SUN/55.32

Starlink Project
Starlink User Note 55.32

Malcolm J. Currie
G.J.Privett
A.J.Chipperfield
D.S.Berry
A.C.Davenhall

2015 December 14

CONVERT – A Format-conversion Package
Version 1.8
User's Manual

Abstract

The CONVERT package contains utilities for converting data files between Starlink's Extensible *n*-dimensional Data Format (NDF), which is used by most Starlink applications, and a number of other common data formats. Using these utilities, astronomers can process their data selecting the best applications from a variety of Starlink or other packages.

Most of the CONVERT utilities may be run from the shell or ICL in the normal way, or invoked automatically by the NDF library's 'on-the-fly' data-conversion system, but there are also IDL procedures for handling NDFs from within IDL.

Contents

List of Figures

1 Introduction

If life were simple there would only be one data format, but in reality there are numerous formats for storing *n*-dimensional astronomical data associated with various software packages. In Starlink we have not been immune to this, having the original INTERIM BDF, HDS IMAGE format, and FIGARO DSTs to name but three. However, Starlink is now taking the novel approach of supporting different packages sharing a common data format—the NDF[1] (Extensible *n*-Dimensional-data format)—which most Starlink packages are already using.

The purpose of CONVERT is the interchange of data files to and from the NDF. Thus it enables astronomers to select the best applications from a variety of packages, including those originating abroad like IRAF. In addition it assists packages that wish to move to using the NDF.

CONVERT is available from all three UNIX platforms supported by Starlink. The supported conversions are currently as follows:

[1] See SUN/33 for an introduction to the NDF.

ASCII2NDF – Converts a text file to an NDF.

AST2NDF – Converts an Asterix data cube into a standard NDF.

DA2NDF – Converts a direct-access unformatted file to an NDF.

DST2NDF – Converts a Figaro (Version 2) DST file to an NDF.

FITS2NDF – Converts FITS files into NDFs.

GASP2NDF – Converts an image in GASP format to an NDF.

GIF2NDF – Converts an image in GIF format to an NDF.

IRAF2NDF – Converts an IRAF image to an NDF.

IRCAM2NDF – Converts an IRCAM data file to a series of NDFs.

MTFITS2NDF – Converts a FITS tape to one or more NDFs.

NDF2ASCII – Converts an NDF to a text file.

NDF2DA – Converts an NDF to a direct-access unformatted file.

NDF2DST – Converts an NDF to a Figaro (Version 2) DST file.

NDF2FITS – Converts an NDF to a FITS file.

NDF2GASP – Converts a two-dimensional NDF into a GASP image.

NDF2GIF – Converts an NDF to a GIF file.

NDF2IRAF – Converts an NDF to an IRAF image.

NDF2PGM – Converts an NDF to a PGM format.

NDF2TIFF – Converts an NDF to a TIFF file.

NDF2UNF – Converts an NDF to a sequential unformatted file.

SPECX2NDF – Converts a SPECX map into a simple data cube, or SPECX data files to individual spectra.

TIFF2NDF – Converts an image in TIFF format to an NDF.

UNF2NDF – Converts a sequential unformatted file to an NDF.

In addition there are FITS readers within KAPPA (see SUN/95) which will convert FITS files and tapes to NDFs. These are more tolerant of 'almost FITS' files, but lack support for IMAGE and BINTABLE extensions.

IDL procedures for handling NDFs and other methods of converting NDFs to IDL format are described in Appendix B of this document.

Starting up the CONVERT package will also set up defaults for the automatic NDF conversion facilities (described in Section 3) to enable applications which use the NDF library to read and write most of the file formats handled by the CONVERT package, and some others.

The various formats supported by CONVERT do not have one-to-one correspondence and therefore in general it is not possible to apply a forward and reverse conversion and finish with a duplicate of the initial data file. This hysteresis is particularly likely when starting with an NDF, since many simpler formats have no way of storing certain NDF data items, like variance and axis widths. However, if you are dealing with a simple file containing just a data array and linear axis centres, then it should be possible to avoid loss of information except with GIF and TIFF formats which will reduce the absolute data values to 256 greyscale levels.

Note – the input data file is not deleted or altered in any way.

2 Running CONVERT

2.1 Starting CONVERT from the UNIX shell

The command `convert` defines CONVERT commands from the UNIX shell.

```
% convert
```

Note that the % is the UNIX shell's prompt which you do not type.

A message similar to:

```
CONVERT commands are now available -- (Version 1.0, 1997 August)

Defaults for automatic NDF conversion are set.

Type conhelp for help on CONVERT commands.
Type "showme sun55" to browse the hypertext documentation.
```

will be displayed. You will then be able to mix CONVERT and UNIX commands.

The `convert` command is defined by the Starlink startup procedures to 'source' file `convert.csh` in the CONVERT executables directory. Non-Starlink sites must make their own arrangements.

2.2 Starting CONVERT from ICL

To start ICL, type:

```
% icl
```

You will see any messages produced by system and user procedures, followed by the `ICL>` prompt, something like the following.

```
ICL (UNIX) Version 3.1-5 20/05/97

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>
```

Then, to make the CONVERT commands known to the command language, type:

```
ICL> CONVERT
```

This will produce a CONVERT startup message similar to:

```
CONVERT commands are now available -- (Version 1.0, 1997 August)

Defaults for automatic NDF conversion are set.

Type CONHELP or HELP CONVERT for help on CONVERT commands.
Type "showme sun55" to browse the hypertext documentation.
```

The ICL command `CONVERT` is defined by the standard Starlink ICL login files to `LOAD` file `convert.icl` in the CONVERT executables directory. Non-Starlink sites must make their own arrangements.

2.3 Issuing CONVERT Commands

Having initialised CONVERT you are now ready to issue a CONVERT command. To run an application you can just give its name (or its name preceded by `con_`[2])—you will be prompted for any required *parameters*. Alternatively, you may enter parameter values on the command line specified by position or by keyword. If you want to override any defaulted parameters, then you specify the parameter's value on the command line. Note that *from UNIX the commands are in lowercase*, whereas from ICL the case does not matter.

Most CONVERT applications can be run as simply as:

```
<application> <in> <out>
```

where <application> is the application's name, <in> is the input file, and <out> is the output file following the conversion. For instance, from the UNIX shell,

```
% dst2ndf old new
```

or, from ICL,

```
ICL> DST2NDF old new
```

both instruct the application DST2NDF to convert the DST file called `old.dst` to the NDF called `new.sdf`. Note that for UNIX, the case of the filename is significant.

The following example has the same effect as those immediately above, only this time you are prompted for the filenames needed by DST2NDF.

```
ICL> DST2NDF
IN - Name of Figaro (.DST) file to be converted /' '/ > old
OUT - Name of output NDF /@f1/ > new
```

[2]The con_<name> form is defined for use where there may be confusion between commands of the same name from different packages.

The value between the / / delimiters is a suggested default. You can choose to accept the suggestion by pressing carriage return.

The simple usage, (<application> <in> <out>), will usually produce a result but many applications have additional parameters which you can set to give finer control over the conversion. See the application specifications in Appendix A for details of the options available.

You can find details of how to use parameters for controlling Starlink program options in Section 4 of SUN/95 or in Chapter 8 of SG/4. However, you should be able to get along using intuition alone, or, perhaps by consulting the application specifications in Appendix A, which include usage, parameters, examples and details of the conversion process.

In most cases, one invocation of a CONVERT application is required for each file conversion but in some cases, inputs may be defined as 'GROUPS' of names, including wildcards (see the application specifications for details).

2.4 Obtaining Help

You can get the top-level help information for CONVERT by typing:

```
% conhelp
```

from the UNIX shell, or:

```
ICL> CONHELP
```

from ICL. (You can also access CONVERT help from ICL by using the ICL command, HELP.)

The help topics are mostly detailed descriptions of the applications but also include global information on matters such as using parameters. *e.g.* the following command gives help on the application UNF2NDF.

```
% conhelp unf2ndf
```

If you have commenced running an application you can still access the help library whenever you are prompted for a parameter; you enter ?. Here is an example.

```
NOPEREC - Number of data values per output record /512/ > ?

NDF2UNF

  Parameters

    NOPEREC = _INTEGER (Read)
       The number of data values per record of the output file.  It
       should be in the range 1 to 8191, unless the array is double
       precision, when the upper limit is 4095.  The suggested
       default is the current value. [The first dimension of the NDF]

NOPEREC - Number of data values per output record /512/ >
```

2.5 Hypertext Help

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

```
% showme sun55
```

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.

3 Automatic Format Conversion with the NDF Library

SSN/20 describes a system incorporated into the NDF library routines which enables applications written to read or write NDFs to handle any arbitrary 'foreign' format for which a conversion utility can be defined. The system operates via environment variables which define the set of permitted conversions and the commands required to do them.

3.1 The Default Conversion Commands

CONVERT startup will define defaults for the NDF-conversion environment variables which permit automatic conversion of files in the formats handled by CONVERT (except for AST, IRCAM, PGM. SPECX and FITS tapes). It also allows data compression.

The list of format names and associated filename extensions defined by CONVERT is set out in Table 1—the filename extensions tell the system which format the file is in. For the unformatted and ASCII conversions the format names and extensions are somewhat arbitrary. This list will nullify any existing list so private conversions must be added *after* CONVERT startup.

For the unformatted and ASCII conversions the format names and extensions are somewhat arbitrary. The FITS and STREAM formats have synonym file extensions for the conversion to NDF. The standard file extension is required for the conversion to the foreign format.

Table 2 lists the utilities used to perform the conversions. In general the default parameter values are used—non-default parameters (other than the input and output filenames) are listed in the table.

Table 2 also contains a column headed 'Variable'. Most of the command lines issued to do the automatic conversion will include the translation of an environment variable named NDF_FROM_*fmt*_PARS or NDF_TO_*fmt*_PARS as appropriate (where *fmt* is the format name). This may be used to give additional parameters to the command if you do not want to define a completely new command for yourself.

Where the 'Variable' column contains a tick, the variable *must* be used to supply the SHAPE parameter; where it contains a cross, additional parameters cannot be specified.

For example, suppose application `rdndf` uses the NDF library to read one NDF (named by the first parameter) and write another (named by the second parameter). This application could be made to read a TEXT file (`data.txt`) containing the required values for a 50×10 data array, and write its results as a FITS file (`output.fit`) as follows:

```
% convert
CONVERT commands are now available -- (Version 1.4, 2001 November)

Defaults for automatic NDF conversion are set.

Type conhelp for help on CONVERT commands.
Type "showme sun55" to browse the hypertext documentation.

% setenv NDF_FROM_TEXT_PARS 'SHAPE=[50,10]'
% rdndf data.txt output.fit
```

The order of the formats in the tables also defines a search path. If you omit the file extension, the system will search for an NDF of that name. If that is absent, it will try a `.fit` FITS file. If neither are present it tries an IRAF file, and so on. The recognised formats and their order is defined through the environment variable NDF_FORMATS_IN. The shell `convert` startup defines NDF_FORMATS_IN as given below.

```
'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),
```

Format	Extension	Extension Synonyms	Description
FITS	.fit	.fits .fts .FITS .FIT .lilo .lihi .silo .sihi .mxlo .mxhi .rilo .rihi .vdlo .vdhi	FITS
FIGARO	.dst		Figaro (Version 2) DST
IRAF	.imh		IRAF
STREAM	.das	.str	Unformatted direct-access or stream
UNFORMATTED	.unf		Unformatted with FITS header
UNF0	.dat		Unformatted without FITS header
ASCII	.asc		ASCII with FITS header
TEXT	.txt		ASCII without FITS header
GIF	.gif		Graphics Interchange Format
TIFF	.tif		Tag Image File Format
GASP	.hdr		GASP
COMPRESSED	.sdf.Z		Compressed NDF
GZIP	.sdf.gz		**gzip** compressed NDF

Table 1: Defined Formats and Extensions

```
FITS(.FITS),FITS(.FIT),FITS(.lilo),FITS(.lihi),
FITS(.silo),FITS(.sihi),FITS(.mxlo),FITS(.mxhi),
FITS(.rilo),FITS(.rihi),FITS(.vdlo),FITS(.vdhi),STREAM(.str)'
```

but from ICL the CONVERT command does not define the synonyms due to a limitation of ICL. Thus NDF_FORMATS_IN is defined to be the following.

```
'FITS(.fit),FIGARO(.dst),IRAF(.imh),STREAM(.das),UNFORMATTED(.unf),
 UNFO(.dat),ASCII(.asc),TEXT(.txt),GIF(.gif),TIFF(.tif),GASP(.hdr),
 COMPRESSED(.sdf.Z),GZIP(.sdf.gz)'
```

When creating an output file, there is a similar list of recognised formats. The CONVERT startup procedures define NDF_FORMATS_OUT as follows.

```
'.,FITS(.fit),FIGARO(.dst),IRAF(.imh),STREAM(.das),UNFORMATTED(.unf),
UNFO(.dat),ASCII(.asc),TEXT(.txt),GIF(.gif),TIFF(.tif),GASP(.hdr),
COMPRESSED(.sdf.Z),GZIP(.sdf.gz)'
```

The leading dot indicates that if you omit the file extension, the output file will be an NDF.

There are some examples of the automatic system in action and use of NDF_FORMATS_IN and NDF_FORMATS_OUT in SUN/95, Section 15.1.

4 Automatic HISTORY Creation

The NDF data format offers a means by which processing commands and user comments may be recorded, so that at some future time you can determine how a given NDF was created. History recording is optional.[3] However, now massive storage is cheap we recommend that you routinely use NDF history recording.

History recording may be enabled on an NDF-by-NDF basis using the KAPPA command HISSET as described in the NDF History section of SUN/95. In the case of CONVERT this has a disadvantage—the CONVERT conversion command will be absent from the NDFs created by the *x*2NDF commands. If you wish that all new NDFs created by CONVERT have history recording enabled at their creation, you should set the environment variable NDF_AUTO_HISTORY to a non-zero integer value. In fact, we recommend that you set this in your login script so any NDF created from scratch will have this property.

5 Acknowledgments

Jo Murray wrote the original versions of the applications that convert between DSTs and NDFs. Alan Chipperfield produced the IDL converters. Rhys Morris wrote the original versions of IRAF2NDF, NDF2IRAF, GASP2NDF and NDF2GASP. Grant Privett wrote the TIFF and

[3] This was because at the time it was created history records could potentially be quite bulky for spectra.

contributed to the GIF conversion utilities. Clive Davenhall wrote AST2NDF and SPECX2NDF; the latter was substantially revised by David Berry to work with the AST SpecFrame.

Rodney Warren-Smith devised the format-conversion facilities for the NDF data-access library and Alan Chipperfield implemented the CONVERT components of it.

A Specifications of CONVERT Applications

A.1 Explanatory Notes

The *Parameters* section lists the application's parameters, with the format:

```
name  =  type (access)
   description
```

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.

`[default]` If the brackets contain a value in teletype-fount, this is the explicit default value.

There is also 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. An example should clarify.

```
ndf2ascii in out [comp] [reclen] noperec=?
```

IN, OUT, COMP, and RECLEN are all positional parameters. Only IN, and OUT would be prompted if not given on the command line. The remaining parameter, NOPEREC, depends on the value of another parameter (it is FIXED), and will be prompted for when FIXED is `TRUE`.

The *Examples* section 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.

Examples give command lines as accepted by the tasks themselves. From the UNIX shell, *metacharacters* (notably `[,]` and `"`) *must be escaped or enclosed in single quotes*. For example:

```
ascii2ndf ngc253q.dat ngc253 q shape='[100,60]'
fits2ndf '"abc.fit,def.fits"' 'fgh,ijk"' fmtcnv='"F,T"' noproexts
```

FORMAT	In/out	Utility	Non-default parameters	Variable
FITS	in	FITS2NDF		
	out	NDF2FITS	bitpix=-1 proexts=t	
FIGARO	in	DST2NDF		
	out	NDF2DST		×
IRAF	in	IRAF2NDF		
	out	NDF2IRAF		
STREAM	in	DA2NDF	noperec=!	
	out	NDF2DA		
UNFORMATTED	in	UNF2NDF	fits=t noperec=!	
	out	NDF2UNF	fits=t	
UNF0	in	UNF2NDF	fits=f noperec=!	√
	out	NDF2UNF	fits=f	
ASCII	in	ASCII2NDF	fits=t	
	out	NDF2ASCII	fits=t reclen=80	
TEXT	in	ASCII2NDF	fits=f	√
	out	NDF2ASCII	fits=f reclen=80	
GIF	in	GIF2NDF		×
	out	NDF2GIF		×
TIFF	in	TIFF2NDF		×
	out	NDF2TIFF		×
GASP	in	GASP2NDF		
	out	NDF2GASP	fillbad=0	
COMPRESSED	in	**uncompress**		×
	out	**compress**		×
GZIP	in	**gzip**		×
	out	**gunzip**		×

Table 2: Conversion Commands.

ASCII2NDF
Converts a text file to an NDF

Description:

This application converts a text file to an NDF. Only one of the array components may be created from the input file. Preceding the input data there may be an optional header. This header may be skipped, or may consist of a simple FITS header. In the former case the shape of the NDF has be to be supplied.

Usage:

```
ascii2ndf in out [comp] [skip] shape [type]
```

Parameters:

COMP = LITERAL (Read)

The NDF component to be copied. It may be `"Data"`, `"Quality"` or `"Variance"`. To create a variance or quality array the NDF must already exist. `["Data"]`

FITS = _LOGICAL (Read)

If `TRUE`, the initial records of the formatted file are interpreted as a FITS header (with one card image per record) from which the shape, data type, and axis centres are derived. The last record of the FITS-like header must be terminated by an END keyword; subsequent records in the input file are treated as an array component given by COMP. `[FALSE]`

IN = FILENAME (Read)

Name of the input text Fortran file. The file will normally have variable-length records when there is a header, but always fixed-length records when there is no header. The maximum record length allowed is 512 bytes.

MAXLEN = INTEGER (Read)

The maximum record length in bytes of records within the input text file. Unless the records are longer than 512 bytes, you can use the default value. The suggested value is the current value. `[512]`

OUT = NDF (Read and Write)

Output NDF data structure. When COMP is not `"Data"` the NDF is modified rather than a new NDF created. It becomes the new current NDF.

SHAPE = _INTEGER (Read)

The shape of the NDF to be created. For example, `[40,30,20]` would create 40 columns by 30 lines by 10 bands. It is only accessed when FITS is `FALSE`.

SKIP = INTEGER (Read)

The number of header records to be skipped at the start of the input file before finding the data array or FITS-like header. `[0]`

TYPE = LITERAL (Read)

The data type of the output NDF. It must be one of the following HDS types: `"_BYTE"`, `"_WORD"`, `"_REAL"`, `"_INTEGER"`, `"_INT64"`, `"_DOUBLE"`, `"_UBYTE"`, `"_UWORD"` corresponding to signed byte, signed word, real, integer, 64-bit integer, double precision, unsigned byte, and unsigned word. See SUN/92 for further details. An

unambiguous abbreviation may be given. TYPE is ignored when COMP = "Quality" since the QUALITY component must comprise unsigned bytes (equivalent to TYPE = "_UBYTE") to be a valid NDF. The suggested default is the current value. TYPE is only accessed when FITS is FALSE. ["_REAL"]

Examples:

```
ascii2ndf ngc253.dat ngc253 shape=[100,60]
```

This copies a data array from the text file ngc253.dat to the NDF called ngc253. The input file does not contain a header section. The NDF is two-dimensional: 100 elements in x by 60 in y. Its data array has type _REAL.

```
ascii2ndf ngc253q.dat ngc253 q shape=[100,60]
```

This copies a quality array from the text file ngc253q.dat to an existing NDF called ngc253 (such as created in the first example). The input file does not contain a header section. The NDF is two-dimensional: 100 elements in x by 60 in y. Its data array has type _UBYTE.

```
ascii2ndf ngc253.dat ngc253 fits
```

This copies a data array from the text file ngc253.dat to the NDF called ngc253. The input file contains a FITS-like header section, which is copied to the FITS extension of the NDF. The shape of the NDF is controlled by the mandatory FITS keywords NAXIS, AXIS1, ..., AXISn, and the data type by keywords BITPIX and UNSIGNED.

```
ascii2ndf type="_uword" in=ngc253.dat out=ngc253 maxlen=4000 \
```

This copies a data array from the text file ngc253.dat to the NDF called ngc253. The input file does not contain a header section. The NDF has the current shape and data type is unsigned word. The maximum record length is 4000 bytes.

```
ascii2ndf spectrum ZZ skip=2 shape=200
```

This copies a data array from the text file spectrum to the NDF called ZZ. The input file contains two header records that are ignored. The NDF is one-dimensional comprising 200 elements of type _REAL.

```
ascii2ndf spectrum.lis ZZ skip=1 fits
```

This copies a data array from the text file spectrum.lis to the NDF called ZZ. The input file contains one header record, that is ignored, followed by a FITS-like header section, which is copied to the FITS extension of the NDF. The shape of the NDF is controlled by the mandatory FITS keywords NAXIS, AXIS1, ..., AXISn, and the data type by keywords BITPIX and UNSIGNED.

