3	0.804	0.357	0.271	Coral4	0.545	0.243	0.184
CornflowerBlue	0.392	0.584	0.929	Cornsilk	1.000	0.973	0.863
Cornsilk1	1.000	0.973	0.863	Cornsilk2	0.933	0.910	0.804

Standard Colour Set							
Name	R	G	B	Name	R	G	B
DarkGoldenrod3	0.804	0.584	0.047	DarkGoldenrod4	0.545	0.396	0.031
DarkGreen	0.000	0.392	0.000	DarkKhaki	0.741	0.718	0.420
DarkOliveGreen	0.333	0.420	0.184	DarkOliveGreen1	0.792	1.000	0.439
DarkOliveGreen2	0.737	0.933	0.408	DarkOliveGreen3	0.635	0.804	0.353
DarkOliveGreen4	0.431	0.545	0.239	DarkOrange	1.000	0.549	0.000
DarkOrange1	1.000	0.498	0.000	DarkOrange2	0.933	0.463	0.000
DarkOrange3	0.804	0.400	0.000	DarkOrange4	0.545	0.271	0.000
DarkOrchid	0.600	0.196	0.800	DarkOrchid1	0.749	0.243	1.000
DarkOrchid2	0.698	0.227	0.933	DarkOrchid3	0.604	0.196	0.804
DarkOrchid4	0.408	0.133	0.545	DarkSalmon	0.914	0.588	0.478
DarkSeaGreen	0.561	0.737	0.561	DarkSeaGreen1	0.757	1.000	0.757
DarkSeaGreen2	0.706	0.933	0.706	DarkSeaGreen3	0.608	0.804	0.608
DarkSeaGreen4	0.412	0.545	0.412	DarkSlateBlue	0.282	0.239	0.545
DarkSlateGrey	0.184	0.310	0.310	DarkSlateGrey1	0.592	1.000	1.000
DarkSlateGrey2	0.553	0.933	0.933	DarkSlateGrey3	0.475	0.804	0.804
DarkSlateGrey4	0.322	0.545	0.545	DarkTurquoise	0.000	0.808	0.820
DarkViolet	0.580	0.000	0.827	DeepPink	1.000	0.078	0.576
DeepPink1	1.000	0.078	0.576	DeepPink2	0.933	0.071	0.537
DeepPink3	0.804	0.063	0.463	DeepPink4	0.545	0.039	0.314
DeepSkyBlue	0.000	0.749	1.000	DeepSkyBlue1	0.000	0.749	1.000
DeepSkyBlue2	0.000	0.698	0.933	DeepSkyBlue3	0.000	0.604	0.804
DeepSkyBlue4	0.000	0.408	0.545	DimGrey	0.412	0.412	0.412
DodgerBlue	0.118	0.565	1.000	DodgerBlue1	0.118	0.565	1.000
DodgerBlue2	0.110	0.525	0.933	DodgerBlue3	0.094	0.455	0.804
DodgerBlue4	0.063	0.306	0.545	Firebrick	0.698	0.133	0.133
Firebrick1	1.000	0.188	0.188	Firebrick2	0.933	0.173	0.173
Firebrick3	0.804	0.149	0.149	Firebrick4	0.545	0.102	0.102
FloralWhite	1.000	0.980	0.941	ForestGreen	0.133	0.545	0.133
Gainsboro	0.863	0.863	0.863	GhostWhite	0.973	0.973	1.000
Gold	1.000	0.843	0.000	Gold1	1.000	0.843	0.000
Gold2	0.933	0.788	0.000	Gold3	0.804	0.678	0.000
Gold4	0.545	0.459	0.000	Goldenrod	0.855	0.647	0.125

Standard Colour Set							
Name	R	G	B	Name	R	G	B
Khaki	0.941	0.902	0.549	Khaki1	1.000	0.965	0.561
Khaki2	0.933	0.902	0.522	Khaki3	0.804	0.776	0.451
Khaki4	0.545	0.525	0.306	Lavender	0.902	0.902	0.980
LavenderBlush	1.000	0.941	0.961	LavenderBlush1	1.000	0.941	0.961
LavenderBlush2	0.933	0.878	0.898	LavenderBlush3	0.804	0.757	0.773
LavenderBlush4	0.545	0.514	0.525	LawnGreen	0.486	0.988	0.000
LemonChiffon	1.000	0.980	0.804	LemonChiffon1	1.000	0.980	0.804
LemonChiffon2	0.933	0.914	0.749	LemonChiffon3	0.804	0.788	0.647
LemonChiffon4	0.545	0.537	0.439	LightBlue	0.678	0.847	0.902
LightBlue1	0.749	0.937	1.000	LightBlue2	0.698	0.875	0.933
LightBlue3	0.604	0.753	0.804	LightBlue4	0.408	0.514	0.545
LightCoral	0.941	0.502	0.502	LightCyan	0.878	1.000	1.000
LightCyan1	0.878	1.000	1.000	LightCyan2	0.820	0.933	0.933
LightCyan3	0.706	0.804	0.804	LightCyan4	0.478	0.545	0.545
LightGoldenrod	0.933	0.867	0.510	LightGoldenrod1	1.000	0.925	0.545
LightGoldenrod2	0.933	0.863	0.510	LightGoldenrod3	0.804	0.745	0.439
LightGoldenrod4	0.545	0.506	0.298	LightGoldenrodYellow	0.980	0.980	0.824
LightGrey	0.827	0.827	0.827	LightPink	1.000	0.714	0.757
LightPink1	1.000	0.682	0.725	LightPink2	0.933	0.635	0.678
LightPink3	0.804	0.549	0.584	LightPink4	0.545	0.373	0.396
LightSalmon	1.000	0.627	0.478	LightSalmon1	1.000	0.627	0.478
LightSalmon2	0.933	0.584	0.447	LightSalmon3	0.804	0.506	0.384
LightSalmon4	0.545	0.341	0.259	LightSeaGreen	0.125	0.698	0.667
LightSkyBlue	0.529	0.808	0.980	LightSkyBlue1	0.690	0.886	1.000
LightSkyBlue2	0.643	0.827	0.933	LightSkyBlue3	0.553	0.714	0.804
LightSkyBlue4	0.376	0.482	0.545	LightSlateBlue	0.518	0.439	1.000
LightSlateGrey	0.467	0.533	0.600	LightSteelBlue	0.690	0.769	0.871
LightSteelBlue1	0.792	0.882	1.000	LightSteelBlue2	0.737	0.824	0.933
LightSteelBlue3	0.635	0.710	0.804	LightSteelBlue4	0.431	0.482	0.545
LightYellow	1.000	1.000	0.878	LightYellow1	1.000	1.000	0.878
LightYellow2	0.933	0.933	0.820	LightYellow3	0.804	0.804	0.706
LightYellow4	0.545	0.545	0.478	LimeGreen	0.196	0.804	0.196