Notes:

The details of the conversion are as follows:

- the ASCII-file array is written to the NDF array as selected by COMP. When the NDF is being modified, the shape of the new component must match that of the NDF.
- If the input file contains a FITS-like header, and a new NDF is created, *i.e.* COMP = `"Data"`, the header records are placed within the NDF's FITS extension. This enables more than one array (input file) to be used to form an NDF. Note that the data array must be created first to make a valid NDF, and it's the FITS structure associated with that array that is wanted. Indeed the application prevents you from doing otherwise.
- The FITS-like header defines the properties of the NDF as follows:
  - BITPIX defines the data type: 8 gives _BYTE, 16 produces _WORD, 32 makes _INTEGER, 64 creates _INT64, −32 gives _REAL, and −64 generates _DOUBLE. For the first two, if there is an extra header record with the keyword UNSIGNED and logical value T, these types become _UBYTE and _UWORD respectively. UNSIGNED is non-standard, since unsigned integers would not follow in a proper FITS file. However, here it is useful to enable unsigned types to be input into an NDF. UNSIGNED may be created by this application's sister, NDF2ASCII. BITPIX is ignored for QUALITY data; type _UBYTE is used.
  - NAXIS, and NAXIS*n* define the shape of the NDF.
  - The TITLE, LABEL, and BUNIT are copied to the NDF TITLE, LABEL, and UNITS NDF components respectively.
  - The CDELT*n*, CRVAL*n*, CTYPE*n*, and CUNIT*n* keywords make linear axis structures within the NDF. CUNIT*n* define the axis units, and the axis labels are assigned to CTYPE*n*. If some are missing, pixel co-ordinates are used for those axes.
  - BSCALE and BZERO in a FITS extension are ignored.
  - BLANK is not used to indicate which input array values should be assigned to a standard bad value.
  - END indicates the last header record unless it terminates a dummy header, and the actual data is in an extension.
- Other data item such as HISTORY, data ORIGIN, and axis widths are not supported, because the text file has a simple structure to enable a diverse set of input files to be converted to NDFs, and to limitations of the standard FITS header.

Related Applications :

CONVERT: NDF2ASCII; KAPPA: TRANDAT; FIGARO: ASCIN and ASCOUT.

AST2NDF
Converts an Asterix data cube into a simple NDF

Description:

This application converts an Asterix data cube into a standard NDF. See Section "Notes" (below) for details of the conversion.

Usage:

```
ast2ndf in out
```

Parameters:

IN = NDF (Read)

The name of the input Asterix data cube. The file extension (`.sdf`) should not be included since it is appended automatically by the application.

OUT = NDF (Write)

The name of the output NDF containing the data cube written by the application. The file extension (`.sdf`) should not be included since it is appended automatically by the application.

Examples:

```
ast2ndf ast_cube ndf_cube
```

This example generates NDF data cube ndf_cube (in file `ndf_cube.sdf`) from Asterix cube ast_cube (in file `ast_cube.sdf`).

Notes:

This application accepts data in the format used by the Asterix package (see SUN/98). These data are cubes, with two axes comprising a regular grid of positions on the sky and the third corresponding to energy or wavelength. The data are Starlink HDS files which are very similar in format to a standard NDF. The following points apply.

- The Asterix QUALITY array is non-standard. There is no QUALITY component in the output NDF. Instead 'bad' or 'null' values are used to indicate missing or suspect values.
- The VARIANCE component is copied if it is present.
- The non-standard Asterix axis components are replaced with standard ones.
- The order of the axes is rearranged.

References:

D.J. Allan and R.J. Vallance, 1995, in SUN/98: *ASTERIX – X-ray Data Processing System*, Starlink.

Related Applications :

KAPPA: AXCONV.

DA2NDF
Converts a direct-access unformatted file to an NDF

Description:

This application converts a direct-access (fixed-length records) unformatted file to an NDF. It can therefore also process unformatted data files generated by C routines. Only one of the array components may be created from the input file. The shape of the NDF has be to be supplied.

Usage:

```
da2ndf in out [comp] noperec shape [type]
```

Parameters:

COMP = LITERAL (Read)
The NDF component to be copied. It may be "Data", "Quality" or "Variance". To create a variance or quality array the NDF must already exist. ["Data"]

IN = FILENAME (Read)
Name of the input direct-access unformatted file.

NOPEREC = _INTEGER (Read)
The number of data values per record of the input file. It must be positive. The suggested default is the size of the first dimension of the array. A null (!) value for NOPEREC causes the size of first dimension to be used.

OUT = NDF (Read and Write)
Output NDF data structure. When COMP is not "Data" the NDF is modified rather than a new NDF created. It becomes the new current NDF. Unusually for an output NDF, there is a suggested default—the current value—to facilitate the inclusion of variance and quality arrays.

SHAPE = _INTEGER (Read)
The shape of the NDF to be created. For example, [40,30,20] would create 40 columns by 30 lines by 10 bands.

TYPE = LITERAL (Read)
The data type of the direct-access file and the NDF. It must be one of the following HDS types: "_BYTE", "_WORD", "_REAL", "_INTEGER", "_INT64", "_DOUBLE", "_UBYTE", "_UWORD" corresponding to signed byte, signed word, real, 64-bit integer, integer, double precision, unsigned byte, and unsigned word respectively. See SUN/92 for further details. An unambiguous abbreviation may be given. TYPE is ignored when COMP = "Quality" since the QUALITY component must comprise unsigned bytes (equivalent to TYPE = "_UBYTE") to be a valid NDF. The suggested default is the current value. ["_REAL"]

Examples:

```
da2ndf ngc253.dat ngc253 shape=[100,60] noperec=8
```

This copies a data array from the direct-access file ngc253.dat to the NDF called

ngc253. The NDF is two-dimensional: 100 elements in x by 60 in y. Its data array has type _REAL. The data records each have 8 real values.

```
da2ndf ngc253q.dat ngc253 q 100 [100,60]
```

This copies a quality array from the direct-access file `ngc253q.dat` to an existing NDF called ngc253 (such as created in the first example). The NDF is two-dimensional: 100 elements in x by 60 in y. Its data array has type _UBYTE. The data records each have 100 unsigned-byte values.

```
da2ndf type="_uword" in=ngc253.dat out=ngc253 \
```

This copies a data array from the direct-access file `ngc253.dat` to the NDF called ngc253. The NDF has the current shape and data type is unsigned word. The current number of values per record is used.

Notes:

The details of the conversion are as follows:

- the direct-access file's array is written to the NDF array as selected by COMP. When the NDF is being modified, the shape of the new component must match that of the NDF. This enables more than one array (input file) to be used to form an NDF. Note that the data array must be created first to make a valid NDF. Indeed the application prevents you from doing otherwise.
- Other data items such as axes are not supported, because of the direct-access file's simple structure.

Related Applications :

CONVERT: NDF2DA.

DST2NDF
Converts a Figaro (Version 2) DST file to an NDF

Description:

This application converts a FIGARO Version-2 DST file to a Version-3 file, *i.e.* to an NDF. The rules for converting the various components of a DST are listed in the "Notes". Since both are hierarchical formats most files can be be converted with little or no information lost.

Usage:

```
dst2ndf in out
```

Parameters:

FORM = LITERAL (Read)

The storage form of the NDF's data and variance arrays. FORM = "Simple" gives the simple form, where the array of data and variance values is located in an ARRAY structure. Here it can have ancillary data like the origin. This is the normal form for an NDF. FORM = "Primitive" offers compatibility with earlier formats, such as IMAGE. In the primitive form the data and variance arrays are primitive components at the top level of the NDF structure, and hence it cannot have ancillary information. ["Simple"]

IN = Figaro file (Read)

The file name of the version 2 file. A file extension must not be given after the name, since ".dst" is appended by the application. The file name is limited to 80 characters.

OUT = NDF (Write)

The file name of the output NDF file. A file extension must not be given after the name, since ".sdf" is appended by the application. Since the NDF_ library is not used, a section definition may not be given following the name. The file name is limited to 80 characters.

Examples:

```
dst2ndf old new
```

This converts the Figaro file old.dst to the NDF called new (in file new.sdf). The NDF has the simple form.

```
dst2ndf horse horse p
```

This converts the Figaro file horse.dst to the NDF called horse (in file horse.sdf). The NDF has the primitive form.

Notes:

The rules for the conversion of the various components are as follows:

Figaro file		NDF	Comments
.Z.DATA	⇒	.DATA_ARRAY.DATA	when FORM = `"SIMPLE"`
.Z.DATA	⇒	.DATA_ARRAY	when FORM = `"PRIMITIVE"`
.Z.ERRORS	⇒	.VARIANCE.DATA	after processing when FORM = `"SIMPLE"`
.Z.ERRORS	⇒	.VARIANCE	after processing when FORM = `"PRIMITIVE"`
.Z.QUALITY	⇒	.QUALITY.QUALITY	must be BYTE array (see Bad-pixel handling below)
	⇒	.QUALITY.BADBITS = 255	
.Z.LABEL	⇒	.LABEL	
.Z.UNITS	⇒	.UNITS	
.Z.IMAGINARY	⇒	.DATA_ARRAY.IMAGINARY_DATA	
.Z.MAGFLAG	⇒	.MORE.FIGARO.MAGFLAG	
.Z.RANGE	⇒	.MORE.FIGARO.RANGE	
.Z.xxxx	⇒	.MORE.FIGARO.Z.xxxx	
.X.DATA	⇒	.AXIS(1).DATA_ARRAY	
.X.ERRORS	⇒	.AXIS(1).VARIANCE	after processing
.X.WIDTH	⇒	.AXIS(1).WIDTH	
.X.LABEL	⇒	.AXIS(1).LABEL	
.X.UNITS	⇒	.AXIS(1).UNITS	
.X.LOG	⇒	.AXIS(1).MORE.FIGARO.LOG	
.X.xxxx	⇒	.AXIS(1).MORE.FIGARO.xxxx	(Similarly for .Y .T .U .V or .W structures which are renamed to AXIS(2), . . . , AXIS(6) in the NDF.)
.OBS.OBJECT	⇒	.TITLE	
.OBS.SECZ	⇒	.MORE.FIGARO.SECZ	
.OBS.TIME	⇒	.MORE.FIGARO.TIME	
.OBS.xxxx	⇒	.MORE.FIGARO.OBS.xxxx	
.FITS.xxxx	⇒	.MORE.FITS.xxxx	into value part of the string
.COMMENTS.xxxx	⇒	.MORE.FITS(*n*)	into comment part of the string
.FITS.xxxx.DATA	⇒	.MORE.FITS(*n*)	into value part of the string
.FITS.xxxx.DESCRIPTION	⇒	.MORE.FITS(*n*)	into comment part of the string
.FITS.xxxx.yyyy	⇒	.MORE.FITS(*n*)	into blank-keyword comment containing `yyyy=value`
.MORE.xxxx	⇒	.MORE.xxxx	
.TABLE	⇒	.MORE.FIGARO.TABLE	
.xxxx	⇒	.MORE.FIGARO.xxxx	

Axis arrays with dimensionality greater than one are not supported by the NDF. Therefore, if the application encounters such an axis array, it processes the array using the following rules, rather than those given above.

Figaro file		NDF	Comments
.X.DATA	⇒	.AXIS(1).MORE.FIGARO.DATA_ARRAY	AXIS(1).DATA_ARRAY is filled with pixel co-ordinates
.X.ERRORS	⇒	.AXIS(1).MORE.FIGARO.VARIANCE	after processing
.X.WIDTH	⇒	.AXIS(1).MORE.FIGARO.WIDTH	

In addition to creating a blank-keyword NDF FITS-extension header for each component of a non-standard DST FITS structure (.FITS.xxxx.yyyy where yyyy is not DATA or DESCRIPTION), this set of related headers are bracketed by blank lines and a comment containing the name of the structure (*i.e.* xxxx).

Related Applications :

CONVERT: NDF2DST.

Bad-pixel handling :

The QUALITY array is only copied if the bad-pixel flag (.Z.FLAGGED) is FALSE or absent. A simple NDF with the bad-pixel flag set to FALSE (meaning that there are no bad-pixels present) is created when .Z.FLAGGED is absent or false and FORM = `"SIMPLE"`.

Implementation Status:

The maximum number of dimensions is 6.

FITS2NDF
Converts FITS files into NDFs

Description:

This application converts one or more files in the FITS format into NDFs. It can process an arbitrary FITS file to produce an NDF, using NDF extensions to store information conveyed in table and image components of the FITS file. While no information is lost, in many common cases this would prove inconvenient especially as no meaning is attached to the NDF extension components. Therefore, FITS2NDF recognises certain data products (currently IUE Final Archive, INES, ISO, and 2dF), and provides tailored conversions that map the FITS data better on to the NDF components. For instance, a FITS IMAGE extension storing data errors will have its data array transferred to the NDF's VARIANCE (after being squared). In addition, FITS2NDF can restore NDFs converted to FITS by the sister task NDF2FITS.

A more general facility is also provided to associate specified FITS extensions with NDF components by means of entries in a file (see the EXTABLE parameter).

Details of the supported special formats and rules for processing them are given in topic "Special Formats"; the general-case processing rules are described in the "Notes".

FITS2NDF can also process both external and internal compressed FITS files. The external compression applies to the whole file and FITS2NDF recognises **gzip** (`.gz`) and UNIX **compress** (`.Z`) formats. Internal compressions are where a large image is tiled and each tile is compressed. The supported formats are Rice, the IRAF PLIO, and GZIP.

Both NDF and FITS use the term extension, and they mean different things. Thus to avoid confusion in the descriptions below, the term 'sub-file' is used to refer to a FITS IMAGE, TABLE or BINTABLE Header and Data Unit (HDU).

Usage:

```
fits2ndf in out
```

Parameters:

CONTAINER = _LOGICAL (Read)

If `TRUE` causes each HDU from the FITS file to be written as a component of the HDS container file specified by the OUT parameter. Each component will be named `HDU_`*n*, where *n* is the FITS HDU number. The primary HDU is numbered 0. Primary and IMAGE HDUs will become NDFs and if the PROFITS parameter is TRUE, each NDF's FITS extension will be created from the header of the FITS sub-file. It will have the form of a primary header and may include cards inherited from the primary header. If the FITS HDU has no data array, an NDF will not be created—if PROFITS is `TRUE`, a structure of type FITS_HEADER, containing the FITS header as an array of type _CHAR*80, is created; if PROFITS is `FALSE`, no component is created. Binary and ASCII tables become components of type 'TABLE', formatted as in the general rules under "Notes" below. `[FALSE]`

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. This parameter is only relevant when WCSCOMP is "WCS" or "Both". The allowed values (case-insensitive) are:

- "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.atnf.csiro.au/people/mcalabre/WCS/) 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 described 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 (amongst others), 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 which allows the use of a wider range of celestial projections, as 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 the GILDAS manual.
- "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 applications. 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 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). [!]

EXTABLE = FILE (Read)

This specifies the name of a text file containing a table associating sub-files from a multi-extension FITS file with specific NDF components. If the null value (!) is given for EXTABLE, FITS sub-files are treated as determined by the PROEXTS parameter (see below).

An EXTABLE file contains records which may be:

- '**component specifier records**', which associate FITS sub-files with NDF components.
- '**NDFNAMES records**', which specify the names of the NDFs to be created. Normally they will be created within the top-level HDS container file specified by the OUT parameter.
- '**directive records**', which inform the table file parser.

Spaces are allowed between elements within records and blank records are ignored.

Component specifier records have the form:

component ; *sub-file_specifiers* ; *transformation_code*

where:

- *component* (case-insensitive) specifies the NDF component and is DATA, VARIANCE, QUALITY or EXTN*i.name*. The EXTN*i.name* form specifies the name *name* of an NDF extension to be created. *name* may be omitted in which case FITS_EXT_*n* is assumed, where *n* is the FITS sub-file number. *i* comprises any characters and may be omitted; it serves to differentiate component specifiers where the default name is to be used.
- *sub-file_specifiers* is a list of FITS sub-file specifiers, separated by commas. The *n*th sub-file specifier from each component specifier record forms a 'sub-file set' and each sub-file set will be used to create one NDF in the output file.
  Each sub-file specifier may be:
  - An integer, specifying the FITS Header and Data Unit (HDU) number. The primary HDU number is 0.
  - *keyword=value* (case-insensitive), specifying a FITS HDU where the specified keyword has the specified value, *e.g.* EXTNAME=IM2. The *keyword*= may be omitted in which case EXTNAME is assumed. Multiple *keyword=value* pairs separated by commas and enclosed in [] may be given as a single sub-file specifier. All the given keywords must match the sub-file header values.
  - Omitted, to indicate that the component is not required for the corresponding NDF. (Commas may be needed to maintain correct sub-file set alignment for later sub-file specifiers.) If the last character of *sub-file_specifiers* is comma, it indicates an omitted specifier at the end. Note that if a sub-file is not specified for the DATA component of an NDF, an error will be reported at closedown.
- *transformation_code* (case-insensitive) is a character string specifying a transformation to be applied to the FITS data before it is written into the NDF component. The code and preceding ";" may be omitted in which case "NONE" (no transformation) is assumed. Currently the only permitted code is "NONE".

There may be more than one component specifier record for a given component, the sub-file specifiers will be concatenated. A sub-file specifier may not span records and only the transformation code specified by the last record for the component will be effective.

NDFNAMES records have the format:

`NDFNAMES` *`name_list`*

Where *`name_list`* is a list of names for the NDFs to be created, one for each sub-file set specified by the component specifier lines. The names are separated by commas. If any of the names are omitted, the last name specified is assumed to be a root name to which an integer counter is to be added until a new name is found. If no names are specified, `EXTN_SET` is used as the root name. For example, `NDFNAMES NDF,,SET_` would result in NDFs named NDF1, NDF2, SET_1, SET_2 *etc.* up to the given number of sub-file sets.

There may be multiple NDFNAMES records, the names will be concatenated. A name may not span records and a comma as the last non-blank character indicates an omitted name.

If there is only one sub-file set, *name_list* may be '`*`', in which case the NDF will be created at the top level of the output file.

Directive records have # in column 1 and will generally be treated as comments and ignored. An exception is a record starting with '`#END`', which may optionally be used to terminate the file.

Each HDU of the FITS file is processed in turn. If it matches one of the sub-file specifiers in the table, it is used to create the specified component of the appropriate NDF in the output file; otherwise the next HDU is processed. The table is searched in sub-file set order. If a table entry is matched it is removed from the table; this means that the same FITS sub-file specifier may be repeated for another NDF component but each FITS HDU can only be used once. If sub-file specifiers remain unmatched at the end, a warning message is displayed.

A simple example of an EXTABLE is:

```
# A simple example
DATA;0,1,2,3,4,5,6
#END
```

The primary HDU and sub-files 1–6 of the FITS file will be written as the DATA components of NDFs EXTN_SET1–EXTN_SET7 within the HDS container file specified by the OUT parameter.

A contrived example, showing more of the facilities, is:

```
# A contrived example
NDFNAMES obs_
DATA; 1, EXTNAME=IM4, IM7; none
VARIANCE; 2,im5, im8
EXTN.CAL;3 ,,[extname=cal,extver=2]
#END
```

The HDS container file specified by the OUT parameter will contain three NDFs, the NDFNAMES record specifies that they will be named OBS_1, OBS_2 and OBS_3.

NDF OBS_1 will have its DATA component created from the first extension (HDU 1) of the FITS file specified by the IN parameter, and its VARIANCE from the second. NDF OBS_1 will have an extension named CAL created from the third FITS extension.

NDF OBS_2 has DATA and VARIANCE components created from the FITS sub-files whose EXTNAME keywords have the value IM4 and IM5 respectively; no CAL extension is created in OBS_2.

OBS_3 DATA and VARIANCE are created from FITS sub-files named IM7 and IM8 and the CAL extension from the FITS sub-file whose EXTNAME and EXTVER keywords have values `'CAL'` and 2 respectively.

In all cases, if the PROFITS parameter is TRUE, the NDF's FITS extension will be created from the header of the sub-file associated with the DATA component of the NDF. It will have the form of a primary header and may include cards inherited from the primary header `[!]`

FMTCNV = LITERAL (Read)

This specifies whether or not format conversion will occur. The conversion applies the values of the FITS keywords BSCALE and BZERO to the FITS data to generate the 'true' data values. This applies to IMAGE extensions, as well as the primary data array. If BSCALE and BZERO are not given in the FITS header, they are taken to be 1.0 and 0.0 respectively.

If FMTCNV=`"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 NDF will be converted from the FITS data type to _REAL or _DOUBLE in the NDF.

The special value FMTCNV=`"Native"` is a variant of `"FALSE"`, that in addition creates a scaled form of NDF array, provided the array values are scaled through BSCALE and/or BZERO keywords (*i.e.* the keywords' values are not the null 1.0 and 0.0 respectively). This NDF scaled array contains the unscaled data values, and the scale and offset.

The actual NDF data type for FMTCNV=`"TRUE"`, and the data type after applying the scale and offset for FMTCNV=`"NATIVE"` are both specified by Parameter TYPE. However, if TYPE is a blank string or null (`!`), then the choice of floating-point data type depends on the number of significant digits in the BSCALE and BZERO keywords.

FMTCNV may be a list of comma-separated values, enclosed in double quotes, to be applied to each conversion in turn. An error results if more values than the number of input FITS files are supplied. If too few are given, the last value in the list applied to all the conversions; thus a single value is applied to all the input files. If more than one line is required to enter the information at a prompt then place a `"-"` at the end of each line where a continuation line is desired. `["TRUE"]`

IN = LITERAL (Read)

The names of the FITS-format files to be converted to NDFs. It may be a list of file names or indirection specifications separated by commas and enclosed in double quotes. FITS file names may include the regular expressions (`"*"`, `"?"`, `"[a-z]"` *etc.*) but a `"[]"` construct at the end of the name is assumed to be a sub-file specifier to specify a particular FITS sub-file to be converted. (See the description of an EXTABLE file above for allowed sub-file specifiers but note that only a single `keyword=value` pair is allowed here. Note also that if a specifier contains a `keyword=value` pair, the name(s) must be enclosed in double quotes.) If you really want to have an `[a-z]`-type regular expression at the end of the filename, you can put a null sub-file specifier `"[]"` after it.

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 "#".

OUT = LITERAL (Write)

The names for the output NDFs. These may be enclosed in double quotes and specified as a list of comma-separated names, OR, using modification elements to specify output NDF names based on the input filenames. Indirection may be used if required.

The simplest modification element is the asterisk "*", which means call the output NDF files the same name (without any directory specification) as the corresponding input FITS file, but with file extension ".sdf".

Other types of modification can also occur so OUT = "x*" would mean that the output files would have the same name as the input FITS files except for an "x" prefix. You can also replace a specified string in the output filename, for example OUT="x*|cal|Starlink|" replaces the string "cal" with "Starlink" in any of the output names "x*".

Some of the options create a series of NDFs in the original NDF, which becomes just an HDS container and no longer an NDF.

PROEXTS = _LOGICAL (Read)

This governs how any extensions within the FITS file are processed in the general case. If TRUE, any FITS-file extension is propagated to the NDF as an NDF extension called FITS_EXT_*n*, where *n* is the number of the extension. If FALSE, any FITS-file extensions are ignored. The "Notes" of the general conversion contain details of where and in what form the various FITS-file extensions are stored in the NDF.

This parameter is ignored when the supplied FITS file is one of the special formats, including one defined by an EXTABLE but excluding NDF2FITS-created files, whose structure in terms of multiple FITS objects is defined. Specialist NDF extensions may be created in this case. See topic "Special Formats" for details.

It is also ignored if a sub-file is specified as the IN parameter, or Parameter CONTAINER is TRUE. [TRUE]

PROFITS = _LOGICAL (Read)

If TRUE, the headers of the FITS file are written to the NDF's FITS extension. If a specific FITS sub-file has been specified or parameter CONTAINER is TRUE or an EXTABLE is in use, the FITS extension will appear as a primary header and may include cards inherited from the primary HDU; otherwise the FITS header is written verbatim. [TRUE]

TYPE = LITERAL (Read)

The data type of the output NDF's data and variance arrays. It is normally one of the following HDS types: "_BYTE", "_WORD", "_REAL", "_INTEGER", "_INT64", "_DOUBLE", "_UBYTE", "_UWORD" corresponding to signed byte, signed word, real, integer, 64-bit integer, double precision, unsigned byte, and unsigned word. See SUN/92 for further details. An unambiguous abbreviation may be given. TYPE is ignored when COMP = "Quality" since the QUALITY component must comprise unsigned bytes (equivalent to TYPE = "_UBYTE") to be a valid NDF. The suggested default is the current value. Note that setting TYPE may result in a loss of precision, and should be used with care.

A null value (!) or blank requests that the type be propagated from the FITS (using

the BITPIX keyword); or if FMTCNV is "TRUE", the type is either _REAL or _DOUBLE depending on the precision of the BSCALE and BZERO keywords.

TYPE may be a list of comma-separated values enclosed in double quotes, that are applied to each conversion in turn. An error results if more values than the number of input FITS files are supplied. If too few are given, the last value in the list is applied to all the conversions; thus a single value is applied to all the input files. If more than one line is required to enter the information at a prompt then place a "-" at the end of each line where a continuation line is desired. [!]

WCSATTRS = LITERAL (Read)

A comma-separated list of keyword=value pairs which modify the way WCS information is extracted from the FITS headers. Each of the keywords should be an attribute of an AST FitsChan. This is the object which is responsible for interpreting the FITS WCS headers, and is described full in the documentation for the AST library (see SUN/210). For instance, to force CAR projections to be interpreted as simple linear mappings from pixel co-ordinates to celestial co-ordinates (rather than the non-linear mapping implied by the FITS-WCS conventions), use WCSATTRS="CarLin=1". A null value (!) results in all attributes using default values. [!]

WCSCOMP = LITERAL (Read)

This requests where co-ordinate information is stored in the NDF for arbitrary FITS files. FITS files from certain sources (see "Special Formats" below) adopt their own conventions such as always creating AXIS structures and not WCS, thus ignore this parameter. The allowed values are as follows.

- "Axis" — Writes co-ordinates of each element in the AXIS structure.
- "WCS" — Stores co-ordinate information in the WCS component.
- "Both" — Writes co-ordinate information in both the AXIS and WCS components.
- "None" — Omits co-ordinate information.

"WCS" is the recommended option as it offers most flexibility and many facilities such as transformations between co-ordinate systems. However, some legacy applications such as Figaro do not recognise WCS and for these "Axis" is more appropriate. If you are mixing data processing packages then you may need "Both", but care should be exercised to avoid inconsistent representations, especially if the data are exported to FITS with NDF2FITS (see its Parameter USEAXIS). ["WCS"]

Examples:

```
fits2ndf 256.fit f256 fmtcnv=f
```

This converts the FITS file called 256.fit to the NDF called f256. The data type of the NDF's data array matches that of the FITS primary data array. A FITS extension is created in f256, and FITS sub-files are propagated to NDF extensions.

```
fits2ndf 256.fit f256 fmtcnv=native type=_real
```

As above but now a _REAL type scaled data array is created, assuming that 256.fit contains scaled integer data with BITPIX=8 or 16 and non-default BSCALE and BZERO keywords.

```
fits2ndf 256.fit f256 fmtcnv=t type=_real wcscomp=axis
```

As the first example, but now a _REAL type data array is created by applying the scale and offset from BSCALE and BZERO keywords to the integer values stored in `256.fit`. Co-ordinate information is written only to the AXIS structure.

```
fits2ndf 256.fit f256 noprofits noproexts
```

As the previous example except there will be a format conversion from a FITS integer data type to floating point in the NDF using the BSCALE and BZERO keywords, and there will be no extensions written within f256.

```
fits2ndf "*.fit,p*.fits" *
```

This converts a set of FITS files given by the list `"*.fit,p*.fits"`, where `*` is the match-any-character wildcard.
The resultant NDFs take the filenames of the FITS files, so if one of the FITS files was `parker.fits`, the resultant NDF would be called `parker`. Format conversion is performed on integer data types. A FITS extension is created in each NDF and any FITS sub-files present are propagated to NDF extensions.

```
fits2ndf swp25000.mxlo mxlo25000
```

This converts the IUE MXLO FITS file called `swp25000.mxlo` to the NDF called `mxlo25000`. Should the dataset comprise both the large- and small-aperture spectra, they will be found in NDFs `mxlo25000.large` and `mxlo25000.small` respectively.

```
fits2ndf SWP19966.MXHI mxhi19966
```

This converts the IUE MXHI FITS file called `SWP19966.MXHI` to a series of NDFs within a file `mxhi19966.sdf`. Each NDF corresponds to an order. Thus for instance the one hundredth order will be in the NDF called `mxhi19966.order100`.

```
fits2ndf data/*.silo silo*|swp|| noprofits
```

This converts all the IUE SILO FITS files with file extension `.silo` in directory data to NDFs in the current directory. Each name of an NDF is derived from the corresponding FITS filename; the original name has the `"swp"` removed and `"silo"` is prefixed. So for example, `swp25000.silo` would become an NDF called silo25000. No FITS extension is created.

```
fits2ndf "abc.fit,def.fts" "fgh,ijk" fmtcnv="F,T" noproexts
```

This converts the FITS files `abc.fit` and `def.fts` to the NDFs called fgh and ijk respectively. Format conversion is applied to `abc.fit` but not to `def.fts`. FITS extensions are created in the NDFs but there are no extensions for any FITS sub-files that may be present.

```
fits2ndf 256.fit f256 fmtcnv=f encodings=DSS
```

This is the same as the first example except that it is specified that the co-ordinate system information to be stored in the WCS component of the NDF must be based on the FITS keywords written with Digitised Sky Survey (DSS) images. If these keywords are not present in the FITS header then no WCS component will be created. All the earlier examples retained the default null value for the ENCODINGS parameter, resulting in the choice of keywords being based on the contents of the FITS header (see the description of the ENCODINGS parameter for details).

```
fits2ndf 256.fit f256 fmtcnv=f encodings="DSS,native"
```

This is the same as the previous example except that if no DSS keywords are available, then the co-ordinate system information stored in the NDF's WCS component will be based on keywords written by applications which use the AST library (see SUN/210). One such application is NDF2FITS. This 'native' encoding provides a 'loss-free' means of transferring information about co-ordinate systems (*i.e.* no information is lost; this may not be the case with other encodings). If the file `256.fit` contains neither DSS nor native AST keywords, then no WCS component will be created.

```
fits2ndf "multifile.fit[extname=im3]" *
```

This will create an NDF, multifile, from the first FITS extension in file `multifile.fit` whose EXTNAME keyword has the value "im3".

```
fits2ndf multifile.fit multifile extable=table1
```

This will create a series of NDFs in the container file `multifile.sdf` according to the specifications in the EXTABLE-format file `table1`.

Notes:

Some sources of FITS files that require special conversion rules, particularly because they use binary tables, are recognised. Details of the processing for these is given within topic "Special Formats".

Two other special cases are when a particular sub-file is specified by the IN parameter and when conversion is driven by an EXTABLE file.

The general rules for the conversion apply if the FITS file is not one of the "Special Formats" (including one defined by an EXTABLE) and Parameter CONTAINER is not TRUE.

The general rules are as follows:

- The primary data array of the FITS file becomes the NDF's data array. There is an option using Parameter FMTCNV to convert integer data to floating point using the values of FITS keywords BSCALE and BZERO.
- Any integer array elements with value equal to the FITS keyword BLANK become bad values in the NDF data array. Likewise any floating-point data set to an IEEE

not-a-number value also become bad values in the NDF's data array. The BAD_PIXEL flag is set appropriately.

- NDF quality and variance arrays are not created.
- A verbatim copy of the FITS primary header is placed in the NDF's FITS extension when Parameter PROFITS is `TRUE`.
- Here are details of the processing of standard items from the the FITS header, listed by FITS keyword.
  - CRVAL*n*, CDELT*n*, CRPIX*n*, CTYPE*n*, CUNIT*n* — define the NDF's WCS and/or AXIS components (see Parameters ENCODINGS and WCSCOMP).
  - OBJECT, LABEL, BUNIT — if present are equated to the NDF's TITLE, LABEL, and UNITS components respectively.
  - LBOUND*n* — if present, this specifies the pixel origin for the n^{th} dimension.
- Additional sub-files within the FITS files are converted into extensions within the NDF if Parameter PROEXTS is `TRUE`. These extensions are named FITS_EXT_*m* for the m^{th} sub-file.
- An IMAGE sub-file is treated like the primary data array, and follows the rules give above. However, the resultant NDF is an extension of the main NDF.
- A BINTABLE or TABLE sub-file is converted into a structure of type TABLE. This has a NROWS component specifying the number of rows, and a COLUMNS structure containing a series of further structures, each of which takes its name from the label of the corresponding column in the FITS table. If there is no label for the *n*th column, the structure is called COLUMN*n*. These COLUMN structures contain a column of table data values in component DATA, preserving the original data type; and optional UNITS and COMMENT components which specify the column's units and the meaning of the column. Thus for example, for the third sub-file of NDF called ABC, the data for column called RA would be located in ABC.MORE.FITS_EXT_3.COLUMNS.RA.DATA.
- A random-group FITS file creates an NDF for each group. As they are related observations the series of NDFs are stored in a single HDS container file whose name is still given by parameter OUT. Each group NDF has component name FITS_G*n*, where *n* is the group number.

  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. The keywords of these FITS cards are derived from the values of PTYPE*m* in the main header, where *m* is the number of the group parameter.
- You can supply compressed FITS files, such as the Rice compression.
- NDF history recording is enabled in the output NDF.

Special Formats :

- NDF2FITS
  - This is recognised by the presence of an HDUCLAS1 keyword set to `'NDF'`. The conversion is similar to the general case, except the processing of FITS sub-files and HISTORY headers.

  - An IMAGE sub-file converts to an NDF VARIANCE component, provided the keyword HDUCLAS2 is present and has a value that is either `'VARIANCE'` or `'ERROR'`.
  - An IMAGE sub-file converts to an NDF QUALITY component, provided the keyword HDUCLAS2 is present and has value `'QUALITY'`.
  - FITS ASCII and binary tables become NDF extensions, however, the original structure path and data type are restored using the values of the EXTNAME and EXTTYPE keywords respectively. An extension may be an array of structures, the shape being stored in the EXTSHAPE keyword. The shapes of multi-dimensional arrays within the extensions are also restored.
  - HISTORY cards in a special format created by NDF2FITS are converted back into NDF history records. This will only work provided the HISTORY headers have not been tampered. Such headers are not transferred to the FITS airlock, when PROFITS=`TRUE`.
  - Any SMURF package's ancillary IMAGE sub-files are restored to a SMURF extension, with the original names and structure contents. Thus the global HISTORY present in each sub-file is not duplicated in each SMURF-extension NDF.

- IUE Final Archive LILO, LIHI, SILO, SIHI
  - This converts an IUE LI or SI product stored as a FITS primary data array and IMAGE extension containing the quality into an NDF. Other FITS headers are used to create AXIS structures (SI products only), and character components.
  - Details of the conversion are:
    * The primary data array of the FITS file becomes NDF main data array. The value of Parameter FMTCNV controls whether keywords BSCALE and BZERO are applied to scale the data; FMTCNV along with the number of significant characters in the keywords decide the data type of the array. It is expected that this will be _REAL if FMTCNV is `TRUE`, and _WORD otherwise.
    * The quality array comes from the IMAGE extension of the FITS file. The twos complement values are divided by -128 to obtain the most-significant 8 bits of the 14 in use. There is no check that the dimension and axis-defining FITS headers in this extension match those of the main data array. The standard indicates that they will be the same.
    * The FILENAME header value becomes the NDF's TITLE component.
    * The BUNIT header value becomes the NDF's UNITS component.
    * The CDELT*n*, CRPIX*n*, and CRVAL*n* define the axis centres. CTYPE*n* defines the axis labels. Axis information is only available for the SILO and SIHI products.
    * The primary headers may be written to the NDF's FITS extension when Parameter PROFITS is `TRUE`.

- IUE Final Archive MXLO
  - This will usually be a single 1-dimensional NDF, however, if the binary table contains two rows, a pair of NDFs are stored in a single HDS container file whose

name is specified by parameter OUT. The name of each NDF is either SMALL or LARGE depending on the size of the aperture used. Thus for OUT=ABC, the small-aperture observation will be in an NDF called ABC.SMALL.

– Only the most-significant 8 bits of the quality flags are transferred to the NDF.
– The primary headers may be written to the standard FITS airlock extension when PROFITS is TRUE.
– The conversion from binary-table columns and headers to NDF objects is as follows:

NPOINTS	Number of elements
WAVELENGTH	Start wavelength, axis label and units
DELTAW	Incremental wavelength
FLUX	Data array, label, units, bad-pixel flag
SIGMA	Data-error array
QUALITY	Quality array
remaining columns	Component in IUE_MX extension (NET and BACKGROUND are NDFs)

• IUE Final Archive MXHI

– This creates a series of NDFs within a single HDS container file whose name is specified by Parameter OUT. Each NDF corresponds to a spectral order, and may be accessed individually. The name of each NDF is ORDER followed by the spectral-order number. For instance, when OUT=SWP, the 85^{th}-order spectrum will be in an NDF called SWP.ORDER85.
– Only the most-significant 8 bits of the quality flags are transferred to the NDF.
– The primary headers may be written to the standard FITS airlock extension when PROFITS is TRUE. To save space, this appears once in the NDF specified by Parameter OUT.
– The conversion from binary-table columns and headers to NDF objects is as follows:

NPOINTS	Number of non-zero elements
WAVELENGTH	Start wavelength of the non-zero elements, label, and units
STARTPIX	Lower bound of the non-zero elements
DELTAW	Incremental wavelength
ABS_CAL	Data array, label, and units
QUALITY	Quality array
remaining columns (except 14-17)	Component in IUE_MH extension (NOISE, NET, BACKGROUND, and RIPPLE are NDFs each comprising a data array, label, units and wavelength axis)

  - It may be possible to evaluate an approximate error array for the absolutely calibrated data (ABS_CAL), by multiplying the NOISE by the ratio ABS_CAL / NET for each element.
  - The Chebyshev coefficients, limits, and scale factor in columns 14 to 17 are omitted as the evaluated background fit is propagated in BACKGROUND.

- IUE INES reduced spectra
  - This generates a single 1-dimensional NDF.
  - Only the most-significant 8 bits of the quality flags are transferred to the NDF.
  - The primary headers may be written to the standard FITS airlock extension when PROFITS is `TRUE`.
  - The conversion from binary-table columns and headers to NDF objects is as follows:

WAVELENGTH	Start wavelength, axis label and units
FLUX	Data array, label, units, bad-pixel flag
SIGMA	Data-error array
QUALITY	Quality array

- ISO CAM auto-analysis (CMAP, CMOS)
  - The CAM auto-analysis FITS products have a binary table using the 'Green Bank' convention, where rows of the table represent a series of observations, and each row is equivalent to a normal simple header and data unit. Thus most of the columns have the same names as the standard FITS keywords.
  - If there is only one observation, a normal NDF is produced; if there are more than one, the HDS container file of the supplied NDF is used to store a series of NDFs—one for each observation—called OBS*n*, where *n* is the observation number. Each observation comprises three rows in the binary table corresponding to the flux, the r.m.s. errors, and the integration times.
  - The conversion from binary-table columns to NDF objects is as follows:

ARRAY	Data, error, exposure arrays depending on the value of column TYPE
BLANK	Data blank (*i.e.* undefined value)
BUNIT	Data units
BSCALE	Data scale factor
BZERO	Data offset
CDELT*n*	Pixel increment along axis *n*
CRPIX*n*	Axis *n* reference pixel
CRVAL*n*	Axis *n* co-ordinate of reference pixel
CTYPE*n*	Label for axis *n*
CUNIT*n*	Units for axis *n*
NAXIS	Number of dimensions
NAXIS*n*	Dimension of axis *n*
remaining columns	Keyword in FITS extension

Some of these remaining columns overwrite the (global) values in the primary headers. The integration times are stored as an NDF within an extension called EXPOSURE.

The creation of axis information and extensions does not occur for the error array, as these are already generated when the data-array row in the binary table is processed.

The BITPIX column is ignored as the data type is determined through the use the TFORM*n* keyword and the value of FMTCNV in conjunction with the BSCALE and BZERO columns.

- ISO LWS auto-analysis (LWS AN)
  - The conversion from binary-table columns to NDF objects is as follows:

LSANFLX	Data array, label, and units
LSANFLXU	Data errors, hence variance
LSANDET	Quality (bits 1 to 4)
LSANSDIR	Quality (bit 5)
LSANRPID	Axis centres, labels, and units *x*-*y* positions— dimensions 1 and 2)
LSANSCNT	Axis centre, label, and unit (scan (count index—dimension 4)
LSANWAV	Axis centre, label, and unit (wavelength—dimension 3)
LSANWAVU	Axis errors (wavelength—dimension 3)
LSANFILL	not copied
remaining columns	column name in LWSAN extension

- ISO SWS auto-analysis (SWS AA)
  - The conversion from binary-table columns to NDF objects is as follows:

SWAAWAVE	Axis centres, label, and units
SWAAFLUX	Data array, label, and units
SWAASTDV	Data errors, hence variance
SWAADETN	Quality
SWAARPID	not copied
SWAASPAR	not copied
remaining columns	column name in SWSAA extension

- AAO 2dF
  - The conversion is restricted to a 2dF archive FITS file created by task NDF2FITS. FITS2NDF restores the original NDF. It creates the 2dF FIBRES extension and its constituent structures, and NDF_CLASS extension. In addition the variance, axes, and HISTORY records are converted.
  - The HISTORY propagation assumes that the FITS HISTORY headers have not been tampered.
  - Details of the conversion are:
    * The primary data array becomes the NDF's data array. Any NaN values present become bad values in the NDF.
    * The keywords CRVAL*n*, CDELT*n*, CRPIX*n*, CTYPE*n*, CUNIT*n* are used to create the NDF axis centres, labels, and units.
    * The OBJECT, LABEL, BUNIT keywords define the NDF's TITLE, LABEL, and UNITS components respectively, if they are defined.