Standard Colour Set							
Name	R	G	B	Name	R	G	B
MistyRose	1.000	0.894	0.882	MistyRose1	1.000	0.894	0.882
MistyRose2	0.933	0.835	0.824	MistyRose3	0.804	0.718	0.710
MistyRose4	0.545	0.490	0.482	Moccasin	1.000	0.894	0.710
NavajoWhite	1.000	0.871	0.678	NavajoWhite1	1.000	0.871	0.678
NavajoWhite2	0.933	0.812	0.631	NavajoWhite3	0.804	0.702	0.545
NavajoWhite4	0.545	0.475	0.369	Navy	0.000	0.000	0.502
NavyBlue	0.000	0.000	0.502	OldLace	0.992	0.961	0.902
OliveDrab	0.420	0.557	0.137	OliveDrab1	0.753	1.000	0.243
OliveDrab2	0.702	0.933	0.227	OliveDrab3	0.604	0.804	0.196
OliveDrab4	0.412	0.545	0.133	Orange	1.000	0.647	0.000
Orange1	1.000	0.647	0.000	Orange2	0.933	0.604	0.000
Orange3	0.804	0.522	0.000	Orange4	0.545	0.353	0.000
OrangeRed	1.000	0.271	0.000	OrangeRed1	1.000	0.271	0.000
OrangeRed2	0.933	0.251	0.000	OrangeRed3	0.804	0.216	0.000
OrangeRed4	0.545	0.145	0.000	Orchid	0.855	0.439	0.839
Orchid1	1.000	0.514	0.980	Orchid2	0.933	0.478	0.914
Orchid3	0.804	0.412	0.788	Orchid4	0.545	0.278	0.537
PaleGoldenrod	0.933	0.910	0.667	PaleGreen	0.596	0.984	0.596
PaleGreen1	0.604	1.000	0.604	PaleGreen2	0.565	0.933	0.565
PaleGreen3	0.486	0.804	0.486	PaleGreen4	0.329	0.545	0.329
PaleTurquoise	0.686	0.933	0.933	PaleTurquoise1	0.733	1.000	1.000
PaleTurquoise2	0.682	0.933	0.933	PaleTurquoise3	0.588	0.804	0.804
PaleTurquoise4	0.400	0.545	0.545	PaleVioletRed	0.859	0.439	0.576
PaleVioletRed1	1.000	0.510	0.671	PaleVioletRed2	0.933	0.475	0.624
PaleVioletRed3	0.804	0.408	0.537	PaleVioletRed4	0.545	0.278	0.365
PapayaWhip	1.000	0.937	0.835	PeachPuff	1.000	0.855	0.725
PeachPuff1	1.000	0.855	0.725	PeachPuff2	0.933	0.796	0.678
PeachPuff3	0.804	0.686	0.584	PeachPuff4	0.545	0.467	0.396
Peru	0.804	0.522	0.247	Pink	1.000	0.753	0.796
Pink1	1.000	0.710	0.773	Pink2	0.933	0.663	0.722
Pink3	0.804	0.569	0.620	Pink4	0.545	0.388	0.424
Plum	0.867	0.627	0.867	Plum1	1.000	0.733	1.000

Standard Colour Set							
Name	R	G	B	Name	R	G	B
SandyBrown	0.957	0.643	0.376	SeaGreen	0.180	0.545	0.341
SeaGreen1	0.329	1.000	0.624	SeaGreen2	0.306	0.933	0.580
SeaGreen3	0.263	0.804	0.502	SeaGreen4	0.180	0.545	0.341
Seashell	1.000	0.961	0.933	Seashell1	1.000	0.961	0.933
Seashell2	0.933	0.898	0.871	Seashell3	0.804	0.773	0.749
Seashell4	0.545	0.525	0.510	Sienna	0.627	0.322	0.176
Sienna1	1.000	0.510	0.278	Sienna2	0.933	0.475	0.259
Sienna3	0.804	0.408	0.224	Sienna4	0.545	0.278	0.149
SkyBlue	0.529	0.808	0.922	SkyBlue1	0.529	0.808	1.000
SkyBlue2	0.494	0.753	0.933	SkyBlue3	0.424	0.651	0.804
SkyBlue4	0.290	0.439	0.545	SlateBlue	0.416	0.353	0.804
SlateBlue1	0.514	0.435	1.000	SlateBlue2	0.478	0.404	0.933
SlateBlue3	0.412	0.349	0.804	SlateBlue4	0.278	0.235	0.545
SlateGrey	0.439	0.502	0.565	SlateGrey1	0.776	0.886	1.000
SlateGrey2	0.725	0.827	0.933	SlateGrey3	0.624	0.714	0.804
SlateGrey4	0.424	0.482	0.545	Snow	1.000	0.980	0.980
Snow1	1.000	0.980	0.980	Snow2	0.933	0.914	0.914
Snow3	0.804	0.788	0.788	Snow4	0.545	0.537	0.537
SpringGreen	0.000	1.000	0.498	SpringGreen1	0.000	1.000	0.498
SpringGreen2	0.000	0.933	0.463	SpringGreen3	0.000	0.804	0.400
SpringGreen4	0.000	0.545	0.271	SteelBlue	0.275	0.510	0.706
SteelBlue1	0.388	0.722	1.000	SteelBlue2	0.361	0.675	0.933
SteelBlue3	0.310	0.580	0.804	SteelBlue4	0.212	0.392	0.545
Tan	0.824	0.706	0.549	Tan1	1.000	0.647	0.310
Tan2	0.933	0.604	0.286	Tan3	0.804	0.522	0.247
Tan4	0.545	0.353	0.169	Thistle	0.847	0.749	0.847
Thistle1	1.000	0.882	1.000	Thistle2	0.933	0.824	0.933
Thistle3	0.804	0.710	0.804	Thistle4	0.545	0.482	0.545
Tomato	1.000	0.388	0.278	Tomato1	1.000	0.388	0.278
Tomato2	0.933	0.361	0.259	Tomato3	0.804	0.310	0.224
Tomato4	0.545	0.212	0.149	Turquoise	0.251	0.878	0.816
Turquoise1	0.000	0.961	1.000	Turquoise2	0.000	0.898	0.933

G Using MathMaps

A MathMap is an AST Object which contains a recipe for transforming positions from one co-ordinate Frame to another and (optionally) back again. These transformations are specified using arithmetic operations and mathematical functions similar to those available in Fortran. WCSADD can be used to create a MathMap and store it in a text file, and such text files can then be used by applications such as REGRID which require a mapping.