* HISTORY cards in a special format created by NDF2FITS are converted back into NDF history records.
* The NDF variance is derived from the data array of an IMAGE extension (usually the first), if present, provided the IMAGE extension headers have an HDUCLAS2 keyword whose value is either `'VARIANCE'` or `'ERROR'`.
* The NDF_CLASS extension within the NDF is filled using the a FITS binary-table extension whose EXTNAME keyword's value is NDF_CLASS. Note: no error is reported if this extension does not exist within the FITS file.
* The FIBRES extension is created from another FITS binary table whose EXTNAME keyword's value is FIBRES. The OBJECT substructure's component names, data types, and values are taken from the binary-table columns themselves, and the components of the FIELD substructure are extracted from recognised keywords in the binary-table's header. Note: no error is reported if this extension does not exist within the FITS file.
* A FITS extension in the NDF may be written to store the primary data unit's headers when Parameter PROFITS is `TRUE`. This FITS airlock will not contain any NDF-style HISTORY records.

References :

Bailey J.A. 1997, 2dF Software Report 14, version 0.5.

NASA Office of Standards and Technology, 1994, *A User's Guide for the Flexible Image Transport System (FITS)*, version 3.1.

NASA Office of Standards and Technology, 1995, *Definition of the Flexible Image Transport System (FITS)*, version 1.1.

Related Applications :

CONVERT: MTFITS2NDF, NDF2FITS; CURSA: xcatview; KAPPA: FITSDIN, FITSIN.

GASP2NDF
Converts an image in GASP format to an NDF

Description:

This application converts a GAlaxy Surface Photometry (GASP) format file into an NDF.

Usage:

```
gasp2ndf in out shape=?
```

Parameters:

IN = FILENAME (Read)

A character string containing the name of GASP file to convert. The extension should not be given, since `".dat"` is assumed.

OUT = NDF (Write)

The name of the output NDF.

SHAPE(2) = _INTEGER (Read)

The dimensions of the GASP image (the number of columns followed by the number of rows). Each dimension must be in the range 1 to 1024. This parameter is only used if supplied on the command line, or if the header file corresponding to the GASP image does not exist or cannot be opened.

Examples:

```
gasp2ndf m31_gasp m31
```

Convert a GASP file called `m31_gasp.dat` into an NDF called m31. The dimensions of the image are taken from the header file `m31_gasp.hdr`.

```
gasp2ndf n1068 ngc1068 shape=[256,512]
```

Take the pixel values in the GASP file `n1068.dat` and create the NDF `ngc1068` with dimensions 256 columns by 512 rows.

Notes:

- A GASP image is limited to a maximum of 1024 by 1024 elements. It must be two dimensional.
- The GASP image is written to the NDF's data array. The data array has type _WORD. No other NDF components are created.
- If the header file is corrupted, you must remove the offending `".hdr"` file or specify the shape of the GASP image on the command line, otherwise the application will continually abort.

Related Applications :

CONVERT: NDF2GASP.

References :

GASP documentation (MUD/66).

GIF2NDF
Converts a GIF file into an NDF.

Description:

This Bourne-shell script converts a Graphics Interchange Format (GIF) file into an unsigned-byte (256 grey-level) NDF format file. It handles one- or two-dimensional images. The script uses various NETPBM utilities to produce a FITS file, flipped top to bottom, and then FITS2NDF to produce the final NDF. Error messages are converted into Starlink style (preceded by `!`).

Usage:

```
gif2ndf in [out]
```

Parameters:

IN = FILENAME (Read)
The name of the GIF file to be converted. (A `.gif` name extension is assumed if omitted.)

OUT = NDF (Write)
The name of the NDF to be generated (without the `.sdf` extension). If the OUT parameter is omitted, the value of the IN parameter is used.

Examples:

```
gif2ndf old new
```

This converts the GIF file `old.gif` into an NDF called new (in file `new.sdf`).

```
gif2ndf horse
```

This converts the GIF file `horse.gif` into an NDF called horse (in file `horse.sdf`).

Notes:

The following points should be remembered:

- This initial version of the script generates images with at most 256 grey levels. It does not use the image colour lookup table.
- Input image filenames must have the file extension `.gif`.
- The NETPBM utilities `giftopnm`, `ppmtopgm`, `pnmflip` and `pnmtofits` must be available on your PATH.

Related Applications :

CONVERT: NDF2GIF.

IRAF2NDF
Converts an IRAF image to an NDF

Description:

This application converts an IRAF image to an NDF. See the "Notes" for details of the conversion.

Usage:

```
iraf2ndf in out
```

Parameters:

IN = LITERAL (Read)
The name of the IRAF image. Note that this excludes the ".imh" file extension.

OUT = NDF (Write)
The name of the NDF to be produced.

PROFITS = _LOGICAL (Read)
If TRUE, the user headers of the IRAF file are written verbatim to the NDF's FITS extension. Any IRAF history records are also appended to the FITS extension. The FITS extension is not created if there are no user headers present in the IRAF file. [TRUE]

PROHIS = _LOGICAL (Read)
This parameter decides whether or not to create NDF HISTORY records. Only the IRAF headers with keyword HISTORY, and which originated from NDF HISTORY records are used. If PROHIS=TRUE, NDF HISTORY records are created. [TRUE]

Examples:

```
iraf2ndf ell_galaxy new_galaxy
```

Converts the IRAF image ell_galaxy (comprising files ell_galaxy.imh and ell_galaxy.pix) to an NDF called new_galaxy.

```
iraf2ndf ell_galaxy new_galaxy noprofits noprohis
```

As above, except no FITS extension is created, and NDF-style HISTORY lines in ell_galaxy.imh are not transferred to HISTORY records in NDF new_galaxy.

Notes:

The rules for the conversion are as follows:

- The NDF is created with bounds determined by any LBOUND*n* keywords in the IRAF image header.
- The NDF data array is copied from the ".pix" file.

- The title of the IRAF image (object i_title in the ".imh" header file) becomes the NDF title. Likewise headers OBJECT and BUNIT become the NDF label and units respectively.
- The pixel origin is set if any LBOUND*n* headers are present.
- Lines from the IRAF image header file may be transferred to the FITS extension of the NDF, when PROFITS=TRUE. Any compulsory FITS keywords that are missing are added. Certain other keywords are not propagated. These are the IRAF "Mini World Co-ordinate System" (MWCS) keywords WCSDIM, DC_FLAG, WAT*d_nnn* (*d* is dimension, *nnn* is the line number). Certain NDF-style HISTORY lines in the header are also be ignored when PROHIS=TRUE (see two notes below).
- When PROFITS=TRUE, lines from the HISTORY section of the IRAF image are also extracted and added to the NDF's FITS extension as FITS HISTORY lines. Two extra HISTORY lines are added to record the original name of the image and the date of the format conversion.
- When PROHIS=TRUE, any HISTORY lines in the IRAF headers, which originated from an NDF2IRAF conversion of NDF HISTORY records. Such headers are not transferred to the FITS airlock, when PROFITS=TRUE.
- Most axis information can be propagated either from standard FITS-like keywords, or certain MCWS headers. Supported systems and formats are listed below.
  - FITS
    - linear
    - log-linear
  - Equispec
    - linear
    - log-linear
  - Multispec
    - linear
    - log-linear
    - Chebyshev and Legendre polynomials
    - Linear and cubic Spline
    - Explicit list of co-ordinates

  However, for Multispec axes, only the first (spec1) axis co-ordinates are transferred to the NDF AXIS centres. Any spec2 ... spec*n* co-ordinates, present when the data array is not one-dimensional or multiple fits have been stored, are ignored. The weights for multiple fits are thus also ignored. The data type of the axis centres is _REAL or _DOUBLE depending on the number of significant digits in the co-ordinates or coefficients.

  The axis labels and units are also propagated, where present, to the NDF AXIS structure. In the FITS system, these are derived from the CTYPE*n* and CUNIT*n* keywords. In the MWCS, these components originate in the label and units parameters.

  The redshift correction, when present, is applied to the MCWS axis co-ordinates.

Related Applications :

CONVERT: NDF2IRAF.

Pitfalls :

- Bad pixels in the IRAF image are not replaced.
- Some of the routines required for accessing the IRAF header file are written in SPP. Macros are used to find the start of the header line section, this constitutes an 'Interface violation' as these macros are not part of the IMFORT interface specification. It is possible that these may be changed in the future, so beware.

References :

IRAF User Handbook Volume 1A: *A User's Guide to FORTRAN Programming in IRAF, the IMFORT Interface*, by Doug Tody.

Keywords :

CONVERT, IRAF

Implementation Status:

- Only handles one-, two-, and three-dimensional IRAF files.
- The NDF produced has type _WORD or _REAL corresponding to the type of the IRAF image. (The IRAF IMFORT FORTRAN subroutine library only supports these data types: signed words and real.) The pixel type of the image can be changed from within IRAF using the 'chpixtype' task in the 'images' package.
- See "Release Notes" for IRAF Version compatibility.

IRCAM2NDF
Converts an IRCAM data file to a series of NDFs

Description:

This applications converts an HDS file in the IRCAM format listed in IRCAM User Note 11 to one or more NDFs. See the "Notes" for a detailed list of the rules of the conversion.

Usage:

```
ircam2ndf in prefix obs [config]
```

Parameters:

CONFIG = LITERAL (Read)

The choice of data array to place in the NDF. It can have one of the following configuration values:

- "STARE" — the image of the object or sky;
- "CHOP" — the chopped image of the sky;
- "KTCSTARE" — the electronic pedestal or bias associated with the stare image of the object or sky;
- "KTCCHOP" — the electronic pedestal or bias associated with the chopped image of the sky.

Note that at the time of writing chopping has not been implemented for IRCAM. For practical purposes CONFIG="STARE" should be used. The suggested default is the current value. ["STARE"]

FMTCNV = _LOGICAL (Read)

This specifies whether or not format conversion may occur. If FMTCNV is FALSE, the data type of the data array in the NDF will be the same as that in the IRCAM file, and there is no scale factor and offset applied. If FMTCNV is TRUE, whenever the IRCAM observation has non-null scale and offset values, the observation data array will be converted to type _REAL in the NDF, and the scale and offset applied to the input data values to give the 'true' data values. A null scale factor is 1 and a null offset is 0. [FALSE]

IN = IRCAM (Read)

The name of the input IRCAM file to convert to NDFs. The suggested value is the current value.

OBS() = LITERAL (Read)

A list of the observation numbers to be converted into NDFs. Observations are numbered consecutively from 1 up to the actual number of observations in the IRCAM file. Single observations or a set of adjacent observations may be specified, *e.g.* entering [4,6-9,12,14-16] will read observations 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.)

If you wish to extract all the observations enter the wildcard *. 5-* will read from 5 to the last observation. The processing will continue until the last observation is converted. The suggested value is the current value.

PREFIX = LITERAL (Read)
The prefix of the output NDFs. The name of an NDF is the prefix followed by the observation number. The suggested value is the current value.

Examples:

```
ircam2ndf ircam_27aug89_1cl rhooph obs=*
```

This converts the IRCAM data file called `ircam_27aug89_1cl` into a series of NDFs called rhooph1, rhooph2 *etc.* There is no format conversion applied.

```
ircam2ndf ircam_27aug89_1cl rhooph [32,34-36] fmtcnv
```

This converts four observations in the IRCAM data file called `ircam_27aug89_1cl` into NDFs called rhooph32, rhooph34, rhooph35, rhooph36. The scale and offset are applied.

```
ircam2ndf in=ircam_04nov90_1c config="KTC" obs=5 prefix=bias
```

This converts the fifth observation in the IRCAM data file called `ircam_04nov90_1c.sdf` into an NDF called bias5 containing the electronic pedestal in its data array. There is no format conversion applied.

Notes:

The rules for the conversion of the various components are as follows:

IRCAM file		NDF	Comments
.OBS.PHASEA.DATA_ARRAY	⇒	.DATA_ARRAY	when Parameter CONFIG="STARE"
.OBS.PHASEB.DATA_ARRAY	⇒	.DATA_ARRAY	when Parameter CONFIG="CHOP"
.OBS.KTCA.DATA_ARRAY	⇒	.DATA_ARRAY	when Parameter CONFIG="KTCSTARE"
.OBS.KTCB.DATA_ARRAY	⇒	.DATA_ARRAY	when Parameter CONFIG="KTCCHOP"
.OBS.DATA_LABEL	⇒	.LABEL	
.OBS.DATA_UNITS	⇒	.UNITS	
.OBS.TITLE	⇒	.TITLE	If .OBS.TITLE is a blank string, OBS.DATA_OBJECT is copied to the NDF title instead.
.OBS.AXIS1_LABEL	⇒	.AXIS(1).LABEL	
.OBS.AXIS2_LABEL	⇒	.AXIS(2).LABEL	
.OBS.AXIS1_UNITS	⇒	.AXIS(1).UNITS	
.OBS.AXIS2_UNITS	⇒	.AXIS(2).UNITS	

IRCAM file		NDF	Comments
.GENERAL.INSTRUMENT.PLATE_SCALE becomes the increment between the axis centres, with co-ordinate (0.0,0.0) located at the image centre. The NDF axis units both become "arcseconds".			
.GENERAL	⇒	.MORE.IRCAM.GENERAL	
.GENERAL.x	⇒	.MORE.IRCAM.GENERAL.x	
.GENERAL.x.y	⇒	.MORE.IRCAM.GENERAL.x.y	
.OBS.x	⇒	.MORE.IRCAM.OBS.x	This excludes the components of OBS already listed above and DATA_BLANK.

- The data types of the IRCAM GENERAL structures have not been propagated to the NDF IRCAM extensions, because it would violate the rules of SGP/38. In the IRCAM file these all have the same type STRUCTURE. The new data types are as follows:

Extension Name	Data Type
IRCAM.GENERAL	IRCAM_GENERAL
IRCAM.GENERAL.INSTRUMENT	IRCAM_INSTRUM
IRCAM.GENERAL.ID	IRCAM_ID
IRCAM.GENERAL.TELESCOPE	IRCAM_TELESCOPE

- Upon completion the number of observations successfully converted to NDFs is reported.

Bad-pixel Handling :

Elements of the data array equal to the IRCAM component .OBS.DATA_BLANK are replaced by the standard bad value.

Implementation Status:

- The data array in the NDF is in the primitive form.
- The application aborts if the data array chosen by parameter CONFIG does not exist in the observation.

MTFITS2NDF
Converts FITS magnetic tape files into NDFs.

Description:

This application converts files from a FITS tape into NDFs by using shell commands **mt** and **dd** to position the tape and convert the selected tape files into FITS disk files and then using FITS2NDF to produce the NDFs. The intermediate FITS files may be saved.

Usage:

```
mtfits2ndf in out block=n [of=fits_file] [<fits2ndf_pars>]
```

Parameters:

BLOCK = _INTEGER (Read)
The FITS blocking factor, *i.e.* the block size on the tape is this value multiplied by the standard FITS block size. The suggested default is 10.

IN = DEVICE (Read)
The name of a tape device. For correct tape positioning, a no-rewind device must be used. The device name may have file specifiers appended, separated by commas and enclosed in `[]`. The file specifiers indicate which files from the tape are to be processed. For example: `[2]` indicates the second file on the tape. `[4-6]` indicates files 4 to 6. `[5-]` indicates file 5 to the last file on the tape. `[1,3-5,7-]` indicates files 1,3,4,5, and 7 to the end of the tape. If no file specifiers are given, all files on the tape will be processed.

OF = LITERAL (Read)
Name(prefix) of the intermediate FITS file(s) copied from the tape. Only set this if you want to save the intermediate FITS file(s). If a number of files are being produced, the name should contain a `*`, which will be replaced by the corresponding FITS tape file number. If OF is not specified, `mtf2ndf*.fits` will be used and deleted. (See also "Notes").

OUT = NDF (Write)
The name of the NDF(s) to be produced by FITS2NDF. This is passed to FITS2NDF but only a single element string can be specified—it can contain the matching patterns allowed for FITS2NDF, for example `"*"`.

`<fits2ndf_pars>`
Other parameters will be passed to FITS2NDF—see the description of FITS2NDF.

Notes:

- This application is a `tcsh` script which calls an ADAM A-task. CONVERT startup sets alias `mtfits2ndf` to '`tcsh mtfits2ndf.tcsh`', and `tcsh` must be on the user's PATH.
- The string specified for the intermediate FITS file name(s) will be presented as the IN parameter for the FITS2NDF call. All files matching the string will be used, whether or not they were produced in this run. (See the FITS2NDF description for details.)

Examples:

```
mtfits2ndf /dev/nst0[2] f256 block=10 fmtcnv=f
```

This converts the second file on the tape on device `/dev/nst0` to an NDF called `f256`. The FITS blocking factor of the tape is 10. As a result of the parameters passed to FITS2NDF, the data type of the NDF's data array matches that of the FITS primary data array, a FITS extension is created in `f256`, and FITS sub-files are propagated to NDF extensions.

```
mtfits2ndf /dev/nst0 * block=1 of=ral256_*.fit
```

Will convert each file on the tape on device `/dev/nst0` (with a blocking factor of 1) to FITS disk files named `ral256_*.fit`, where `*` is replaced by each tape file number. The FITS files will be converted to NDFs named `ral256_*` and retained.

```
mtfits2ndf
```

The user is prompted for the input device, the output NDF name and the FITS blocking factor. All other parameters are defaulted.

References :

NASA Office of Standards and Technology, 1997, *A User's Guide for the Flexible Image Transport System (FITS)*, Version 4.0.

NASA Office of Standards and Technology, 1999, *Definition of the Flexible Image Transport System (FITS)*.

Deficiencies :

- Facilities for naming multiple output files are limited.
- The command is not available from ICL, nor as an option in the automatic (on-the-fly) conversion system.
- Extensions within a FITS file may not be specified. However, it is possible to pass an EXTABLE parameter to the FITS2NDF operation to select extensions.
- Only tape devices for which the **mt** and **dd** commands will work may be used.

Related Applications :

KAPPA: FITSDIN, FITSIN; CONVERT: FITS2NDF.

NDF2ASCII
Converts an NDF to a text file

Description:

This application converts an NDF to a Fortran text file. Only one of the array components may be copied to the output file. Preceding the data there is an optional header consisting of either the FITS extension with the values of certain keywords replaced by information derived from the NDF, or a minimal FITS header also derived from the NDF. The output file uses LIST carriagecontrol.

Usage:

```
ndf2ascii in out [comp] [reclen] noperec=?
```

Parameters:

COMP = LITERAL (Read)

The NDF component to be copied. It may be "Data", "Quality" or "Variance". ["Data"]

FITS = _LOGICAL (Read)

If TRUE, any FITS extension is written to start of the output file, unless there is no extension whereupon a minimal FITS header is written to the ASCII file. [FALSE]

FIXED = _LOGICAL (Read)

When FIXED is TRUE, the output file allocates a fixed number of characters per data value. The number of characters chosen is the minimum that prevents any loss of precision, and hence is dependent on the data type of the NDF array. These widths in characters for each HDS data type are as follows: _UBYTE, 3; _BYTE, 4; _UWORD, 5; _WORD, 6; _INTEGER, 11; _INT64, 20; _REAL, 16; and _DOUBLE, 24. The record length is the product of the number of characters per value plus one (for a delimiting space), times the number of values per record given by parameter NOPEREC, up to a maximum of 512.

When FIXED is FALSE, data values are packed as efficiently as possible within each record. The length of each record is given by Parameter RECLEN. [FALSE]

IN = NDF (Read)

Input NDF data structure. The suggested default is the current NDF if one exists, otherwise it is the current value.

NOPEREC = _INTEGER (Read)

The number of data values per record of the output file, when FIXED is TRUE. It should be positive on UNIX platforms. The suggested default is the current value, or 8 when there is not one. The upper limit is given by 512 divided by the number of characters per value plus 1 (see Parameter FIXED).

OUT = FILENAME (Write)

Name of the output formatted Fortran file. The file will normally have variable-length records when there is a header, but always fixed-length records when there is no header.

RECLEN = _INTEGER (Read)
The maximum record length in bytes (characters) of the output file. This has a maximum length of 512 (for efficiency reasons), and must be greater than 31 on UNIX systems. The lower limit is further increased to 80 when there is a FITS header to be copied. It is only used when FIXED is FALSE and will default to the current value, or 132 if there is no current value. []

Examples:

```
ndf2ascii cluster cluster.dat
```

This copies the data array of the NDF called cluster to a text file called cluster.dat. The maximum recordlength of cluster.dat is 132 bytes, and the data values are packed into these records as efficiently as possible.

```
ndf2ascii cluster cluster.dat v
```

This copies the variance of the NDF called cluster to a text file called cluster.dat. The maximum recordlength of cluster.dat is 132 bytes, and the variance values are packed into these records as efficiently as possible.

```
ndf2ascii cluster cluster.dat fixed noperec=12
```

This copies the data array of the NDF called cluster to a text file called cluster.dat. There are twelve data values per record in cluster.dat.

```
ndf2ascii out=ndf234.dat fits reclen=80 in=@234
```

This copies the data array of the NDF called 234 to a text file called ndf234.dat. The maximum recordlength of ndf234.dat is 80 bytes, and the data values are packed into these records as efficiently as possible. If there is a FITS extension, it is copied to ndf234.dat with substitution of certain keywords, otherwise a minimal FITS header is produced.