G.1 Defining Transformation Functions

A MathMap's transformation functions are defined by a set of character strings holding Fortran-like expressions. If the required Mapping has Nin input axes and Nout output axes, you would normally supply the Nout expressions for the forward transformation. For instance, if Nout is 2 you might use:

```
'R = SQRT( X * X + Y * Y )'
'THETA = ATAN2( Y, X )'
```

which defines a transformation from Cartesian to polar co-ordinates. Here, the variables that appear on the left of each expression (R and THETA) provide names for the output variables and those that appear on the right (X and Y) are references to input variables.

To complement this, you must also supply expressions for the inverse transformation. In this case, the number of expressions given would normally match the number of MathMap input co-ordinates, Nin. If Nin is 2, you might use:

```
'X = R $*$ COS( THETA )'
'Y = R $*$ SIN( THETA )'
```

which expresses the transformation from polar to Cartesian co-ordinates. Note that here the input variables (X and Y) are named on the left of each expression, and the output variables (R and THETA) are referenced on the right.

Normally, you cannot refer to a variable on the right of an expression unless it is named on the left of an expression in the complementary set of functions. Therefore both sets of functions (forward and inverse) must be formulated using the same consistent set of variable names. This means that if you wish to leave one of the transformations undefined, you must supply dummy expressions which simply name each of the output (or input) variables. For example, you might use:

```
'X'
'Y'
```

for the inverse transformation above, which serves to name the input variables but without defining an inverse transformation.

G.2 Calculating Intermediate Values

It is sometimes useful to calculate intermediate values and then to use these in the final expressions for the output (or input) variables. This may be done by supplying additional expressions for the forward (or inverse) transformation functions. For instance, the following array of five expressions describes two-dimensional pin-cushion distortion:

```
'R = SQRT( XIN $*$ XIN $+$ YIN $*$ YIN )'
'ROUT = R $*$ ( 1 $+$ 0.1 $*$ R $*$ R )'
'THETA = ATAN2( YIN, XIN )',
'XOUT = ROUT $*$ COS( THETA )'
'YOUT = ROUT $*$ SIN( THETA )'
```

Here, we first calculate three intermediate results (*R*, *ROUT* and *THETA*) and then use these to calculate the final results (*XOUT* and *YOUT*). The MathMap knows that only the final two results constitute values for the output variables because its Nout attribute is set to 2. You may define as many intermediate variables in this way as you choose. Having defined a variable, you may then refer to it on the right of any subsequent expressions.

Note that when defining the inverse transformation you may only refer to the output variables *XOUT* and *YOUT*. The intermediate variables *R*, *ROUT* and *THETA* (above) are private to the forward transformation and may not be referenced by the inverse transformation. The inverse transformation may, however, define its own private intermediate variables.

G.3 Expression Syntax

The expressions given for the forward and inverse transformations closely follow the syntax of Fortran (with some extensions for compatibility with the C language). They may contain references to variables and literal constants, together with arithmetic, logical, relational and bitwise operators, and function invocations. A set of symbolic constants is also available. Each of these is described in detail below. Parentheses may be used to override the normal order of evaluation. There is no built-in limit to the length of expressions and they are insensitive to case or the presence of additional white space.

G.4 Variables

Variable names must begin with an alphabetic character and may contain only alphabetic characters, digits, and the underscore character `"_"`. There is no built-in limit to the length of variable names.

G.5 Literal Constants

Literal constants, such as `"0"`, `"1"`, `"0.007"` or `"2.505E-16"` may appear in expressions, with the decimal point and exponent being optional (a `"D"` may also be used as an exponent character). A unary minus `"-"` may be used as a prefix.

G.6 Arithmetic Precision

All arithmetic is floating point, performed in double precision.

G.7 Propagation of Missing Data

Unless indicated otherwise, if any argument of a function or operator has a bad value (indicating missing data), then the result of that function or operation is also bad, so that such values are propagated automatically through all operations performed by MathMap transformations. The special value used to flag bad values can be represented in expressions by the symbolic constant "<bad>".

A <bad> result is also produced in response to any numerical error (such as division by zero or numerical overflow), or if an invalid argument value is provided to a function or operator.

G.8 Arithmetic Operators

The following arithmetic operators are available:

- X1 + X2: Sum of X1 and X2.
- X1 - X2: Difference of X1 and X2.
- X1 * X2: Product of X1 and X2.
- X1 / X2: Ratio of X1 and X2.
- X1 ** X2: X1 raised to the power of X2.
- + X: Unary plus, has no effect on its argument.
- - X: Unary minus, negates its argument.

G.9 Logical Operators

Logical values are represented using zero to indicate .FALSE. And non-zero to indicate .TRUE.. In addition, the bad value is taken to mean `"unknown"`. The values returned by logical operators may therefore be 0, 1 or bad. Where appropriate, `"tri-state"` logic is implemented. For example, A.OR.B may evaluate to 1 if A is non-zero, even if B has the bad value. This is because the result of the operation would not be affected by the value of B, so long as A is non-zero.

The following logical operators are available:

- X1 .AND. X2: Logical AND between X1 and X2, returning 1 if both X1 and X2 are non-zero, and 0 otherwise. This operator implements tri-state logic. (The synonym `"&&"` is also provided for compatibility with C.)
- X1 .OR. X2: Logical OR between X1 and X2, returning 1 if either X1 or X2 are non-zero, and 0 otherwise. This operator implements tri-state logic. (The synonym `"||"` is also provided for compatibility with C.)
- X1 .NEQV. X2: Logical exclusive OR (XOR) between X1 and X2, returning 1 if exactly one of X1 and X2 is non-zero, and 0 otherwise. Tri-state logic is not used with this operator. (The synonym `".XOR."` is also provided, although this is not standard Fortran. In addition, the C-like synonym `"^^"` may be used, although this is also not standard.)

- X1 .EQV. X2: Tests whether the logical states of X1 and X2 (*i.e.* .TRUE./.FALSE.) are equal. It is the negative of the exclusive OR (XOR) function. Tri-state logic is not used with this operator.
- .NOT. X: Logical unary NOT operation, returning 1 if X is zero, and 0 otherwise. (The synonym "!" is also provided for compatibility with C.)

G.10 Relational Operators

Relational operators return the logical result (0 or 1) of comparing the values of two floating-point values for equality or inequality. The bad value may also be returned if either argument is <bad>.

The following relational operators are available:

- X1 .EQ. X2: Tests whether X1 equals X2. (The synonym "==" is also provided for compatibility with C.)
- X1 .NE. X2: Tests whether X1 is unequal to X2. (The synonym "!=" is also provided for compatibility with C.)
- X1 .GT. X2: Tests whether X1 is greater than X2. (The synonym ">" is also provided for compatibility with C.)
- X1 .GE. X2: Tests whether X1 is greater than or equal to X2. (The synonym ">=" is also provided for compatibility with C.)
- X1 .LT. X2: Tests whether X1 is less than X2. (The synonym "<" is also provided for compatibility with C.)
- X1 .LE. X2: Tests whether X1 is less than or equal to X2. (The synonym "<=" is also provided for compatibility with C.)

Note that relational operators cannot usefully be used to compare values with the <bad> value (representing missing data), because the result is always <bad>. The ISBAD() function should be used instead.

Note, also, that because logical operators can operate on floating point values, care must be taken to use parentheses in some cases where they would not normally be required in Fortran. For example, the expression:

```
.NOT.  A .EQ. B
```

must be written:

```
.NOT. ( A .EQ. B )
```

to prevent the .NOT. Operator from associating with the variable A.

G.11 Bitwise Operators

Bitwise operators are often useful when operating on raw data (*e.g.* from instruments), so they are provided for use in MathMap expressions. In this case, however, the values on which they operate are floating-point values rather than the more usual pure integers. In order to produce results which match the pure integer case, the operands are regarded as fixed point binary numbers (*i.e.* with the binary equivalent of a decimal point) with negative numbers represented using twos-complement notation. For integer values, the resulting bit pattern corresponds to that of the equivalent signed integer (digits to the right of the point being zero). Operations on the bits representing the fractional part are also possible, however.

The following bitwise operators are available:

- X1 >> X2: Rightward bit shift. The integer value of X2 is taken (rounding towards zero) and the bits representing X1 are then shifted this number of places to the right (or to the left if the number of places is negative). This is equivalent to dividing X1 by the corresponding power of 2.
- X1 << X2: Leftward bit shift. The integer value of X2 is taken (rounding towards zero), and the bits representing X1 are then shifted this number of places to the left (or to the right if the number of places is negative). This is equivalent to multiplying X1 by the corresponding power of 2.
- X1 & X2: Bitwise AND between the bits of X1 and those of X2 (equivalent to a logical AND applied at each bit position in turn).
- X1 | X2: Bitwise OR between the bits of X1 and those of X2 (equivalent to a logical OR applied at each bit position in turn).
- X1 ^ X2: Bitwise exclusive OR (XOR) between the bits of X1 and those of X2 (equivalent to a logical XOR applied at each bit position in turn).

Note that no bit inversion operator is provided. This is because inverting the bits of a twos-complement fixed point binary number is equivalent to simply negating it. This differs from the pure integer case because bits to the right of the binary point are also inverted. To invert only those bits to the left of the binary point, use a bitwise exclusive OR with the value -1 (*i.e.* X^-1).

G.12 Functions

The following functions are available:

- ABS(X): Absolute value of X (sign removal), same as FABS(X).
- ACOS(X): Inverse cosine of X, in radians.
- ACOSD(X): Inverse cosine of X, in degrees.
- ACOSH(X): Inverse hyperbolic cosine of X.
- ACOTH(X): Inverse hyperbolic cotangent of X.
- ACSCH(X): Inverse hyperbolic cosecant of X.

- AINT(X): Integer part of X (round towards zero), same as INT(X).
- ASECH(X): Inverse hyperbolic secant of X.
- ASIN(X): Inverse sine of X, in radians.
- ASIND(X): Inverse sine of X, in degrees.
- ASINH(X): Inverse hyperbolic sine of X.
- ATAN(X): Inverse tangent of X, in radians.
- ATAND(X): Inverse tangent of X, in degrees.
- ATANH(X): Inverse hyperbolic tangent of X.
- ATAN2(X1, X2): Inverse tangent of X1/X2, in radians.
- ATAN2D(X1, X2): Inverse tangent of X1/X2, in degrees.
- CEIL(X): Smallest integer value not less then X (round towards plus infinity).
- COS(X): Cosine of X in radians.
- COSD(X): Cosine of X in degrees.
- COSH(X): Hyperbolic cosine of X.
- COTH(X): Hyperbolic cotangent of X.
- CSCH(X): Hyperbolic cosecant of X.
- DIM(X1, X2): Returns X1-X2 if X1 is greater than X2, otherwise 0.
- EXP(X): Exponential function of X.
- FABS(X): Absolute value of X (sign removal), same as ABS(X).
- FLOOR(X): Largest integer not greater than X (round towards minus infinity).
- FMOD(X1, X2): Remainder when X1 is divided by X2, same as MOD(X1, X2).
- GAUSS(X1, X2): Random sample from a Gaussian distribution with mean X1 and standard deviation X2.
- INT(X): Integer part of X (round towards zero), same as AINT(X).
- ISBAD(X): Returns 1 if X has the <bad> value, otherwise 0.
- LOG(X): Natural logarithm of X.
- LOG10(X): Logarithm of X to base 10.
- MAX(X1, X2, ...): Maximum of two or more values.
- MIN(X1, X2, ...): Minimum of two or more values.
- MOD(X1, X2): Remainder when X1 is divided by X2, same as FMOD(X1, X2).

- NINT(X): Nearest integer to X (round to nearest).
- POISSON(X): Random integer-valued sample from a Poisson distribution with mean X.
- POW(X1, X2): X1 raised to the power of X2.
- RAND(X1, X2): Random sample from a uniform distribution in the range X1 to X2 inclusive.
- SECH(X): Hyperbolic secant of X.
- SIGN(X1, X2): Absolute value of X1 with the sign of X2 (transfer of sign).
- SIN(X): Sine of X in radians.
- SINC(X): Sinc function of X [= SIN(X)/X].
- SIND(X): Sine of X in degrees.
- SINH(X): Hyperbolic sine of X.
- SQR(X): Square of X (= X∗X).
- SQRT(X): Square root of X.
- TAN(X): Tangent of X in radians.
- TAND(X): Tangent of X in degrees.
- TANH(X): Hyperbolic tangent of X.

G.13 Symbolic Constants

The following symbolic constants are available (the enclosing "<>" brackets must be included):

- <bad>: The 'bad' value used to flag missing data. Note that you cannot usefully compare values with this constant because the result is always <bad>. The ISBAD() function should be used instead.
- <dig>: Number of decimal digits of precision available in a floating-point (double-precision) value.
- <e>: Base of natural logarithms.
- <epsilon>: Smallest positive number such that 1.0+<epsilon> is distinguishable from unity.
- <mant_dig>: The number of base <radix> digits stored in the mantissa of a floating-point (double-precision) value.
- <max>: Maximum representable floating-point (double-precision) value.
- <max_10_exp>: Maximum integer such that 10 raised to that power can be represented as a floating-point (double-precision) value.