Notes:

The details of the conversion are as follows:

- the NDF array as selected by COMP is written to the ASCII file in records following an optional header. When FIXED is FALSE all records are padded out to the recordlength.
- The NDF array elements are written in Fortran order, *i.e.* the first dimension varies fastest, followed by the second dimension and so on. For example, a 2x2x2-element cube's indices will appear in the order (1,1,1), (2,1,1), (1,2,1), (2,2,1), (1,1,2), (2,1,2), (1,2,2), (2,2,2).
- HISTORY is not propagated.
- ORIGIN information is lost.
- When a header is to be made, it is composed of FITS-like card images as follows:
  - The number of dimensions of the data array is written to the keyword NAXIS, and the actual dimensions to NAXIS1, NAXIS2 *etc.* as appropriate.

  - If the NDF contains any linear axis structures the information necessary to generate these structures is written to the FITS-like headers. For example, if a linear AXIS(1) structure exists in the input NDF the value of the first data point is stored with the keyword CRVAL1, and the incremental value between successive axis data is stored in keyword CDELT1. By definition the reference pixel is 1.0 and is stored in keyword CRPIX1. If there is an axis label it is written to keyword CTYPE1, and axis unit is written to CUNIT1. (Similarly for AXIS(2) structures *etc.*) FITS does not have a standard method of storing axis widths and variances, so these NDF components will not be propagated to the header. Non-linear axis data arrays cannot be represented by CRVAL*n* and CDELT*n*, and must be ignored.
  - If the input NDF contains TITLE, LABEL, or UNITS components these are stored with the keywords TITLE, LABEL, or BUNIT respectively.
  - If the input NDF contains a FITS extension, the FITS items may be written to the FITS-like header, with the following exceptions:
    * BITPIX is derived from the type of the NDF data array, and so it is not copied from the NDF FITS extension.
    * NAXIS, and NAXIS*n* are derived from the dimensions of the NDF data array as described above, so these items are not copied from the NDF FITS extension.
    * The TITLE, LABEL, and BUNIT descriptors are only copied if no TITLE, LABEL, and UNITS NDF components respectively have already been copied into these headers.
    * The CDELT*n*, CRVAL*n*, CTYPE*n*, CUNIT*n*, and CRTYPE*n* descriptors in the FITS extension are only copied if the input NDF contained no linear axis structures.
    * The standard order of the FITS keywords is preserved, thus BITPIX, NAXIS and AXIS*n* appear immediately after the first card image, which should be SIMPLE.
    * BSCALE and BZERO in a FITS extension are copied when BITPIX is positive, *i.e.* the array is not floating-point.
  - An extra header record with keyword UNSIGNED and logical value T is added when the array data type is one of the HDS unsigned integer types. This is done because standard FITS does not support unsigned integers, and allows (in conjunction with BITPIX) applications reading the unformatted file to determine the data type of the array.
  - The last header record card will be the standard FITS END.
- Other extensions are not propagated.

Related Applications :

CONVERT: ASCII2NDF; KAPPA: TRANDAT; FIGARO: ASCIN and ASCOUT.

Implementation Status:

- All non-complex numeric data types are supported.
- The value of bad pixels is not written to a FITS-like header record with keyword BLANK.

NDF2DA
Converts an NDF to a direct-access unformatted file

Description:

This application converts an NDF to a direct-access unformatted file, which is equivalent to fixed-length records, or a data stream suitable for reading by C routines. Only one of the array components may be copied to the output file.

Usage:

```
ndf2da in out [comp] [noperec]
```

Parameters:

COMP = LITERAL (Read)
The NDF component to be copied. It may be "Data", "Quality" or "Variance". ["Data"]

IN = NDF (Read)
Input NDF data structure. The suggested default is the current NDF if one exists, otherwise it is the current value.

NOPEREC = _INTEGER (Read)
The number of data values per record of the output file. It must be positive. The suggested default is the current value. [The first dimension of the NDF]

OUT = FILENAME (Write)
Name of the output direct-access unformatted file.

Examples:

```
ndf2da cluster cluster.dat
```

This copies the data array of the NDF called cluster to a direct-access unformatted file called `cluster.dat`. The number of data values per record is equal to the size of the first dimension of the NDF.

```
ndf2da cluster cluster.dat v
```

This copies the variance of the NDF called cluster to a direct-access unformatted file called `cluster.dat`. The number of variance values per record is equal to the size of the first dimension of the NDF.

```
ndf2da cluster cluster.dat noperec=12
```

This copies the data array of the NDF called cluster to a direct-access unformatted file called cluster.dat. There are twelve data values per record in `cluster.dat`.

Notes:

The details of the conversion are as follows:

- the NDF array as selected by COMP is written to the unformatted file in records.
- all other NDF components are lost.

Related Applications :

CONVERT: DA2NDF.

NDF2DST
Converts an NDF to a Figaro (Version 2) DST file

Description:

This application converts an NDF to a FIGARO (Version 2) 'DST' file. The rules for converting the various components of a DST are listed in the "Notes". Since both are hierarchical formats most files can be be converted with little or no information lost.

Usage:

```
ndf2dst in out
```

Parameters:

IN = NDF (Read)

Input NDF data structure. The suggested default is the current NDF if one exists, otherwise it is the current value.

OUT = Figaro (Write)

Output Figaro file name. This excludes the file extension. The file created will be given extension ".dst".

Examples:

```
ndf2dst old new
```

This converts the NDF called old (in file old.sdf) to the Figaro file new.dst.

```
ndf2dst spectre spectre
```

This converts the NDF called spectre (in file spectre.sdf) to the Figaro file spectre.dst.

Notes:

The rules for the conversion are as follows:

NDF		Figaro file	Comments
Main data array	⇒	.Z.DATA	
Imaginary array	⇒	.Z.IMAGINARY	
Bad-pixel flag	⇒	.Z.FLAGGED	
Units	⇒	.Z.UNITS	
Label	⇒	.Z.LABEL	
Variance	⇒	.Z.ERRORS	After processing
Quality	⇒		It is not copied directly though bad values indicated by QUALITY flags will be flagged in .Z.DATA in addition to any flagged values actually in the input main data array. .Z.FLAGGED is set accordingly.

NDF		Figaro file	Comments
Title	⇒	.OBS.OBJECT	
AXIS(1) structure	⇒	.X	
AXIS(1) Data	⇒	.X.DATA_ARRAY	unless there is a DATA component of AXIS(1).MORE.FIGARO to allow for a non-1-dimensional array
AXIS(1) Variance	⇒	.X.VARIANCE	unless there is a VARIANCE component of AXIS(1).MORE.FIGARO to allow for a non-1-dimensional array
AXIS(1) Width	⇒	.X.WIDTH	unless there is a WIDTH component of AXIS(1).MORE.FIGARO to allow for a non-1-dimensional array
AXIS(1) Units	⇒	.X.UNITS	
AXIS(1) Label	⇒	.X.LABEL	
AXIS(1).MORE.FIGARO.xxx	⇒	.X.xxx	
			Similarly for AXIS(2), . . . , AXIS(6) which are renamed to .Y .T .U .V or .W
FIGARO extension:			
.MORE.FIGARO.MAGFLAG	⇒	.Z.MAGFLAG	
.MORE.FIGARO.RANGE	⇒	.Z.RANGE	
.MORE.FIGARO.SECZ	⇒	.OBS.SECZ	
.MORE.FIGARO.TIME	⇒	.OBS.TIME	
.MORE.FIGARO.xxx	⇒	.xxx	recursively
FITS extension:			
.MORE.FITS			
Items	⇒	.FITS.xxx	
Comments	⇒	.COMMENTS.xxx	
Other extensions:			
.MORE.other	⇒	.MORE.other	

Related Applications :

CONVERT: DST2NDF.

NDF2FITS
Converts NDFs into FITS files

Description:

This application converts one or more NDF datasets into FITS-format files. NDF2FITS stores any variance and quality information in IMAGE extensions ('sub-files') within the FITS file; and it uses binary tables to hold any NDF-extension data present, except for the FITS-airlock extension, which may be merged into the output FITS file's headers.

You can select which NDF array components to export to the FITS file, and choose the data type of the data and variance arrays. You can control whether or not to propagate extensions and history information.

The application also accepts NDFs stored as top-level components of an HDS container file.

Both NDF and FITS use the term extension, and they mean different things. Thus to avoid confusion in the descriptions below, the term 'sub-file' is used to refer to a FITS IMAGE, TABLE or BINTABLE Header and Data Unit (HDU).

Usage:

```
ndf2fits in out [comp] [bitpix] [origin]
```

Parameters:

ALLOWTAB = _LOGICAL (Read)

If `TRUE`, tables of world co-ordinates may be written using the TAB algorithm as defined in the FITS-WCS Paper III. Examples where such a table might be present in the WCS include wavelengths of pre-scrunched spectra, and the presence of distortions that prevent co-ordinates being defined by analytical expressions. Since many FITS readers are yet to support the TAB algorithm, which uses a FITS binary-table extension to store the co-ordinates, this parameter permits this facility to be disabled. `[TRUE]`

AXISORDER = LITERAL (Read)

Specifies the order of WCS axes within the output FITS header. It can be either null (`!`), `"Copy"` or a space-separated list of axis symbols (case insensitive). If it is null, the order is determined automatically so that the *i*th WCS axis is the WCS axis that is most nearly parallel to the *i*th pixel axis. If it is `"Copy"`, the *i*th WCS axis in the FITS header is the *i*th WCS axis in the NDF's current WCS Frame. Otherwise, the string must be a space-separated list of axis symbols that gives the order for the WCS axes. An error is reported if the list does not contain any of the axis symbols present in the current WCS Frame, but no error is reported if the list also contains other symbols. `[!]`

BITPIX = LITERAL (Read)

The FITS bits-per-pixel (BITPIX) value for each conversion. This specifies the data type of the output FITS file. Permitted values are: `8` for unsigned byte, `16` for signed word, `32` for integer, `64` for 64-bit integer, `-32` for real, `-64` for double precision. There are three other special values.

- BITPIX=0 will cause the output file to have the data type equivalent to that of the input NDF.
- BITPIX=-1 requests that the output file has the data type corresponding to the value of the BITPIX keyword in the NDF's FITS extension. If the extension or BITPIX keyword is absent, the output file takes the data type of the input array.
- BITPIX="Native" requests that any scaled arrays in the NDF be copied to the scaled data type. Otherwise behaviour reverts to BITPIX=-1, which may in turn be effectively BITPIX=0. The case-insensitive value may be abbreviated to "n".

BITPIX values must be enclosed in double quotes and may be a list of comma-separated values to be applied to each conversion in turn. An error results if more values than the number of input NDFs are supplied. If too few are given, the last value in the list is applied to the remainder of the NDFs; thus a single value is applied to all the conversions. The given values must be in the same order as that of the input NDFs. Indirection through a text file may be used. If more than one line is required to enter the information at a prompt then type a "-" at the end of each line where a continuation line is desired. [0]

CHECKSUM = _LOGICAL (Read)

If TRUE, each header and data unit in the FITS file will contain the integrity-check keywords CHECKSUM and DATASUM immediately before the END card. [TRUE]

COMP = LITERAL (Read)

The list of array components to attempt to transfer to each FITS file. The acceptable values are "D" for the main data array "V" for variance, "Q" for quality, or any permutation thereof. The special value "A" means all components, *i.e.* COMP="DVQ". Thus COMP="VD" requests that both the data array and variance are to be converted if present. During processing at least one, if not all, of the requested components must be present, otherwise an error is reported and processing turns to the next input NDF. If the DATA component is in the list, it will always be processed first into the FITS primary array. The order of the variance and quality in COMP decides the order they will appear in the FITS file.

The choice of COMP may affect automatic quality masking. See "Quality Masking" for the details.

COMP may be a list of comma-separated values to be applied to each conversion in turn. The list must be enclosed in double quotes. An error results if more values than the number of input NDFs are supplied. If too few are given, the last value in the list is applied to the remainder of the NDFs; thus a single value is applied to all the conversions. The given values must be in the same order as that of the input NDFs. Indirection through a text file may be used. If more than one line is required to enter the information at a prompt then type a "-" at the end of each line where a continuation line is desired. ["A"]

CONTAINER = _LOGICAL (Read)

If TRUE, the supplied IN files are any multi-NDF HDS container files, in which the NDFs reside as top-level components. This option is primarily intended to support the UKIRT format where the NDFs are named .I*n*, $n >= 1$, and one named HEADER containing global metadata in its FITS airlock. The .I*n* NDFs may also contain FITS airlocks, storing metadata pertinent to that NDF, such as observation times. The individual NDFs often represent separate integrations nodded along a slit or spatially.

Note that this is not a group, so a single value applies to all the supplied input files. [FALSE]

DUPLEX = _LOGICAL (Read)
This qualifies the effect of PROFITS=TRUE. DUPLEX=FALSE means that the airlock headers only appear with the primary array. DUPLEX=TRUE, propagates the FITS airlock headers for other array components of the NDF. [FALSE]

ENCODING = LITERAL (Read)
Controls the FITS keywords which will be used to encode the World Co-ordinate System (WCS) information within the FITS header. The value supplied should be one of the encodings listed in the "World Co-ordinate Systems" section below. In addition, the value "Auto" may also be supplied, in which case a suitable default encoding is chosen based on the contents of the NDF's FITS extension and WCS component. ["Auto"]

IN = LITERAL (Read)
The names of the NDFs to be converted into FITS format. It may be a list of NDF names or direction specifications separated by commas and enclosed in double quotes. NDF names may include wild-cards ("*", "?"). 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 "#".

NATIVE = _LOGICAL (Read)
If a TRUE value is given for Parameter NATIVE, then World Co-ordinate System (WCS) information will be written to the FITS header in the form of a 'native' encoding (see "World Co-ordinate Systems" below). This will be in addition to the encoding specified using Parameter ENCODING, and will usually result in two descriptions of the WCS information being stored in the FITS header (unless ENCODING parameter produces a native encoding in which case only one native encoding is stored in the header). Including a native encoding in the header will enable other AST-based software (such as FITS2NDF) to reconstruct the full details of the WCS information. The other non-native encodings will usually result in some information being lost. [FALSE]

MERGE = _LOGICAL (Read)
Whether or not to merge the FITS-airlocks' headers of the header NDF of a UKIRT multi-NDF container file with its sole data NDF into the primary header and data unit (HDU). This parameter is only used when CONTAINER is TRUE; and when the container file only has two component NDFs: one data NDF of arbitrary name, and the other called HEADER that stores the global headers of the dataset. [TRUE]

ORIGIN = LITERAL (Read)
The origin of the FITS files. This becomes the value of the ORIGIN keyword in the FITS headers. If a null value is given it defaults to "Starlink Software". [!]

OUT = LITERAL (Write)
The names for the output FITS files. These may be enclosed in double quotes and specified as a list of comma-separated names, or they may be created automatically on the basis of the input NDF names. To do this, the string supplied for this parameter should include an asterisk "*". This character is a token which represents the name of the corresponding input NDF, but with a file type of ".fit" instead of ".sdf", and

with no directory specification. Thus, simply supplying "*" for this parameter will create a group of output files in the current directory with the same names as the input NDFs, but with file type ".fit". You can also specify some simple editing to be performed. For instance, "new-*|.fit|.fits|" will add the string "new-" to the start of every file name, and will substitute the string ".fits" for the original string ".fit".

NDF2FITS will not permit you to overwrite an existing FITS file, unless you supply an exclamation mark prefix (suitably escaped if you are using a UNIX shell).

PROEXTS = _LOGICAL (Read)

If TRUE, the NDF extensions (other than the FITS extension) are propagated to the FITS files as FITS binary-table sub-files, one per structure of the hierarchy. [FALSE]

PROFITS = _LOGICAL (Read)

If TRUE, the contents of the FITS extension of the NDF are merged with the header information derived from the standard NDF components. See the "Notes" for details of the merger. [TRUE]

PROHIS = _LOGICAL (Read)

If TRUE, any NDF history records are written to the primary FITS header as HISTORY cards. These follow the mandatory headers and any merged FITS-extension headers (see Parameter PROFITS). [TRUE]

PROVENANCE = LITERAL (Read)

This controls the export of NDF provenance information to the FITS file. Allowed values are as follows.

- "None" — No provenance is written.
- "CADC" — The CADC headers are written. These record the number and paths of both the direct parents of the NDF being converted, and its root ancestors (the ones without parents). It also modifies the PRODUCT keyword to be unique for each FITS sub-file.
- "Generic" — Encapsulates the entire PROVENANCE structure in FITS headers in sets of five character-value indexed headers. there is a set for the current NDF and each parent. See Section "Provenance" for more details.

["None"]

USEAXIS = _LOGICAL (Read)

Whether or not to export AXIS co-ordinates to an alternate world co-ordinate representation in the FITS headers. Such an alternate may require a FITS sub-file to store lookup tables of co-ordinates using the -TAB projection type. The default null value requests no AXIS information be stored unless the current NDF contains AXIS information but no WCS. An explicit TRUE or FALSE selection demands the chosen setting irrespective of how the current NDF stores co-ordinate information. [!]

Examples:

```
ndf2fits horse logo.fit d
```

This converts the NDF called horse to the new FITS file called logo.fit. The data type of the FITS primary data array matches that of the NDF's data array. The FITS extension in the NDF is merged into the FITS header of logo.fit.

```
ndf2fits horse !logo.fit d proexts
```

This converts the NDF called horse to the FITS file called `logo.fit`. An existing `logo.fit` will be overwritten. The data type of the FITS primary data array matches that of the NDF's data array. The FITS extension in the NDF is merged into the FITS header of `logo.fit`. In addition any NDF extensions (apart from FITS) are turned into binary tables that follow the primary header and data unit.

```
ndf2fits horse logo.fit noprohis
```

This converts the NDF called horse to the new FITS file called `logo.fit`. The data type of the FITS primary data array matches that of the NDF's data array. The FITS extension in the NDF is merged into the FITS header of `logo.fit`. Should horse contain variance and quality arrays, these are written in IMAGE sub-files. Any history information in the NDF is not relayed to the FITS file.

```
ndf2fits "data/a*z" * comp=v noprofits bitpix=-32
```

This converts the NDFs with names beginning with `"a"` and ending in `"z"` in the directory called `data` into FITS files of the same name and with a file extension of `".fit"`. The variance array becomes the data array of each new FITS file. The data type of the FITS primary data array single-precision floating point. Any FITS extension in the NDF is ignored.

```
ndf2fits "abc,def" "jvp1.fit,jvp2.fit" comp=d bitpix="16,-64"
```

This converts the NDFs called abc and def into FITS files called `jvp1.fit` and `jvp2.fit` respectively. The data type of the FITS primary data array is signed integer words in `jvp1.fit`, and double-precision floating point in `jvp2.fit`. The FITS extension in each NDF is merged into the FITS header of the corresponding FITS file.

```
ndf2fits horse logo.fit d native encoding="fits-wcs"
```

This is the same as the first example except that the co-ordinate system information stored in the NDF's WCS component is written to the FITS file twice; once using the FITS-WCS headers, and once using a special set of 'native' keywords recognised by the AST library (see SUN/210). The native encoding provides a 'loss-free' means of transferring co-ordinate system information (*i.e.* no information is lost; other encodings may cause information to be lost). Only applications based on the AST library (such as FITS2NDF) are able to interpret native encodings.

```
ndf2fits u20040730_00675 merge container accept
```

This converts the UIST container file `u20040730_00675.sdf` to new FITS file `u20040730_00675.fit`, merging its .I1 and .HEADER structures into a single NDF before the conversion. The output file has only one header and data unit.

```
ndf2fits in=c20011204_00016 out=cgs4_16.fit container
```

This converts the CGS4 container file `c20011204_00016.sdf` to the multiple-extension FITS file `cgs4_16.fit`. The primary HDU has the global metadata from the .HEADER's FITS airlock. The four integrations in I1, I2, I3, and I4 components of the container file are converted to FITS IMAGE sub-files.

```
ndf2fits in=huge out=huge.fits comp=d bitpix=n
```

This converts the NDF called huge to the new FITS file called `huge.fits`. The data type of the FITS primary data array matches that of the NDF's scaled data array. The scale and offset coefficients used to form the FITS array are also taken from the NDF's scaled array.

```
ndf2fits in=huge out=huge.fits comp=d bitpix=-1
```

As the previous example, except that the data type of the FITS primary data array is that given by the BITPIX keyword in the FITS airlock of NDF huge and the scaling factors are determined.