- <max_exp>: Maximum integer such that <radix> raised to that power minus 1 can be represented as a floating-point (double-precision) value.
- <min>: Smallest positive number which can be represented as a normalised floating-point (double-precision) value.
- <min_10_exp>: Minimum negative integer such that 10 raised to that power can be represented as a normalised floating-point (double-precision) value.
- <min_exp>: Minimum negative integer such that <radix> raised to that power minus 1 can be represented as a normalised floating-point (double-precision) value.
- <pi>: Ratio of the circumference of a circle to its diameter.
- <radix>: The radix (number base) used to represent the mantissa of floating-point (double-precision) values.
- <rounds>: The mode used for rounding floating-point results after addition. Possible values include: -1 (indeterminate), 0 (toward zero), 1 (to nearest), 2 (toward plus infinity) and 3 (toward minus infinity). Other values indicate machine-dependent behaviour.

G.14 Evaluation Precedence and Associativity

Items appearing in expressions are evaluated in the following order (highest precedence first):

- Constants and variables
- Function arguments and parenthesised expressions
- Function invocations
- Unary + - ! .not.
- **
- * /
- + -
- << >>
- < .lt. <= .le. > .gt. >= .ge.
- == .eq. != .ne.
- &
- ^
- |
- && .and.
- ^^

- || .or
- .eqv. .neqv. .xor.

All operators associate from left-to-right, except for unary +, unary -, !, .not. And ** which associate from right-to-left.

H Standard Components in an NDF

An NDF comprises a main data array plus a collection of objects drawn from a set of standard items and extensions (see SUN/33). Only the main data array must be present; all the other items are optional.

`example.sdf` is an NDF which contains all the standard NDF components, except a WCS component and a FITS extension; it also has a FIGARO extension. The structure of the file (as revealed by HDSTRACE) is shown below. The layout is

```
NAME(dimensions)    <TYPE>     VALUE(S)
```

and each level down the hierarchy is indented. Note that scalar objects have no dimensions

```
EXAMPLE  <NDF>

   DATA_ARRAY     <ARRAY>          {structure}
      DATA(856)      <_REAL>          *,0.2284551,-2.040089,45.84504,56.47374,
                                      ... 746.2602,820.8976,570.0729,*,449.574
      ORIGIN(1)      <_INTEGER>       4
      BAD_PIXEL      <_LOGICAL>       FALSE

   TITLE          <_CHAR*30>       'HR6259 - AAT fibre data'
   LABEL          <_CHAR*20>       'Flux'
   UNITS          <_CHAR*20>       'Counts/s'
   QUALITY        <QUALITY>        {structure}
      BADBITS        <_UBYTE>         1
      QUALITY(856)   <_UBYTE>         1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
                                      ... 0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0

   VARIANCE       <ARRAY>          {structure}
      DATA(856)      <_REAL>          2.1,0.1713413,1.5301,34.38378,42.35531,
                                      ... 615.6732,427.5547,353.9127,337.1805
      ORIGIN(1)      <_INTEGER>       4
      BAD_PIXEL      <_LOGICAL>       FALSE

   AXIS(1)        <AXIS>           {structure}

   Contents of AXIS(1)
      DATA_ARRAY(856)  <_REAL>        3847.142,3847.672,3848.201,3848.731,
                                      ... 4298.309,4298.838,4299.368,4299.897
      LABEL          <_CHAR*20>       'Wavelength'
      UNITS          <_CHAR*20>       'Angstroms'

   HISTORY        <HISTORY>        {structure}
      CREATED        <_CHAR*30>       '1990-DEC-12 08:21:02.324'
      CURRENT_RECORD  <_INTEGER>      3
      RECORDS(10)    <HIST_REC>       {array of structures}

      Contents of RECORDS(1)
```

```
        TEXT          <_CHAR*40>      'Extracted spectrum from fibre data.'
        DATE          <_CHAR*25>      '1990-DEC-19 08:43:03.08'
        COMMAND       <_CHAR*30>      'FIGARO V2.4 FINDSP command'


    MORE          <EXT>           {structure}
       FIGARO        <EXT>           {structure}
        TIME          <_REAL>         1275
        SECZ          <_REAL>         2.13

End of Trace.
```

Of course, this is only an example format. There are various ways of representing some of the components. These *variants* are described in SGP/38, but not all are currently supported.

The components are considered in detail below. The names (in bold typeface) are significant as they are used by the NDF access routines to identify the components.

DATA — the main data array (called DATA_ARRAY for historical reasons) is the only component which must be present in an NDF. In the case of `example.sdf`, is a one-dimensional array of real type with 856 elements. It can have up to seven dimensions. It is particularly referenced via parameter names IN, OUT, and NDF.

If neither origin nor bad-pixel flag were present, the DATA component could have been a one-dimensional array like this,

```
DATA_ARRAY(856)  <_REAL>       *,0.2284551,-2.040089,45.84504,56.47374,
                               ... 746.2602,820.8976,570.0729,*,449.574
```

rather than the structure shown above.

ORIGIN — an array of the indices of the first pixel along each axis, defaulting to 1. See Section 12.1 for further information and a graphic. ORIGIN may be set with task SETORIGIN, or after processing an NDF section.

BAD_PIXEL — the bad-pixel flag indicating whether there may be bad pixels present. See SETBADC.1 for further information. The flag may be set with task SETBAD.

TITLE — the character string `"HR6259 - AAT fibre data"` describes the contents of the NDF. The NDF's TITLE might be used as the title of a graph *etc.* It may be set with task SETTITLE. Applications that create an NDF assign a TITLE to the NDF via a parameter, called TITLE unless the application generates several NDFs.

LABEL — the character string `"Flux"` describes the quantity represented in the NDF's main data array. The LABEL is intended for use on the axis of graphs *etc.* It may be set using the task SETLABEL.

UNITS — this character string describes the physical units of the quantity stored in the main data array, in this case, `"Counts/s"`. It may be set via the command SETUNITS.

QUALITY — this component is used to indicate the quality of each element in the main data array, for example whether each pixel is vignetted or not. The quality structure contains a quality array and a BADBITS value, both of which *must* be of type _UBYTE. The quality

array has the same shape and size as the main data array and is used in conjunction with the BADBITS value to decide the quality of a pixel in the main data array. In `example.sdf` the BADBITS component has value 1. This means that a value of 1 in the quality array indicates a bad pixel in the main data array, whereas any other value indicates that the associated pixel is good. (Note that the pixel is bad if the bit-wise comparison QUALITY "AND" BADBITS is non-zero). The meanings of the QUALITY bits are arbitrary. See the task SETBB. To enter new quality information, use the SETQUAL command.

VARIANCE — the variance array is the same shape and size as the main data array and contains the errors associated with the individual data values. These are stored as *variance* estimates for each pixel. VARIANCE may be set with the SETVAR command.

AXIS — the AXIS structure may contain axis information for any dimension of the NDF's main array. In this case, the main data array is only one-dimensional, therefore only the AXIS(1) structure is present. This structure contains the actual axis data array of pixel centres, and also label and units information. KAPPA uses the label and units for axis annotations. Not shown in this example are optional array components for storing pixel widths and the variance of the axis centres. All axes or none must be present. Use SETAXIS to set the values of an AXIS array component; AXLABEL and AXUNITS to set an axis LABEL or UNITS component; and SETNORM to set an axis normalisation flag. For more information see Section 12.2.

WCS — this component provides an alternative, and superior, method for storing co-ordinate information. The AXIS structure described above has the severe limitation that it can only describe co-ordinate systems in which all axes are independent (*i.e.* the value on any axis can be changed without needing to change the values on any other axes). This means for instance that axes cannot be rotated (other than by multiples of 90°), and cannot be used to describe celestial co-ordinates over a large field, or near a pole. The WCS component overcomes these restrictions. It contains descriptions of an arbitrary number of different co-ordinate Frames, together with the mappings required to convert positions between these Frames. All NDFs have four default Frames available; these are known as the GRID, FRACTION, PIXEL, and AXIS Frames. The PIXEL Frame corresponds to pixel co-ordinates. The AXIS Frame corresponds to the co-ordinate Frame implied by the NDF AXIS structures. The GRID Frame is similar to the PIXEL Frame, the main difference being that it has a different origin; the centre of the first ('bottom left') pixel is always $(1.0, 1.0)$ in the GRID Frame regardless of the pixel origin of the NDF. The FRACTION Frame corresponds to normalised PIXEL or GRID co-ordinates, scaled from zero to one. Other Frames can be added to the WCS component in various ways, for instance, by importing them from appropriate FITS headers (FITSIN, FITSDIN), or using an appropriate application to create them from scratch (*e.g.* WCSADD, SETSKY or GAIA). See Section 12 for more information.

HISTORY — this component provides a record of the processing history of the NDF. Only the first of three records is shown for `example.sdf`. This indicates that the spectrum was extracted from fibre data using the FIGARO FINDSP command on 1990 December 19. The history recording level is set by task HISSET. This task also allows you to switch off history recording or delete the history records. HISLIST lists an NDF's history. You can add commentary with HISCOM.

EXTENSIONs — the purpose of extensions is to store non-standard items. These auxiliary data could be information about the original observing setup, such as the airmass during the

observation or the temperature of the detector; they may be calibration data or results produced during processing of the data array, *e.g.* spectral-line fits. The extensions are located within the MORE top-level component. `example.sdf` began life as a FIGARO file. It was converted to an NDF using the command DST2NDF (see SUN/55). It contains values for the airmass and exposure time associated with the observations. These are stored in the FIGARO extension, and the intention is that the FIGARO applications which use these values will know where to find them. Task SETEXT lets the contents of extensions be listed, created, deleted, renamed and assigned new values.

One extension that is used by KAPPA is the FITS extension. This holds the FITS headers as an array of 80-character elements, *i.e.* one FITS card image per array element. You can extract the values of ancillary items from the FITS extension to a non-standard extension via FITSIMP. Use FITSEXP to do the reverse operation. The extension can be listed via the command FITSLIST. FITSEDIT allows you to edit the headers prior to export of the dataset to another format such as FITS or IRAF.

KAPPA uses a PROVENANCE extension to record details of the ancestor NDFs used to generate an NDF. This lets you determine how the NDF before you, came to be. For each ancestor NDF, the provenance information includes its path at creation time, the creation epoch, the software that generated the ancestor, and indices to its parent NDFs (if any exist) within the PROVENANCE extension. There is also a MORE component within PROVENANCE that can be filled with arbitrary additional information. Provenance recording is controlled through the `AUTOPROV` environment variable: set it to `1` to enable recording, and set it to any other value to disable recording. When `AUTOPROV` is undefined, then an output NDF will have provenance only if at least one of the input NDFs has provenance. You can examine provenance with PROVSHOW, and modify it with PROVADD and PROVMOD. Selected provenance may be removed with PROVREM.

I IMAGE data format

The IMAGE format as used by some commands in earlier versions of KAPPA is a simple HDS structure, comprising a floating-point data array, a character title, and the maximum and minimum data values. It is variant of the original Wright-Giddings IMAGE structure. There are others is use that contain more items. An example structure is shown schematically below using the HDSTRACE (SUN/102) notation; see Appendix H.

```
HORSEHEAD  <IMAGE>

   DATA_ARRAY(384,512)  <_REAL>   100.5,102.6,110.1,109.8,105.3,107.6,
                                  ... 123.1,117.3,119,120.5,127.3,108.4
   TITLE          <_CHAR*72>      'KAPPA - Flip'
   DATA_MIN       <_REAL>         28.513
   DATA_MAX       <_REAL>         255.94

End of Trace.
```

The DATA_ARRAY may have up to seven dimensions. IMAGE structures are associated with parameters like INPIC and OUTPIC. The TITLE object of new IMAGE structures takes the value of the Parameter OTITLE. DATA_MIN and DATA_MAX are now ignored.

The IMAGE format is not too dissimilar from a *primitive* NDF with no extensions. Indeed if it did not have DATA_MAX and DATA_MIN it would be a *bona fide* NDF. Thus applications that handle the IMAGE format can follow the rules of SGP/38 and process it like an NDF. In effect this means that all extensions are propagated to output files, and a quality array is propagated where the processing does not invalidate its values. IMAGE applications also handle most *simple* NDFs correctly (those where the data array is an array of numbers at the second level of the hierarchical structure). This similarity in formats enables NDF and IMAGE applications to work in co-operation, and so the conversion within KAPPA was undertaken piecemeal over several years. Note that the primitive variant is no longer the norm for NDFs, since for example, it does not support origin information.

J Supported HDS Data Types

KAPPA applications can process NDFs in one or more of the following HDS data types. The correspondence between Fortran types and HDS data types is as follows:

HDS Type	**Number of bytes**	**FORTRAN Type**
_DOUBLE	8	DOUBLE PRECISION
_INTEGER	4	INTEGER
_INT64	8	INTEGER*8 (non-standard)
_REAL	4	REAL
_UBYTE	1	BYTE
_BYTE	1	BYTE
_UWORD	2	INTEGER*2
_WORD	2	INTEGER*2

(_UBYTE) and (_UWORD) types are unsigned and so permit data ranges of 0–255 and 0–65535 respectively.

K Release Notes—V2.6

K.1 New Commands

The following new applications have been added:

COMPLEX Converts between representations of complex data, such as extracting or combining real and imaginary parts.

MOCGEN Creates a description of selected regions in an image using the IVOA's Multi-Order Coverage (MOC) scheme.

PIXBIN Places each input pixel into a bin defined by one or more index NDFs and creates an output NDF holding all the values in each bin.