Notes:

The rules for the conversion are as follows:

- The NDF main data array becomes the primary data array of the FITS file if it is in value of Parameter COMP, otherwise the first array defined by Parameter COMP will become the primary data array. A conversion from floating point to integer or to a shorter integer type will cause the output array to be scaled and offset, the values being recorded in keywords BSCALE and BZERO. There is an offset (keyword BZERO) applied to signed byte and unsigned word types to make them unsigned-byte and signed-word values respectively in the FITS array (this is because FITS does not support these data types).
- The FITS keyword BLANK records the bad values for integer output types. Bad values in floating-point output arrays are denoted by IEEE not-a-number values.
- The NDF's quality and variance arrays appear in individual FITS IMAGE sub-files immediately following the primary header and data unit, unless that component already appears as the primary data array. The quality array will always be written as an unsigned-byte array in the FITS file, regardless of the value of the Parameter BITPIX.
- Here are details of the processing of standard items from the NDF into the FITS header, listed by FITS keyword.
  - SIMPLE, EXTEND, PCOUNT, GCOUNT — all take their default values.
  - BITPIX, NAXIS, NAXISn — are derived directly from the NDF data array; however the BITPIX in the FITS airlock extension is transferred when Parameter BITPIX=-1.
  - CRVAL*n*, CDELT*n*, CRPIX*n*, CTYPE*n*, CUNIT*n* — are derived from the NDF WCS component if possible (see "World Co-ordinate Systems"). If this is not

possible, and if PROFITS is TRUE, then it copies the headers of a valid WCS specified in the NDF's FITS airlock. Should that attempt fail, the last resort tries the NDF AXIS component, if it exists. If its co-ordinates are non-linear, the AXIS co-ordinates may be exported in a -TAB sub-file subject to the value of Parameter USEAXIS.

- OBJECT, LABEL, BUNIT — the values held in the NDF's TITLE, LABEL, and UNITS components respectively are used if they are defined; otherwise any values found in the FITS extension are used (provided Parameter PROFITS is TRUE). For a variance array, BUNIT is assigned to (< *unit* >)**2, where < *unit* > is the DATA unit; the BUNIT header is absent for a quality array.
- DATE — is created automatically.
- ORIGIN — inherits any existing ORIGIN card in the NDF FITS extension, unless you supply a value through parameter ORIGIN other than the default "Starlink Software".
- EXTNAME — is the array-component name when the EXTNAME appears in the primary header or an IMAGE sub-file. In a binary-table derived from an NDF extension, EXTNAME is the path of the extension within the NDF, the path separator being the usual dot. The path includes the indices to elements of any array structures present; the indices are in a comma-separated list within parentheses.
  If the component is too long to fit within the header (68 characters), EXTNAME is set to @EXTNAMEF. The full path is then stored in keyword EXTNAMEF using the HEASARC Long-string CONTINUE convention (http://fits.gsfc.nasa.gov/registry/continue_keyword.html).
- EXTVER — is only set when EXTNAME (*q.v.*) cannot accommodate the component name, and it is assigned the HDU index to provide a unique identifier.
- EXTLEVEL — is the level in the hierarchical structure of the extension. Thus a top-level extension has value 1, sub-components of this extension have value 2 and so on.
- EXTTYPE — is the data type of the NDF extension used to create a binary table.
- EXTSHAPE — is the shape of the NDF extension used to create a binary table. It is a comma-separated list of the dimensions, and is 0 when the extension is not an array.
- HDUCLAS1, HDUCLAS*n* — "NDF" and the array-component name respectively.
- LBOUND*n* — is the pixel origin for the n^{th} dimension when any of the pixel origins is not equal to 1. (This is not a standard FITS keyword.)
- XTENSION, BSCALE, BZERO, BLANK and END — are not propagated from the NDF's FITS extension. XTENSION will be set for any sub-file. BSCALE and BZERO will be defined based on the chosen output data type in comparison with the NDF array's type, but cards with values 1.0 and 0.0 respectively are written to reserve places in the header section. These 'reservation' cards are for efficiency and they can always be deleted later. BLANK is set to the Starlink standard bad value corresponding to the type specified by BITPIX, but only for integer types and not for the quality array. It appears regardless of whether or not there are bad values actually present in the array; this is for the same efficiency reasons as before. The END card terminates the FITS header.

  - HISTORY — HISTORY headers are propagated from the FITS airlock when PROFITS is `TRUE`, and from the NDF history component when PROHIS is `TRUE`.
  - DATASUM and CHECKSUM — data-integrity keywords are written when Parameter CHECKSUM is `TRUE`, replacing any existing values. When Parameter CHECKSUM is `FALSE` and PROFITS is `TRUE` any existing values inherited from the FITS airlock are removed to prevent storage of invalid checksums relating to another data file.

  See also the sections "Provenance" and "World Co-ordinate Systems" for details of headers used to describe the PROVENANCE extension and WCS information respectively.

- Extension information may be transferred to the FITS file when PROEXTS is `TRUE`. The whole hierarchy of extensions is propagated in order. This includes substructures, and arrays of extensions and substructures. However, at present, any extension structure containing only substructures is not propagated itself (as zero-column tables are not permitted), although its substructures may be converted.

  Each extension or substructure creates a one-row binary table, where the columns of the table correspond to the primitive (non-structure) components. The name of each column is the component name. The column order is the same as the component order. The shapes of multi-dimensional arrays are recorded using the TDIM*n* keyword, where *n* is the column number. The HEASARCH convention for specifying the width of character arrays (keyword TFORM*n*='*r*A*w*', where *r* is the total number of characters in the column and *w* is the width of an element) is used. The EXTNAME, EXTTYPE, EXTSHAPE and EXTLEVEL keywords (see above) are written to the binary-table header.

There are additional rules if a multi-NDF container file is being converted (see Parameter CONTAINER). This excludes the case where there are but two NDFs—one data and the other just headers—that have already been merged (see Parameter MERGE):

- For multiple NDFs a header-only HDU may be created followed by an IMAGE sub-file containing the data array (or whichever other array is first specified by COMP).
- BITPIX for the header HDU is set to an arbitrary 8.
- Additional keywords are written for each IMAGE sub-file.
  - HDSNAME — is the NDF name for a component NDF in a multi-NDF container file, for example `I2`.
  - HDSTYPE — is set to `NDF` for a component NDF in a multi-NDF container file.

World Co-ordinate Systems :

Any co-ordinate system information stored in the WCS component of the NDF is written to the FITS header using one of the following encoding systems (the encodings used are determined by parameters ENCODING and NATIVE):

- `"FITS-IRAF"` — This uses keywords CRVAL*i*, CRPIX*i*, and 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 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.atnf.csiro.au/people/mcalabre/WCS/) 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 a 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 described 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 (amongst others), 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 which allows the use of a wider range of celestial as 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 the GILDAS manual.
- "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 applications. It allows more complicated WCS information to be stored and retrieved than any of the other encodings.

Values for FITS keywords generated by the above encodings will always be used in preference to any corresponding keywords found in the FITS extension (even if PROFITS is TRUE). If this is not what is required, the WCS component of the NDF should be erased using the KAPPA command ERASE before running NDF2FITS. Note, if PROFITS is TRUE, then any WCS-related keywords in the FITS extension which are not replaced by keywords derived from the WCS component may appear in the output FITS file. If this causes a problem, then PROFITS should be set to FALSE or the offending keywords removed using KAPPA FITSEDIT for example.

Provenance :

The following PROVENANCE headers are written if parameter PROVENANCE is set to "Generic".

- PRVP*n* — is the path of the *n*th NDF.
- PRVIn — is a comma-separated list of the identifiers of the direct parents for the *n*th ancestor.
- PRVDn — is the creation date in ISO order of the *n*th ancestor.
- PRVCn — is the software used to create the *n*th ancestor

- PRVMn — lists the contents of the MORE structure of the *n*th parent.

All have value `<unknown>` if the information could not be found, except for the PRVM*n* header, which is omitted if there is no MORE information to record. The index *n* used in each keyword's name is the provenance identifier for the NDF, and starts at 0 for the NDF being converted to FITS.

The following PROVENANCE headers are written if parameter PROVENANCE is set to `"CADC"`.

- PRVCNT — is the number of immediate parents.
- PRV*m* — is name of the *m*th immediate parent.
- OBSCNT — is the number of root ancestor OBS*m* headers.
- OBS*m* — is *m*th root ancestor identifier from its MORE.OBSIDSS component.
- FILEID — is the name of the output FITS file.
- PRODUCT is modified or added to each sub-file's header to be the primary header's value of PRODUCT with a `_<extnam>` suffix, where `<extnam>` is the extension name in lowercase.

When PROFITS is `TRUE` any existing provenance keywords in the FITS airlock are not copied to the FITS file.

Quality Masking :

NDF automatic quality masking is a facility whereby any bad quality information (flagged by the bad-bits mask) present can be incorporated in the data or variance as bad values. NDF2FITS uses this facility in exported data variance information provided the quality array is not transferred. Thus if a QUALITY component is present in the input NDF, the data and any variance arrays will not be masked whenever Parameter COMP's value is `'A'` or contains `'Q'`.

Special Formats :

In the general case, NDF extensions (excluding the FITS extension) may be converted to one-row binary tables in the FITS file when Parameter PROEXTS is `TRUE`. This preserves the information, but it may not be accessible to the recipient's FITS reader. Therefore, in some cases it is desirable to understand the meanings of certain NDF extensions, and create standard FITS products for compatibility.

At present only one product is supported, but others may be added as required.

- AAO 2dF

  Standard processing is used except for the 2dF FIBRES extension and its constituent structures. The NDF may be restored from the created FITS file using FITS2NDF. The FIBRES extension converts to the second binary table in the FITS file (the NDF_CLASS extension appears in the first).

  To propagate the OBJECT substructure, NDF2FITS creates a binary table of constant width (224 bytes) with one row per fibre. The total number of rows is obtained from component NUM_FIBRES. If a possible OBJECT component is missing from the NDF, a null column is written for that component. The columns inherit the data types of

the OBJECT structure's components. Column meanings and units are assigned based upon information in the reference given below.

The FIELD structure components are converted into additional keywords of the same name in the binary-table header, with the exception that components with names longer than eight characters have abbreviated keywords: UNALLOC*xxx* becomes UNAL-*xxx* (*xxx*=OBJ, GUI, or SKY), CONFIGMJD becomes CONFMJD, and *x*SWITCHOFF becomes *x*SWTCHOF (*x*=X or Y). If any FIELD component is missing it is ignored.

Keywords for the extension level, name, and type appear in the binary-table header.

- JCMT SMURF

  Standard processing is used except for the SMURF-type extension. This contains NDFs such as EXP_TIME and TSYS. Each such NDF is treated like the main NDF except that it is assumed that these extension NDFs have no extensions of their own. FITS airlock information and HISTORY are inherited from the parent NDF. Also the extension keywords are written: EXTNAME gives the path to the NDF, EXTLEVEL records the extension hierarchy level, and EXTTYPE is set to `"NDF"`. Any non-NDF components of the SMURF extension are written to a binary table in the normal fashion.

References :

Bailey J.A. 1997, 2dF Software Report 14, version 0.5.

NASA Office of Standards and Technology, 1994, *A User's Guide for the Flexible Image Transport System (FITS)*, version 3.1.

NASA Office of Standards and Technology, 1995, *Definition of the Flexible Image Transport System (FITS)*, version 1.1.

Related Applications :

CONVERT: FITS2NDF; KAPPA: FITSDIN, FITSIN.

Implementation Status:

- All NDF data types are supported.

NDF2GASP
Converts a two-dimensional NDF into a GASP image

Description:

This application converts a two-dimensional NDF into the GAlaxy Surface Photometry (GASP) package's format. See the "Notes" for the details of the conversion.

Usage:

```
ndf2gasp in out [fillbad]
```

Parameters:

IN = NDF (Read)

The input NDF data structure. The suggested default is the current NDF if one exists, otherwise it is the current value.

FILLBAD = _INTEGER (Read)

The value used to replace bad pixels in the NDF's data array before it is copied to the GASP file. The value must be in the range of signed words (−32768 to 32767). A null value (`!`) means no replacements are to be made. This parameter is ignored if there are no bad values. `[!]`

OUT = FILENAME (Write)

The name of the output GASP image. Two files are produced with the same name but different file extensions. The `".dat"` file contains the data array, and `".hdr"` is the associated header file that specifies the dimensions of the image. The suggested default is the current value.

Examples:

```
ndf2gasp abell1367 a1367
```

Converts an NDF called abell1367 into the GASP image comprising the pixel file `a1367.dat` and the header file `a1367.hdr`. If there are any bad values present they are copied verbatim to the GASP image.

```
ndf2gasp ngc253 ngc253 fillbad=-1
```

Converts the NDF called ngc253 to a GASP image comprising the pixel file `ngc253.dat` and the header file `ngc253.hdr`. Any bad values in the data array are replaced by minus one.

Notes:

The rules for the conversion are as follows:

- The NDF data array is copied to the `".dat"` file.
- The dimensions of the NDF data array is written to the `".hdr"` header file.
- All other NDF components are ignored.

Related Applications :

CONVERT: GASP2NDF.

References :

GASP documentation (MUD/66).

Implementation Status:

- The GASP image produced has by definition type SIGNED WORD. There is type conversion of the data array to this type.
- Bad values may arise due to type conversion. These too are substituted by the (non-null) value of FILLBAD.
- For an NDF with an odd number of columns, the last column from the GASP image is trimmed.

NDF2GIF
Converts an NDF into a GIF file.

Description:

This Bourne-shell script converts an NDF into a 256 grey-level Graphics Interchange Format (GIF) file. One- or two-dimensional images can be handled and various methods of scaling the data are provided. The script uses the CONVERT utility NDF2TIFF to produce a TIFF file and then various NETPBM utilities to convert the TIFF file into a GIF file.

If the 'high' scaling value is less than the 'low' value, the output image will be a negative. Bad values are set to 0 for positives and 255 for negatives.

Error messages are converted into Starlink style (preceded by `!`).

Usage:

```
                                 { low=?  high=?
ndf2gif in [out] [scale]         { percentiles=?  [nbins=?]
                                 { sigmas=?
```

Parameters:

IN = NDF (Read)
The name of the input NDF (the `.sdf` extension is not required).

OUT = FILENAME (Write)
The name of the GIF file to be generated (a `.gif` name extension is added if it is omitted). If OUT is omitted, the value of the IN parameter is used. Any existing file with the same name will be overwritten.

The following parameters are actually parameters of the ADAM task NDF2TIFF. Their values on the NDF2GIF command line are just passed through to NDF2TIFF, which may prompt for other required values. The output parameters SCAHIGH and SCALOW will be found in NDF2TIFF's parameter file.

HIGH = _DOUBLE (Read) Only required if SCALE is `"Scale"`. The array value that scales to 255 in the TIFF file. All larger array values are set to 255 when HIGH is greater than LOW, otherwise all array values less than HIGH are set to 255. 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. The highest data value is suggested in prompts.

LOW = _DOUBLE (Read) Only required if SCALE is `"Scale"`. The array value that scales to 0 in the TIFF file. All smaller array values are also set to 0 when LOW is less than HIGH, otherwise all array values greater than LOW are set to 0. 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. The lowest data value is suggested in prompts.

MSG_FILTER = LITERAL (Read) The output message filtering level, `QUIET`, `NORMAL` or `VERBOSE`. If set to verbose, the scaling limits used will be displayed. `[NORMAL]`

NUMBIN = _INTEGER (Read) Only used if SCALE is "Percentiles". The number of histogram bins used to compute percentiles for scaling. (Percentiles mode) [2048]

PERCENTILES(2) = _REAL (Read) Only required if SCALE is "Percentiles". The percentiles that define the scaling limits. For example, [25,75] would scale between the quartile values.

SCALE = LITERAL (Read) The type of scaling to be applied to the array. The options, which may be abbreviated to an unambiguous string and are case-insensitive, are described below:

- "Range" — The image is scaled between the minimum and maximum data values. (This is the default.)
- "Faint" — The image is scaled from the mean minus one standard deviation to the mean plus seven standard deviations.
- "Percentiles" — The image is scaled between the values corresponding to two percentiles.
- "Scale" — You define the upper and lower limits between which the image is to be scaled. The application suggests the maximum and the minimum values minimum values when prompting.
- "Sigmas" — The image is scaled between two standard-deviation limits.

["Range"]

SIGMAS(2) = _REAL (Read) Only required if SCALE is "Sigmas". 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.

Results Parameters:

SCAHIGH = _DOUBLE (Write)
The array value scaled to the maximum colour index for display.

SCALOW = _DOUBLE (Write)
The array value scaled to the minimum colour index for display.

Examples:

```
ndf2gif old new
```

This converts the NDF called old (in file old.sdf) into a GIF file new.gif.

```
ndf2gif horse scale=pe
```

This converts the NDF called horse (in file horse.sdf) into a GIF file horse.gif using percentile scaling. The user will be prompted for the percentiles to use.

Notes:

The following points should be remembered:

- This initial version of the script generates only 256 grey levels and does not use the image colour lookup table so absolute data values may be lost.

- The NETPBM utilities `tifftopnm` and `ppmtogif` must be available on your PATH.

Related Applications :

CONVERT: GIF2NDF, NDF2TIFF.

NDF2IRAF
Converts an NDF to an IRAF image

Description:

This application converts an NDF to an IRAF image. See the "Notes" for details of the conversion.

Usage:

```
ndf2iraf in out [fillbad]
```

Parameters:

IN = NDF (Read)

The input NDF data structure. The suggested default is the current NDF if one exists, otherwise it is the current value.

FILLBAD = _REAL (Read)

The value used to replace bad pixels in the NDF's data array before it is copied to the IRAF file. A null value (!) means no replacements are to be made. This parameter is ignored if there are no bad values. [0]

OUT = LITERAL (Write)

The name of the output IRAF image. Two files are produced with the same name but different extensions. The ".pix" file contains the data array, and ".imh" is the associated header file that may contain a copy of the NDF's FITS extension. The suggested default is the current value.

PROFITS = _LOGICAL (Read)

If TRUE, the contents of the FITS extension of the NDF are merged with the header information derived from the standard NDF components. See the "Notes" for details of the merger. [TRUE]

PROHIS = _LOGICAL (Read)

If TRUE, any NDF history records are written to the primary FITS header as HISTORY cards. These follow the mandatory headers and any merged FITS-extension headers (see parameter PROFITS). [TRUE]

Examples:

```
ndf2iraf abell119 a119
```

Converts an NDF called abell119 into the IRAF image comprising the pixel file a119.pix and the header file a119.imh. If there are any bad values present they are copied verbatim to the IRAF image.

```
ndf2iraf abell119 a119 noprohis
```

As the previous example, except that NDF HISTORY records are not transferred to the headers in a119.imh.

```
ndf2iraf qsospe qsospe fillbad=0
```

Converts the NDF called qsospe to an IRAF image comprising the pixel file `qsospe.imh` and the header file `qsospe.pix`. Any bad values in the data array are replaced by zero.

```
ndf2iraf qsospe qsospe fillbad=0 profits=f
```

As the previous example, except that any FITS airlock information in the NDF are not transferred to the headers in `qsospe.imh`.

Notes:

The rules for the conversion are as follows:

- The NDF data array is copied to the `".pix"` file. Ancillary information listed below is written to the `".imh"` header file in FITS-like headers.
- The IRAF "Mini World Co-ordinate System" (MWCS) is used to record axis information whenever one of the following criteria is satisfied:
  (1) the dataset has some linear axes (system=world);
  (2) the dataset is one-dimensional with a non-linear axis, or is two-dimensional with the first axis non-linear and the second being some aperture number or index (system=multispec);
  (3) the dataset has a linear spectral/dispersion axis along the first dimension and all other dimensions are pixel indices (system=equispec).
- The NDF title, label, units are written to the header keywords TITLE, OBJECT, and BUNIT respectively if they are defined. Otherwise any values for these keywords found in the FITS extension are used (provided Parameter PROFITS is `TRUE`). There is a limit of twenty characters for each.
- The NDF pixel origins are stored in keywords LBOUND*n* for the nth dimension when any of the pixel origins is not equal to 1.
- Keywords HDUCLAS1, HDUCLAS*n* are set to `"NDF"` and the array-component name respectively.
- The BLANK keyword is set to the Starlink standard bad value, but only for the _WORD data type and not for a quality array. It appears regardless of whether or not there are bad values actually present in the array.
- HISTORY headers are propagated from the FITS extension when PROFITS is `TRUE`, and from the NDF HISTORY component when PROHIS is `TRUE`.
- If there is a FITS extension in the NDF, then the elements up to the first END keyword of this are added to the 'user area' of the IRAF header file, when PROFITS=`TRUE`. However, certain keywords are excluded: SIMPLE, NAXIS, NAXIS*n*, BITPIX, EXTEND, PCOUNT, GCOUNT, BSCALE, BZERO, END, and any already created from standard components of the NDF listed above.
- A HISTORY record is added to the IRAF header file indicating that it originated in the named NDF and was converted by NDF2IRAF.
- All other NDF components are not propagated.

Related Applications :

CONVERT: IRAF2NDF.

Pitfalls :

The IMFORT routines refuse to overwrite an IRAF image if an image with the same name exists. The application then aborts.

Some of the routines required for accessing the IRAF header image are written in SPP. Macros are used to find the start of the header line section, this constitutes an 'Interface violation' as these macros are not part of the IMFORT interface specification. It is possible that these may be changed in the future, so beware.

References :

IRAF User Handbook Volume 1A: `A User's Guide to FORTRAN Programming in IRAF, the IMFORT Interface`, by Doug Tody.

Implementation Status:

- Only handles one-, two-, and three-dimensional NDFs.
- Of the NDF's array components only the data array may be copied.
- The IRAF image produced has type SIGNED WORD or REAL dependent of the type of the NDF's data array. (The IRAF IMFORT FORTRAN subroutine library only supports these data types.) For _BYTE, _UBYTE, and _WORD data arrays the IRAF image will have type SIGNED WORD; for all other data types of the NDF data array a REAL IRAF image is made. The pixel type of the image can be changed from within IRAF using the 'chpixtype' task in the 'images' package.
- Bad values may arise due to type conversion. These too are substituted by the (non-null) value of FILLBAD.
- See "Release Notes" for IRAF Version compatibility.

NDF2PGM
Converts an NDF to a PBMPLUS-style PGM-format file.

Description:

This application converts an NDF to a PBMPLUS PGM-format file. The programme first finds the brightest and darkest pixel values in the image. It then uses these to determine suitable scaling factors to convert the image into an 8-bit representation. These are then output to a simple greyscale PBMPLUS PGM file.

Usage:

```
ndf2pgm in out
```

Parameters:

IN = NDF (Read)

The name of the input NDF data structure (without the `.sdf` extension). The suggested default is the current NDF if one exists, otherwise it is the current value.

OUT = _CHAR (Read)

The name of the PGM file be generated. The `.pgm` name extension is added to any output filename that does not contain it.

Examples:

```
ndf2pgm old new
```

This converts the NDF called old (in file `old.sdf`) to the PGM file `new.pgm`.

```
ndf2pgm in=spectre out=spectre.pgm
```

This converts the NDF called spectre (in file `spectre.sdf`) to the PGM file `spectre.pgm`.

Notes:

This programme was written for diagnostic purposes and is included just in case someone finds it useful.

Implementation Status:

Bad values in the data array are replaced with zero in the output PGM file.

NDF2TIFF
Converts an NDF to an 8-bit TIFF-6.0-format file.

Description:

This application converts an NDF to a Tag Image File Format (TIFF). One- or two-dimensional arrays can be handled and various methods of scaling the data are provided.

The routine first finds the brightest and darkest pixel values required by the particular scaling method in use. It then uses these to determine suitable scaling factors and converts the image into an 8-bit representation which is then output to a simple greyscale TIFF-6.0 file.

If the 'high' scaling value is less than the 'low' value, the output image will be a negative. Bad values are set to 0 for positives and 255 for negatives.

Usage:

```
ndf2tiff in out [scale]   { low =?high =?
                          { percentiles =?[nbins =?]
                          { sigmas =?
```

Parameters:

HIGH = _DOUBLE (Read)

The array value that scales to 255 in the TIFF file. It is only required if SCALE is "Scale". All larger array values are set to 255 when HIGH is greater than LOW, otherwise all array values less than HIGH are set to 255. 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. The highest data value is suggested in prompts.

IN = NDF (Read)

The name of the input NDF data structure (without the .sdf extension). The suggested default is the current NDF if one exists, otherwise it is the current value.

LOW = _DOUBLE (Read)

The array value that scales to 0 in the TIFF file. It is only required if SCALE is "Scale". All smaller array values are also set to 0 when LOW is less than HIGH, otherwise all array values greater than LOW are set to 0. 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. The lowest data value is suggested in prompts.

MSG_FILTER = LITERAL (Read)

The output message filtering level, QUIET, NORMAL or VERBOSE. If set to verbose, the scaling limits used will be displayed. [NORMAL]

NUMBIN = _INTEGER (Read)

The number of histogram bins used to compute percentiles for scaling. It is only used if SCALE is "Percentiles". [2048]

OUT = _CHAR (Read)
The name of the TIFF file to be generated. A `.tif` name extension is added to any output filename that does not contain it. Any existing file with the same name will be overwritten.

PERCENTILES(2) = _REAL (Read)
The percentiles that define the scaling limits. For example, `[25,75]` would scale between the quartile values. It is only required if SCALE is `"Percentiles"`.

SCALE = LITERAL (Read)
The type of scaling to be applied to the array. The options, which may be abbreviated to an unambiguous string and are case-insensitive, are described below.

- `"Range"` — The image is scaled between the minimum and maximum data values. (This is the default.)
- `"Faint"` — The image is scaled from the mean minus one standard deviation to the mean plus seven standard deviations.
- `"Percentiles"` — The image is scaled between the values corresponding to two percentiles.
- `"Scale"` — You define the upper and lower limits between which the image is to be scaled. The application suggests the maximum and the minimum values minimum values when prompting.
- `"Sigmas"` — The image is scaled between two standard-deviation limits.

`["Range"]`

SIGMAS(2) = _REAL (Read)
Only required if SCALE is `"Sigmas"`. 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.

Results Parameters:

SCAHIGH = _DOUBLE (Write)
The array value scaled to the maximum colour index for display.

SCALOW = _DOUBLE (Write)
The array value scaled to the minimum colour index for display.

Examples:

```
ndf2tiff old new
```

This converts the NDF called old (in file `old.sdf`) to the TIFF file called `new.tif`.

```
ndf2tiff horse horse pe
```

This converts the NDF called horse (in file `horse.sdf`) into a TIFF file `horse.tif` using percentile scaling. The user will be prompted for the percentiles to use.

Notes:

This application generates only 256 grey levels and does not use any image colour lookup table so absolute data values may be lost.

No compression is applied.

Related Applications :

CONVERT: TIFF2NDF.

NDF2UNF
Converts an NDF to a sequential unformatted file

Description:

This application converts an NDF to a sequential unformatted Fortran file. Only one of the array components may be copied to the output file. Preceding the data there is an optional header consisting of either the FITS extension with the values of certain keywords replaced by information derived from the NDF, or a minimal FITS header also derived from the NDF.

Usage:

```
ndf2unf in out [comp] [noperec]
```

Parameters:

COMP = LITERAL (Read)

The NDF component to be copied. It may be `"Data"`, `"Quality"` or `"Variance"`. `["Data"]`

FITS = _LOGICAL (Read)

If `TRUE`, any FITS extension is written to start of the output file, unless there is no extension whereupon a minimal FITS header is written to the unformatted file. `[FALSE]`

IN = NDF (Read)

Input NDF data structure. The suggested default is the current NDF if one exists, otherwise it is the current value.

NOPEREC = _INTEGER (Read)

The number of data values per record of the output file. It must be positive. The suggested default is the current value. [The first dimension of the NDF]

OUT = FILENAME (Write)

Name of the output sequential unformatted file. The file will but always fixed-length records when there is no header.

Examples:

```
ndf2unf cluster cluster.dat
```

This copies the data array of the NDF called cluster to an unformatted file called `cluster.dat`. The number of data values per record is equal to the size of the first dimension of the NDF.

```
ndf2unf cluster cluster.dat v
```

This copies the variance of the NDF called cluster to an unformatted file called `cluster.dat`. The number of variance values per record is equal to the size of the first dimension of the NDF.

```
ndf2unf cluster cluster.dat noperec=12
```

This copies the data array of the NDF called cluster to an unformatted file called `cluster.dat`. There are twelve data values per record in `cluster.dat`.

```
ndf2unf out=ndf234.dat fits in=@234
```

This copies the data array of the NDF called 234 to an unformatted file called `ndf234.dat`. The number of data values per record is equal to the size of the first dimension of the NDF. If there is a FITS extension, it is copied to `ndf234.dat` with substitution of certain keywords, otherwise a minimal FITS header is produced.

Notes:

The details of the conversion are as follows:

- the NDF array as selected by COMP is written to the unformatted file in records following an optional header.
- HISTORY is not propagated.
- ORIGIN information is lost.
- When a header is to be made, it is composed of FITS-like card images as follows:

  - The number of dimensions of the data array is written to the keyword NAXIS, and the actual dimensions to NAXIS1, NAXIS2 *etc.* as appropriate.
  - If the NDF contains any linear axis structures the information necessary to generate these structures is written to the FITS-like headers. For example, if a linear AXIS(1) structure exists in the input NDF the value of the first data point is stored with the keyword CRVAL1, and the incremental value between successive axis data is stored in keyword CDELT1. By definition the reference pixel is 1.0 and is stored in keyword CRPIX1. If there is an axis label it is written to keyword CTYPE1, and axis unit is written to CUNIT1. (Similarly for AXIS(2) structures *etc.*) FITS does not have a standard method of storing axis widths and variances, so these NDF components will not be propagated to the header. Non-linear axis data arrays cannot be represented by CRVAL*n* and CDELT*n*, and must be ignored.
  - If the input NDF contains TITLE, LABEL or UNITS components these are stored with the keywords TITLE, LABEL or BUNIT respectively.
  - If the input NDF contains a FITS extension, the FITS items may be written to the FITS-like header, with the following exceptions:
    * BITPIX is derived from the type of the NDF data array, and so it is not copied from the NDF FITS extension.
    * NAXIS, and NAXIS*n* are derived from the dimensions of the NDF data array as described above, so these items are not copied from the NDF FITS extension.
    * The TITLE, LABEL, and BUNIT descriptors are only copied if no TITLE, LABEL, and UNITS NDF components respectively have already been copied into these headers.

        * The CDELT*n*, CRVAL*n*, CTYPE*n*, CUNIT*n*, and CRTYPE*n* descriptors in the FITS extension are only copied if the input NDF contained no linear axis structures.
        * The standard order of the FITS keywords is preserved, thus BITPIX, NAXIS and NAXIS*n* appear immediately after the first card image, which should be SIMPLE.
        * BSCALE and BZERO in a FITS extension are copied when BITPIX is positive, *i.e.* the array is not floating-point.
      - An extra header record with keyword UNSIGNED and logical value T is added when the array data type is one of the HDS unsigned integer types. This is done because standard FITS does not support unsigned integers, and allows (in conjunction with BITPIX) applications reading the unformatted file to determine the data type of the array.
      - The last header record card will be the standard FITS END.
  - Other extensions are not propagated.

Related Applications :

CONVERT: UNF2NDF.

Implementation Status:

  - The value of bad pixels is not written to a FITS-like header record with keyword BLANK.

SPECX2NDF
Converts a SPECX map into a simple data cube, or SPECX data files to individual spectra.

Description:

This application converts a SPECX map file into a simple data cube formatted as a standard NDF. It works on map files in Version 4.2 or later of the SPECX format. It can optionally write a schematic of the map grid to a text file.

In addition, it will also convert an HDS container file containing an array of one-dimensional NDFs holding SPECX spectra into a similar container file holding individual, scalar NDFs each holding a single spectrum from the supplied array.

In both cases, WCS components are added to the output NDFs describing the spectral and spatial axes.

A VARIANCE component is added to the output NDF that has a constant value derived from the Tsys value, integration time, and channel spacing in the input.

Usage:

```
specx2ndf in out [gridfile] [telescope] [latitude] [longitude]
```

Parameters:

AXIS = _LOGICAL (Read)
AXIS structures will be added to the output NDF if and only if AXIS is set TRUE. [FALSE]

GRIDFILE = LITERAL (Read)
The name of a text file to which a schematic of the SPECX map will be written. This schematic shows those positions in the map grid where spectra were observed. To indicate that a file containing the schematic is not to be written reply with an exclamation mark ("!"). See Section "Schematic of the map grid" (below) for further details. [!]

IN = NDF (Read)
The name of the input SPECX map, or container file. The file file extension (.sdf) should not be included since it is appended automatically by the application.

LATITUDE = LITERAL (Read)
The geodetic (geographic) latitude of the telescope where the observation was made. The value should be specified in sexagesimal degrees, with a colon (":") to separate the degrees, minutes and seconds, and no embedded spaces. Values in the northern hemisphere are positive. The default corresponds to the latitude of the JCMT. ["19:49:33"]

LONGITUDE = LITERAL (Read)
The geodetic (geographic) longitude of the telescope where the observation was made. The value should be specified in sexagesimal degrees, with a colon (":") to separate the degrees, minutes and seconds, and no embedded spaces. Following the

usual geographic convention longitudes west of Greenwich are positive. The default corresponds to the longitude of the JCMT. `["155:28:47"]`

OUT = NDF (Write)
The name of the output NDF containing the data cube or spectra written by the application. The file extension (`.sdf`) should not be included since it is appended automatically by the application.

SYSTEM = LITERAL (Read)
Celestial co-ordinate system for output cube. SPECX files do not record the co-ordinate system for any offsets. The recognised options are as follows.

- `"AZ"` — azimuth and elevation
- `"GA"` — galactic
- `"RB"` — B1950
- `"RD"` — equatorial of date
- `"RJ"` — J2000

SYSTEM needs to be used to set manually the correct co-ordinates for a map file. `["RJ"]`

TELESCOPE = LITERAL (Read)
The name of the telescope where the observation was made. This parameter is used to look up the geodetic (geographical) latitude and longitude of the telescope. See the documentation of subroutine SLA_OBS in SUN/67 for a list of permitted values. Alternatively, if you wish to explicitly enter the latitude and longitude enter `"COORDS"`. The values are not case sensitive. `["JCMT"]`

Examples:

```
specx2ndf specx_map specx_cube
```

This example generates an NDF data cube called specx_cube (in file `specx_cube.sdf`) from the NDF SPECX map called specx_map (in file `specx_map.sdf`). A text file containing a schematic of the map grid will not be produced.

```
specx2ndf specx_map specx_cube gridfile=map.grid
```

This example generates an NDF data cube called specx_cube (in file `specx_cube.sdf`) from the NDF SPECX map called specx_map (in file `specx_map.sdf`). A text file containing a schematic of the map grid will be written to file `map.grid`.

Input and output map formats :

SPECX map files are written by the SPECX package (see SUN/17) for reducing spectra observed with heterodyne receivers operating in the mm and sub-mm wavelength range of the electromagnetic spectrum. SPECX is usually used to process observations obtained with the James Clerk Maxwell Telescope (JCMT) in Hawaii.

A SPECX map file comprises a regular 'rectangular' two-dimensional grid of map positions on the sky, with spectra observed at the grid points. However, a spectrum is not necessarily available at every grid position; at some positions a spectrum is not observed in order to save observing time. For example, for a grid centred on a typical, roughly circular,

object spectra may be omitted for the positions at the corners of the grid. SPECX map files are standard Starlink NDF HDS structures. The principal array of the NDF is a two-dimensional array of the grid positions. The value of each element is either a pointer to the spectrum observed there (in practice the number of the spectrum in the array where they are stored) or a value indicating that a spectrum was not observed at this point. In effect the SPECX map structure is an implementation of a sparse array.

SPECX2NDF expands a SPECX map file into a simple three-dimensional data cube, again formatted as a standard NDF, in which the first and second pixel axes corresponds to the spatial axes and the third axes correspond to the spectral axis. The advantage of this approach is that the converted file can be examined with standard applications, such as those in KAPPA (see SUN/95) and easily imported into visualisation packages, such as Data Explorer (DX, see SUN/203 and SC/2). When the output data cube is created the columns corresponding to the positions on the sky grid where spectra were not observed are filled with 'bad' values (sometimes called 'magic' or 'null' values), to indicate that valid data are not available at these positions. The standard Starlink bad value is used. Because of the presence of these bad values the expanded cube is usually larger than the original map file.

The created NDF cube has a WCS component in which Axes 1 and 2 are RA and DEC, and Axis 3 is frequency in units of GHz. The nature of these axes can be changed if necessary by subsequent use of the WCSATTRIB application within the KAPPA package. For compatibility with older applications, AXIS structures may also be added to the output cube (see Parameter AXIS). Axes 1 and 2 are offsets from the central position of the map, with units of seconds of arc, and Axis 3 is frequency offset in GHz relative to the central frequency. The pixel origin is placed at the source position on Axes 1 and 2, and the central frequency on Axis 3.

SPECX2NDF reads map files in Version 4.2 or later of the SPECX data format. If it is given a map file in an earlier version of the data format it will terminate with an error message. Note, however, that SPECX itself can read map files in earlier versions of the SPECX format and convert them to Version 4.2.

Schematic of the map grid :

SPECX2NDF has an optional facility to write a crude schematic of the grid of points observed on the sky to an ASCII text file suitable for printing or viewing on a terminal screen. This schematic can be useful in interpreting displays of the data cube. It shows the positions on the grid where spectra were observed. Each spectrum is numbered within the SPECX map structure and the first nine are shown using the digits one to nine. The remaining spectra are shown using an asterisk ("*"). You specify the name of the file to which the schematic is written. Figure 1 shows an example of a schematic.

Auxiliary information :

SPECX2NDF copies all the auxiliary information present in the original map file to the output data cube. However, the arrays holding the original spectra are not copied in order to save disk space.

Input and output spectra formats :

In addition to converting SPECX map files, this application can also convert HDS files which hold an array of one-dimensional NDF structures, each being a single spectrum

extracted by SPECX. Since arrays of NDFs are not easily accessed, this application extracts each NDF from the array and creates a new scalar NDF holding the same data within the output container file. The name of the new NDF is `SPECTRUM`*n* where *n* is its index within the original array of NDFs. Each new scalar NDF is actually three-dimensional and has the format described above for an output cube (*i.e.* Axes 1 and 2 are RA and DEC, and Axis 3 is frequency). However, pixel Axes 1 and 2 span only a single pixel (the size of this single spatial pixel is assumed to be half the size of the resolution of the JCMT at the central frequency). Inclusion of three-dimensional WCS information allows the individual spectra to be aligned on the sky (for instance using the KAPPA WCSALIGN task).

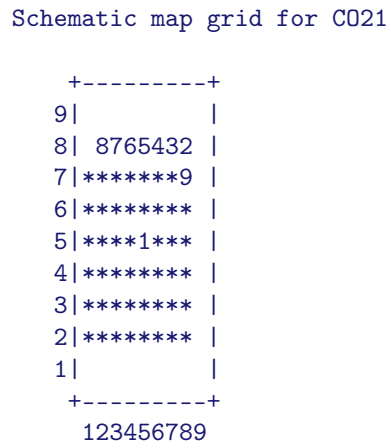

Figure 1: Example of a map schematic

TIFF2NDF
Converts a TIFF file into an NDF.

Description:

This Bourne-shell script converts a 256 grey-level or black-and-white dithered Tag Image File Format (TIFF) into an unsigned-byte NDF file. It handles one- or two-dimensional images. The script uses various NETPBM utilities to produce a FITS file, flipped top to bottom, and then FITS2NDF to produce the final NDF. Error messages are converted into Starlink style (preceded by `!`).

Usage:

```
tiff2ndf in [out]
```

Parameters:

IN = FILENAME (Read)
The name of the TIFF file to be converted. (A `.tif` name extension is assumed if omitted.)

OUT = NDF (Write)
The name of the NDF to be generated (without the `.sdf` extension). If this is omitted, the value of the IN parameter is used.

Examples:

```
tiff2ndf old new
```

This converts the TIFF file `old.tif` into an NDF called new (in file `new.sdf`).

```
tiff2ndf horse
```

This converts the TIFF file `horse.tif` into an NDF called horse (in file `horse.sdf`).

Notes:

The following points should be remembered:

- This initial version of the script handles only greyscale or b/w dithered images. You are responsible for conversion of your images to this format prior to use, including the conversion of RGB values to brightness values.
- Input image file names must have the extension `.tif`.
- The NETPBM utilities `tifftopnm`, `ppmtopgm`, `pnmflip` and `pnmtofits` must be available on your PATH.

Related Applications :

CONVERT: NDF2TIFF.

UNF2NDF
Converts a sequential unformatted file to an NDF

Description:

This application converts a sequential unformatted Fortran file to an NDF. Only one of the array components may be created from the input file. Preceding the input data there may be an optional header. This header may be skipped, or may consist of a simple FITS header. In the former case the shape of the NDF has be to be supplied.

Usage:

```
unf2ndf in out [comp] noperec [skip] shape [type]
```

Parameters:

COMP = LITERAL (Read)

The NDF component to be copied. It may be "Data", "Quality" or "Variance". To create a variance or quality array the NDF must already exist. ["Data"]

FITS = _LOGICAL (Read)

If TRUE, the initial records of the unformatted file are interpreted as a FITS header (with one card image per record) from which the shape, data type, and axis centres are derived. The last record of the FITS-like header must be terminated by an END keyword; subsequent records in the input file are treated as an array component given by COMP. [FALSE]

IN = FILENAME (Read)

Name of the input sequential unformatted Fortran file. The file will normally have variable-length records when there is a header, but always fixed-length records when there is no header.

NOPEREC = _INTEGER (Read)

The number of data values per record of the input file. It must be positive on UNIX systems. The suggested default is the size of the first dimension of the array if there is no current value. A null (!) value for NOPEREC causes the size of first dimension to be used.

OUT = NDF (Read and Write)

Output NDF data structure. When COMP is not "Data" the NDF is modified rather than a new NDF created. It becomes the new current NDF.

SHAPE = _INTEGER (Read)

The shape of the NDF to be created. For example, [40,30,20] would create 40 columns by 30 lines by 10 bands. It is only accessed when FITS is FALSE.