K.2 General Changes

The following applications now support huge NDFs: ADD, ADD, CDIV, CMULT, CSUB, COLLAPSE, DIV, MFITTREND, MSTATS, MULT NDFCOPY, NDFTRACE, NUMB PERMAXES, STATS, and SUB. Huge means more than 2,147,483,647 elements in an array component.

K.3 Modified Commands

The following applications have been modified:

ARDMASK

- It can now process complex data.

CHPIX

- It is easier now to make several edits by supplying the NDF sections and fill values in a text file. This is made possible new parameters called MODE and FILE. The default for MODE preserves the historic interactive behaviour.

DISPLAY

- Parameter KEYPOS(2) accepts negative values to instruct DISPLAY to plot the key aligned with the image. Thus setting KEY=`TRUE` and KEYPOS=`[0,-1]` will draw the colour-table ramp so that it abuts the image with its vertical extent matching the image, as commonly needed for journal graphics.

LINPLOT

- The XMAP parameter now has a new option called `"LRLinear"`, which causes the *X* axis to be annotated linearly increasing from left to right. This differs from the existing `"Linear"` option, which may sometimes produce annotated values that increase from right to left, depending on the nature of the WCS.

LISTSHOW

- New parameters called NDF and COMP allow the data, error, or variance values at the catalogued positions within a chosen NDF to be displayed. These use the interpolation method specified by the new Parameters METHOD and PARAMS. The pixel values are also written to a new output parameter called PIXVALS.

LISTSHOW

- New parameters called NDF and COMP allow the data, error, or variance values at the catalogued positions within a chosen NDF to be displayed. These use the interpolation method specified by the new Parameters METHOD and PARAMS. The pixel values are also written to a new output parameter called PIXVALS.

MANIC

- It is now possible to collapse axes using a median instead of the default mean, controlled by a new parameter called ESTIMATOR.

REMQUAL

- A new parameter called CLEAR allows the corresponding bit within the NDF's QUALITY array to be cleared.

SCATTER

- A new parameter called FIT allows a linear fit to the scatter plot to be calculated and displayed.
- The correlation coefficient is now calculated from the visible points alone. Any points outside the bounds of the plot are ignored. Previously, such points were not ignored.
- The reported number of points plotted now ignores any points outside the bounds of the plot.

SUBSTITUTE

- A new parameter called LUT allows multiple values to be changed simultaneously by supplying a table of old and new values in the form a text file. Interpolation between the values can be performed if required.
- A new parameter called TYPE allows the data type of the output NDF to be specified explicitly.

L Notes from Previous Few Releases

L.1 Release Notes—V2.0

L.1.1 General Changes

- Now supports 64-bit integer data.

L.1.2 New Commands

The following new applications have been added:

CONFIGECHO This is intended as a scripting tool. It displays the value of a named entry in a group of configuration parameters.

NDFECHO This is intended as a scripting tool. It expands a given group expression into a list of explicit NDF names, and displays a specified subset of the expanded names.

L.1.3 Modified Commands

The following applications have been modified:

CHANMAP

- Four new estimators are available: FBAD, FGOOD, NBAD and NGOOD, which produce the fraction/count of good/bad pixel values.

COLLAPSE

- Four new estimators are available: FBAD, FGOOD, NBAD and NGOOD, which produce the fraction/count of good/bad pixel values.

MSTATS

- Four new estimators are available: FBAD, FGOOD, NBAD and NGOOD, which produce the fraction/count of good/bad pixel values.

NORMALIZE

- A new boolean parameter called LOOP permits normalisation against a single row or column when comparing two-dimensional NDFs.

PARGET

- A new boolean parameter called VECTOR specifies the output format to use for vector-valued parameters.

ROTATE

- Now estimates north at the centre of the image rather than at the bottom left corner, and uses a more accurate method.

WCSADD

- The transfer of set attribute values from basis Frame to new Frame can now be controlled using a new boolean parameter called TRANSFER (previously, set attributes were always transferred). The new default is to transfer attributes only if the two Frames have the same class and Domain.

WCSREMOVE

- The Frames to remove can now be specified by name as well as by index.

L.2 Release Notes—V2.1

L.2.1 New Commands

The following new applications have been added:

EXCLUDEBAD This will copy a two-dimensional NDF, excluding any rows or columns that contain too many bad values. Good rows or columns are shuffled down to lower indices to fill the gaps left by the excluded rows or columns, thus causing the output NDF to be smaller than the input NDF.

L.2.2 Modified Commands

The following applications have been modified:

ARDPLOT

- Can now display the outline of a Region even if no picture has been displayed previously on the graphics device. The size of the plot is controlled by the new SIZE parameter. Any existing picture can be ignored by setting the new CLEAR parameter to TRUE.

BEAMFIT

- There is now more control of the initial or fixed sizes and shapes of the beams. Note that **this has involved a change of the type and function of Parameter FIXFWHM**. FIXFWHM like other FIX- parameters is _LOGICAL; it just constrains whether the FWHM values should be fixed. A new parameter called FWHM allows you to set either initial values, or when FIXFWHM is also set TRUE, it sets fixed FWHM values. The interpretation of FWHM values depends on a new CIRCULAR parameter, which constrains the fit to be circular thus there is no minor axis and orientation derived. In combination it is possible to give a list of circular or elliptical FWHMs.
- The output parameters now store the statistics of every fitted beam, not just those of the primary beam.

CENTROID

- The centroid's formatted co-ordinates, such as right ascension and declination, are now normalised into the usual ranges. This aplies both to the reported positions and the output parameters.

COPYBAD

- Now writes the number of good and bad pixels in the output NDF to output parameters NGOOD and NBAD.
- No longer sets the BAD_PIXEL flag for the DATA and VARIANCE components.

DISPLAY

- The MODE parameter can now be set to `"Current"` to force the current upper and lower limits to be re-used.

ERASE

- Now has a parameter called REPORT that indicates if an error should be reported if the specified object does not exist.

GDCLEAR

- Will now remove any unused space from the graphics-database file, thus keeping its size to a minimum.

HISTOGRAM

- The new WIDTH parameter offers the option to specify the bin width instead of the number of bins.

MFITTREND

- Now has a parameter called PROPBAD, which controls whether to propagate bad input values to the returned fit.

NDFECHO

- A new parameter called EXISTS has been added that allows the list of displayed NDF paths to be filtered by removing the paths for NDFs that do not exist.

NORMALIZE

- This will loop if the first NDF is one-dimensional and the second is two-dimensional, provided LOOP=`TRUE`. It previously only worked if the dimensionalities were in the reverse sense.

OUTSET

- The USEAXIS parameter now works, needed when the supplied NDF has more than two axes.

PROVADD

- The inoperative parameter MORE has been removed.

SCATTER

- Now writes the number of pixels used to form the correlation coefficient to output parameter NPIX.

SETQUAL

- It is now possible to copy all quality information from one NDF to another using a new parameter called LIKE.

WCSALIGN

- The Gaussian kernel may now be applied in resampling mode as well as rebinning mode.

L.3 Release Notes—V2.2

L.3.1 Documentation Changes

- SUN/95 has been upgraded to the new style of documentation. Some residual collateral damage to the typesetting is likely to be present.
- Most of the old release notes have been removed from SUN/95, with just the few most-recent sets of notes retained in a separate appendix.
- The detailed descriptions of plotting and AST attributes are now in appendices.

L.3.2 Modified Commands

The following applications have been modified:

COLLAPSE

- Fixed bug in the calculation of the variance for the Sum estimator. Note that this applies to other collapsing commands such as MSTATS.

CONFIGECHO

- A new parameter called LOGFILE has been added that allows the list of displayed configuration parameters to be written to a text file.

COPYBAD

- Restore setting the BAD_PIXEL flag for the DATA and VARIANCE components, only setting it false if no bad pixels were copied and none existed in the input NDF.

SEGMENT

- A bug that caused a crash for NDFs with degenerate axes has been fixed.

SETQUAL

- A new parameter QVALUE can be used to store a constant integer value in the range 0 to 255 in the QUALITY component for all pixels.

WCSALIGN

- A new parameter ALIGNREF can be used to control the co-ordinate system in which the input NDFs are aligned.

WCSMOSAIC

- A new parameter ALIGNREF can be used to control the co-ordinate system in which the input NDFs are aligned.

L.4 Release Notes—V2.3

L.4.1 New Commands

The following new applications have been added:

NDFCOMPARE Compares two NDFs and reports whether they are equivalent, based on a range of different tests.

L.4.2 Modified Commands

The following applications have been modified:

BEAMFIT

- Now works for HEALPix maps with its apparently non-square pixels.
- A long-standing issue of occasional nonsense WCS errors has been rectified by using a better-conditioned algorithm.

FITSMOD

- A missing END header may be appended using the Write mode. Any associated value and/or comment are ignored. The easiest way to append an END header is with the wrapper FITSWRITE.

NORMALIZE

- Now calculates and displays Pearson's coefficient of linear correlation on the remaining data at every iteration.
- New Parameter CORR added to hold the last displayed correlation coefficient.
- New Parameters OUTSLOPE, OUTOFFSET and OUTCORR added. These are one-dimensional NDFs in which the slopes, offsets and correlation coefficients respectively are stored when operating in looping mode (*i.e.* LOOP=TRUE).

ROTATE

- Now writes out the rotation angle actually used to an output parameter (ANGLEUSED).

SQORST

- Propagates UNITS as it used to in the IMAGE-format version.

L.5 Release Notes—V2.4

L.5.1 New Commands

The following new application have been added:

ALIGN2D Aligns a pair of two-dimensional NDFs by minimising the residuals between them.

L.5.2 Modified Commands

The following applications have been modified:

APERADD

- Has a new parameter MASK, which can be used to save an NDF containing a mask showing which pixels were included in the aperture.

COLLAPSE

- A warning that suggested that WLIM should b lowered even when it had the minmum of zero no longer appears.

LINPLOT

- Parameter TEMPSTYLE is withdrawn. The + syntax should be used to set temporary style changes.

LUCY

- A bug that prevented correct background removal when Parameter BACK was null was excised.

MFITTREND

- Has a new FOREST parameter, which improves spectral-line masking in line forests using a smoothed mode rather than the mean and a better estimate of the baseline noise.

- A bug has been fixed preventing fits in the rare combination of neither variance nor bad values being present, and without masking of lines. Bad variances are also now checked before spline fitting.

NDFCOPY

- A bug has been fixed that prevented excess WCS axes from being removed.

NORMALIZE

- Has two new parameters DRAWMARK and DRAWWIDTH that can be used to exclude central markers and width indicator from the plot.

PROVSHOW

- Has a new option SHOW="TREE", which allows the family tree to be stepped through in an interactive manner, with the user choosing which parent is to be displayed next.

WCSADD

- Has a new option MAPTYPE="REFNDF", which causes a copy of a co-ordinate Frame read from a reference NDF to be added into the modified NDF.
- New Parameter RETAIN allows control over whether or not the new Frame becomes the current Frame in the modified NDF on exit.

L.6 Release Notes—V2.5

L.6.1 General Changes

- A log of KAPPA commands can now be written to a text file specified by the environment variable KAPPA_LOG. The log lists the application name and parameter values in separate headed lines. Note that the format of the log may change to simple command lines that could be replayed in a script.

L.6.2 Modified Commands

The following applications have been modified:

ALIGN2D

- Parameter TR may also include the scale and offset in its seventh and eight elements.
- The RMS residual between the aligned and the reference arrays is now written to an output parameter called “RMS”.

CONFIGECHO

- This now reports all elements in an array, not just the first element.

CONTOUR

- The dynamic default for Parameter LABPOS is now `!` (*i.e.* a null value), so no label is now drawn in `"Bounds"` or "Good" mode unless a value is supplied explicitly for LABPOS.

DISPLAY

- Has a new Parameter PENRANGE, which can be used to restrict the range of pens (i.e. colour indices) used. The default is to use the full range of available pens.
- The vertical position of the key can now be controlled through Parameter KEYPOS.

HISCOM

- Has a new Parameter APPNAME, which can be used to change the application name stored in the new history record from the default of "HISCOM". Scripts that generate NDFs can use this facility to record the details of the invocation of the script in the form of a history record in the output NDF.

MFITTREND

- The auto method uses the median rather than the mean to clip outliers. This permits better masking of strong and extended emission.

SQORTST

- Permit an axis scale to be retained by using an asterisk in Parameter PIXSCALE.

TRANDAT

- Will now recognise the string `"BAD"` (case insensitive) within the input text file and generate appropriate bad values in the output NDF.

M Release Notes—V2.5-9

M.1 Modified Commands

The following applications have been modified:

ALIGN2D

- The residuals used to determine the fit are now weighted using the SNR of the data rather than the reciprocal of the variance. This will cause the final alignment to be determined more by the brighter sections of the map than previously.

COLLAPSE

- There is a new estimator option, `"FastMed"`, that offers a substantially faster calculation of unweighted medians.

SETAXIS

- There is a new mode, `"NDF"`. It assigns the axis values using the data values in another NDF, specified by a parameter called AXISNDF.

SQORST

- Has a new Parameter CENTRE, which specifies the centre about which the WCS co-ordinates are stretched or squashed.

WCSALIGN

- The input NDFs can now be aligned with a specified POLPACK catalogue—see new Parameter REFCAT.

WCSMOSAIC

- Has a new parameter called WEIGHTS that can be used to specify a weight for each of the input NDFs.