SKIP = INTEGER (Read)

The number of header records to be skipped at the start of the input file before finding the data array or FITS-like header. [0]

TYPE = LITERAL (Read)

The data type of the input file and output NDF. It must be one of the following HDS types: "_BYTE", "_WORD", "_REAL", "_INTEGER", "_INT64", "_DOUBLE", "_UBYTE",

"_UWORD" corresponding to signed byte, signed word, real, integer, double precision, unsigned byte, and unsigned word. See SUN/92 for further details. An unambiguous abbreviation may be given. TYPE is ignored when COMP = "Quality" since the QUALITY component must comprise unsigned bytes (equivalent to TYPE = "_UBYTE") to be a valid NDF. The suggested default is the current value. TYPE is also only accessed when FITS is FALSE. ["_REAL"]

Examples:

```
unf2ndf ngc253.dat ngc253 shape=[100,60] noperec=8
```

This copies a data array from the unformatted file ngc253.dat to the NDF called ngc253. The input file does not contain a header section. The NDF is two-dimensional: 100 elements in x by 60 in y. Its data array has type _REAL. The data records each have 8 values.

```
unf2ndf ngc253q.dat ngc253 q 100 shape=[100,60]
```

This copies a quality array from the unformatted file ngc253q.dat to an existing NDF called ngc253 (such as created in the first example). The input file does not contain a header section. The NDF is two-dimensional: 100 elements in x by 60 in y. Its data array has type _UBYTE. The data records each have 100 values.

```
unf2ndf ngc253.dat ngc253 fits noperec=!
```

This copies a data array from the unformatted file ngc253.dat to the NDF called ngc253. The input file contains a FITS-like header section, which is copied to the FITS extension of the NDF. The shape of the NDF is controlled by the mandatory FITS keywords NAXIS, AXIS1, ..., AXISn, and the data type by keywords BITPIX and UNSIGNED. Each data record has AXIS1 values (except perhaps for the last).

```
unf2ndf type="_uword" in=ngc253.dat out=ngc253 \
```

This copies a data array from the unformatted file ngc253.dat to the NDF called ngc253. The input file does not contain a header section. The NDF has the current shape and data type is unsigned word. The current number of values per record is used.

```
unf2ndf spectrum zz skip=2 shape=200 noperec=!
```

This copies a data array from the unformatted file spectrum to the NDF called zz. The input file contains two header records that are ignored. The NDF is one-dimensional comprising 200 elements of type _REAL. There is one data record containing the whole array.

```
unf2ndf spectrum.lis ZZ skip=1 fits noperec=20
```

This copies a data array from the unformatted file spectrum.lis to the NDF called ZZ. The input file contains one header record, that is ignored, followed by a

FITS-like header section, which is copied to the FITS extension of the NDF. The shape of the NDF is controlled by the mandatory FITS keywords NAXIS, AXIS1, ..., AXIS*n*, and the data type by keywords BITPIX and UNSIGNED. Each data record has AXIS1 values (except perhaps for the last).

Notes:

The details of the conversion are as follows:

- the unformatted-file array is written to the NDF array as selected by COMP. When the NDF is being modified, the shape of the new component must match that of the NDF.
- If the input file contains a FITS-like header, and a new NDF is created, *i.e.* COMP = `"Data"`, the header records are placed within the NDF's FITS extension. This enables more than one array (input file) to be used to form an NDF. Note that the data array must be created first to make a valid NDF, and it's the FITS structure associated with that array that is wanted. Indeed the application prevents you from doing otherwise.
- The FITS-like header defines the properties of the NDF as follows:
  - BITPIX defines the data type: 8 gives _BYTE, 16 produces _WORD, 32 makes _INTEGER, 64 creates _INT64, -32 gives _REAL, and -64 generates _DOUBLE. For the first two, if there is an extra header record with the keyword UNSIGNED and logical value T, these types become _UBYTE and _UWORD respectively. UNSIGNED is non-standard, since unsigned integers would not follow in a proper FITS file. However, here it is useful to enable unsigned types to be input into an NDF. UNSIGNED may be created by this application's sister, NDF2UNF. BITPIX is ignored for QUALITY data; type _UBYTE is used.
  - NAXIS, and NAXIS*n* define the shape of the NDF.
  - The TITLE, LABEL, and BUNIT are copied to the NDF TITLE, LABEL, and UNITS NDF components respectively.
  - The CDELT*n*, CRVAL*n*, CTYPE*n*, and CUNIT*n* keywords make linear axis structures within the NDF. CUNIT*n* define the axis units, and the axis labels are assigned to CTYPE*n* If some are missing, pixel co-ordinates are used for those axes.
  - BSCALE and BZERO in a FITS extension are ignored.
  - BLANK is not used to indicate which input array values should be assigned to a standard bad value.
  - END indicates the last header record unless it terminates a dummy header, and the actual data is in an extension.
- Other data item such as HISTORY, data ORIGIN, and axis widths are not supported, because the unformatted file has a simple structure to enable a diverse set of input files to be converted to NDFs, and to limitations of the standard FITS header.

Related Applications :

CONVERT: NDF2UNF.

B Handling NDFs in IDL

B.1 The Easy Way

Note that this method cannot be used with the 64-bit Solaris version of IDL (use `% idl -32`).

IDL function READ_NDF is available to convert a component of an NDF to an IDL array, and IDL procedure WRITE_NDF will create an NDF component from an IDL array. So, for example:

```
IDL> tv,read_ndf('comwest')
```

will display the data array of the NDF, `comwest.sdf`, using the IDL command, `TV`, and:

```
IDL> write_ndf,field,'stars'
```

will create an NDF, `stars.sdf`, of a suitable type and size, and write the IDL array, `field`, to its `DATA_ARRAY` component.

```
IDL> write_ndf,q,'stars','QUALITY'
```

will write the IDL array, `q`, to the `QUALITY` component of an existing NDF, `stars.sdf`.

Both READ_NDF and WRITE_NDF can take special action on bad values. For a full description of their arguments see

Appendix C.

Complete structures can be handled by function HDS2IDL and procedure IDL2HDS. So, for example:

```
IDL> comwest=hds2idl('comwest')
% Loaded DLM: HDS2IDL.
IDL> help,comwest,/str
** Structure <40071208>, 5 tags, length=262184, refs=1:
   HDSSTRUCTYPE    STRING    'IMAGE'
   TITLE           STRING    'Comet West, low resolution'
   DATA_ARRAY      FLOAT     Array[256, 256]
   DATA_MIN        FLOAT           3.89062
   DATA_MAX        FLOAT           245.937
IDL>tv,comwest.data_array
```

will display the same image as the READ_NDF example above, but the other components of the NDF are also available in the IDL structure, `comwest`, so that:

```
IDL> idl2hds,comwest,'newcomwest'
```

will create a duplicate of `comwest.sdf` in `newcomwest.sdf`. Exact duplication of the type and structure is not always possible – see the routine descriptions for details.

When CONVERT is installed, the converter procedures and routines are placed in `$CONVERT_DIR` so, to make them available to IDL, that directory must be added to the IDL search paths, `IDL_PATH` and `IDL_DLM_PATH`. This will be done if the environment variable IDL_PATH has been set (usually by 'sourcing' the `idl_setup` script) when you start the CONVERT package by typing:

```
% convert
```

Note that `convert` must be run *after* sourcing the `idl_setup` script.

Note also that having started CONVERT *the NDF library (see SUN/33), which is ultimately used by READ_NDF and WRITE_NDF, will allow them (but not HDS2IDL and IDL2HDS) to do on-the-fly conversion (see Section 3) of any files given as parameters. This opens up the possibility of using almost any data format with IDL.*

As an example:

```
IDL> tv,read_ndf('moon.imh')
```

will display the IRAF file, `moon.imh`.

B.2 Other Methods

B.2.1 A simple route (but rather slow)

The simplest route to use when generating data for IDL from NDFs is to create an ASCII copy of the NDF you are interested in using the CONVERT package application NDF2ASCII and then read the resulting file with IDL. The steps taken might be something like:

```
% ndf2ascii in=imagein out=fileout
```

This will create a file, `fileout`, containing the DATA component of the NDF called `imagein`. If you want to store the VARIANCE or QUALITY components you would use `comp=v` or `comp=q` respectively as additional parameters.

You can then employ a simple IDL batch file such as:

```
; Create IMAGE an 339x244 single precision floating point array.
IMAGE=FLTARR(339,244)

; Open the existing NDF2ASCII file "fileout" for read only access.
OPENR, UNIT, 'fileout', /GET_LUN

; Read formatted input from the specified file unit and
; place in the variable "IMAGE".
READF, UNIT, IMAGE

; Closes the file unit used.
CLOSE, UNIT

; Display the image after suitable scaling.
TVSCL, IMAGE
```

The above example assumes image dimensions of 339×244 pixels. If you are in any doubt as to the dimensions of your image you can determine them using the KAPPA application NDFTRACE.

One advantage of this route is that the ASCII data can instead be read directly into byte, integer or double-precision arrays/structures by simply substituting `INTARR`, `DBLARR` or `BYTARR` for `FLTARR`. Clearly, it should be remembered that attempting to represent floating-point values within a byte array will not work properly, whereas a double-precision array will accommodate double-precision, floating-point, integer or byte values (though somewhat inefficiently in terms of memory consumption).

B.2.2 A faster route (but a little more complicated)

However, the NDF2ASCII routine is not fast and makes this route awkward if time is important. Consequently, you may want to use NDF2UNF to create an F77 unformatted sequential file thus:

```
% ndf2unf in=imagein out=fileout noperec=339
```

Where the `noperec` number should be the size of the first axis of the image.

You can then read the created file using an IDL batch file similar to:

```
; Supply the name of the image and its dimensions.
FNAME='fileout'
SD1=339
SD2=244

; Set up the main array and temporary array
; use NOZERO option to avoid initialisation
IMAGE=INTARR(SD1,SD2,/NOZERO)
TEMP= INTARR(SD1,    /NOZERO)

; Display what is going on.
PRINT, "Converting file: ", FNAME

; Open the file generated by NDF2UNF for read access only.
OPENR, UNIT, fname, /GET_LUN, /F77_UNFORMATTED

; Read the image one record at a time.
FOR I=0,SD2-1  DO BEGIN  READU, UNIT, TEMP & $
  ; Transfer each line into the main image array.
  FOR J=0,SD1-1 DO BEGIN IMAGE(J,I)=TEMP(J) & $
ENDFOR & ENDFOR

; Close the opened file unit.
CLOSE, 1

; Scale the image for display.
IMAGE=CONGRID(IMAGE,SD1,SD2,/INTERP)
WINDOW, 0,XSIZE=SD1,YSIZE=SD2

; Display the image.
TVSCL, IMAGE
```

The image is then contained in the integer array `IMAGE` and may be manipulated by IDL. This would allow such operations as storing it as a UNIX unformatted file where the image might subsequently be read in from disc in one go.

It should be remembered that the data type of the F77 unformatted file created by NDF2UNF may differ depending on the type of data stored in the original NDF. If this is the case you might need to change the type definition of the arrays `image` and `temp` to reflect this. The data type used within each component of an NDF may be determined using NDFTRACE.

B.2.3 Using the IDL Astronomy Users' Library

If this all seems a bit tedious, then those of you with with the IDL Astronomy Users' Library installed on your machines might choose to take advantage of its FITS conversion procedures to make life easier still. Users wishing to obtain a copy of the library can find it at `http://idlastro.gsfc.nasa.gov/homepage.html`.

The library contains IDL procedures from a number of sources that allow FITS format files to be read into IDL data structures. The routine chosen for the example below was FXREAD which may be found in the `/pro/bintable` sub-directory. It is part of a comprehensive suite of FITS conversion programs that seems particularly easy to use.

So if you first convert your NDF into a FITS file using NDF2FITS like this:

```
% ndf2fits in=m42 out=m42.fit comp=d
```

you can then use the following code from within IDL to place the image into an IDL structure and display it.

```
; Read the FITS file
fxread, 'm42.fit', DATA, HEADER

; Determine the size of the image
SIZEX=fxpar(header, 'NAXIS1')
SIZEY=fxpar(header, 'NAXIS2')

; Find the data type being read
DTYPE=fxpar(header, 'BITPIX')

; Scale the image for display
IMAGE=congrid(DATA,SIZEX,SIZEY,/INTERP)
window, 0,xsize=SIZEX,ysize=SIZEY

; Display the image
TVSCL, IMAGE
```

As can be seen above, various values contained within the FITS header of the original can be obtained using the FXPAR procedure.

You should bear in mind that a number of other procedures (such as IEEE_TO_HOST and GET_DATE) from the Astronomy Users' Library are also needed by IDL when compiling this code. Consequently it is essential that the whole library should be obtained from the archive.

C Specifications of CONVERT IDL Procedures

READ_NDF
Convert a Starlink NDF to an IDL array.

Description:

This IDL function will convert a Starlink NDF file of up to seven dimensions to an IDL array of an appropriate type and shape. Bad values in the NDF may be converted to specific values in the IDL array.

If NDF on-the-fly conversion has been activated, the given filename may refer to a file of a different data format which is to be converted.

Usage:

```
Result = READ_NDF( Ndf_name[, Bad_value][,COMPONENT=Comp_name])
```

Arguments:

Ndf_name

A string expression specifying name of the NDF to be read.

Bad_value

Optional - A value to replace in the IDL array any occurrence of the PRIMDAT bad value in the NDF component. The value must be the same type as the array.

Keywords:

COMPONENT

Set this to a string expression specifying the NDF component to be read. It may be `"DATA"`, `"VARIANCE"` or `"QUALITY"` and defaults to `"DATA"`. The case of the string does not matter and it may be abbreviated to one or more characters.

Returned Value:

Result An IDL array of a size and type corresponding with the NDF. The type correspondence is as follows:

_REAL –> floating

_DOUBLE –> double-precision

_UBYTE –> byte

_WORD –> integer

_INTEGER –> longword integer

Examples: Assuming `my_ndf.sdf` is an NDF of type _REAL,

```
IDL> data_array = read_ndf('my_ndf')
```

creates an IDL floating array, data_array, with the same dimensions as the NDF and containing the values from its DATA component.

```
IDL> data_array = read_ndf('my_ndf', !values.f_nan)
```

As above except that any occurrence of a bad value (VAL__BADR as defined by the Starlink PRIMDAT package) in the NDF will be replaced by NaN in the IDL array.

```
IDL> quality = read_ndf('my_ndf',comp='q')
```

creates an IDL byte array from the QUALITY component of the same NDF. (The QUALITY component is always type _UBYTE.) Note that the keyword 'component' and the value 'QUALITY' are case-independent and can be abbreviated.

Deficiencies :

No conversion of the given bad value to the appropriate type for the array will be attempted; instead an error will be reported.

Related Applications :

CONVERT: WRITE_NDF.

WRITE_NDF
Convert an IDL array to a Starlink NDF.

Description:

This IDL procedure will write an IDL array of up to seven dimensions to a Starlink NDF. If NDF on-the-fly conversion has been activated, the given filename may refer to a file of a different data format in which case the NDF is then automatically converted to the required file type.

Usage:

```
IDL> write_ndf, IDL_array, Ndf_name[, Bad_value][, COMPONENT=Comp_name]
```

Arguments:

IDL_array

The IDL array to be converted. This may be an array name or constant of up to seven dimensions. The type of the NDF component created will depend on the type of the given array:

floating –> _REAL

double-precision –> _DOUBLE

byte –> _UBYTE

integer –> _WORD

longword integer –> _INTEGER

No other types are allowed.

Ndf_name

A string expression specifying name of the NDF to be written.

Bad_value

Optional - A value any occurrence of which in the IDL array is to be replaced by the appropriate PRIMDAT bad value in the NDF component. If no such value is found, the NDF bad-pixel flag for the component is set FALSE. The value must be the same type as the array.

Keywords:

COMPONENT

Set this to a string expression specifying the NDF component to be written. The following values are allowed:

`"DATA"` — A new NDF is created with the same dimensions as the IDL array, and the DATA component written.

`"VARIANCE"` — An existing NDF is opened and a new component written. The size of the given array must be the same as the NDF.

`"QUALITY"` — An existing NDF is opened and a new component written. The size of the given array must be the same as the NDF and the type of the IDL array must be Byte.

The case of the string does not matter and it may be abbreviated to one or more characters.

Examples: Assuming `my_data` is an IDL floating array,

```
IDL> write_ndf, my_data, 'my_ndf'
```

creates the NDF 'my_ndf.sdf' with the same dimensions as the IDL array 'my_data', and writes the array to its DATA component (of type _REAL). No checks on bad values are made.

```
IDL> write_ndf, my_data, 'my_ndf', !values.f_nan
```

As above except that any occurrence of the value NaN in the array will be replaced by the VAL__BADR value as defined by the Starlink PRIMDAT package.

```
IDL> write_ndf, my_variances, 'my_ndf', comp='v'
```

Writes the IDL array 'my_variances' to the VARIANCES component of the NDF created above. A check is made that the size of the array corresponds with the size of the NDF. (Note that the keyword COMPONENT and the value `"VARIANCE"` are case-independent and can be abbreviated.)

Deficiencies :

No conversion of the given bad value to the appropriate type for the array will be attempted; instead an error will be reported.

Related Applications :

CONVERT: READ_NDF.

HDS2IDL
Convert a Starlink HDS file to an IDL variable.

Description:

This IDL function will convert a Starlink HDS object into an IDL variable. The object may be a structure or primitive so a complete NDF structure can be obtained, instead of just the single component produced by function READ_NDF.

Usage:

```
Result = HDS2IDL( filename )
```

Arguments:

filename

A string expression specifying an HDS object. The specification may include slices and cells of arrays.

Returned Value:

Result

An IDL variable corresponding to the HDS object. *Structures and primitive types are not necessarily identical (see "Notes").*

Notes:

- Type correspondence is as follows:
  _REAL –> floating
  _DOUBLE –> double-precision
  _UBYTE –> byte
  _WORD –> integer
  _INTEGER –> longword integer
  _LOGICAL –> longword integer (with name change, see below)
  _CHAR –> string (see below)
  _BYTE –> integer (see below)
  _UWORD –> longword integer (see below)
- IDL structures will have an additional STRING component, named HDSSTRUCTYPE, specifying the type of the HDS structure to which it corresponds.
- _LOGICAL HDS components become IDL LONG components with LOGICAL_ prefixed to their name. Any IDL byte, integer or longword structure component must have this name convention if it is to be converted to an HDS _LOGICAL component by `IDL2HDS`.
- IDL strings created from HDS _CHAR components will have trailing spaces removed, so there is no way to determine the size of the original HDS component.

- HDS types _BYTE and _UWORD become IDL components indistinguishable from components produced from HDS types _WORD and _INTEGER so, if they are converted back to HDS by `IDL2HDS`, their HDS type will have changed.
- In an IDL array of structures, each element must be exactly the same structure. For HDS this is not the case, therefore an HDS array of structures, *NAME*, will become an IDL structure, *NAME*, with component HDSSTRUCTYPE set to *TYPE*(*n,m,...*) (where *TYPE* is the type of the HDS array of structures, and *n*, *m etc.* are the dimensions) and components *NAME_i_j... etc.* (where *NAME_i_j... etc.* are structures, one for each element of the HDS array of structures).

Examples:

```
IDL> data_struct = hds2idl('my_file')
```

Assuming `my_file.sdf` is an HDS, file this creates an IDL structure corresponding to it.

Deficiencies :

- It is not possible to obtain an identical structure in all cases (see "Notes").
- Complex values are not handled.

Related Applications :

CONVERT: IDL2HDS.

IDL2HDS
Convert an IDL variable to a Starlink HDS file.

Description:

This IDL procedure will create a Starlink HDS file corresponding to the IDL variable, which may be a scalar, array or structure.

Usage:

```
IDL> idl2hds, IDL_struct, filename
```

Arguments:

IDL_struct

The IDL variable to be written.

filename

A string expression specifying name of the HDS file to be written. *The HDS structure is not necessarily identical to the IDL structure (see "Notes").*

Notes:

- Type correspondence is as follows:
  floating –> _REAL
  double-precision –> _DOUBLE
  byte –> _UBYTE (unless _LOGICAL, see below).
  integer –> _WORD (unless _LOGICAL, see below).
  longword integer –> _INTEGER (unless _LOGICAL, see below).
  string –> _CHAR*(IDL string size)
- Each HDS structure will have a type as defined by the HDSSTRUCTYPE component of the IDL structure. If there is no such component, the HDS type is blank.
- Byte, integer or longword integer components which have LOGICAL_ prefixed to their name will be converted to _LOGICAL HDS components and the prefix will be removed from the name.
- IDL strings will be converted into HDS _CHAR components whose size is the size of the IDL string. If a size greater than the used length is required, the IDL string must be padded with blanks.
- HDS arrays of structures will be created from IDL structures having the form produced from HDS arrays of structures by HDS2IDL.

Examples:

```
IDL> idl2hds, data_struct, 'my_file'
```

Assuming `data_struct` is an IDL structure, this creates the HDS file '`my_file.sdf`' with a corresponding structure.

Deficiencies :

- It is not possible to obtain an identical structure in all cases (see "Notes").
- It is not possible to produce HDS components of type _UWORD or _BYTE.
- Complex values are not handled.
- IDL arrays of more than 1 structure are not handled.
- Only a complete HDS file can be written.

Related Applications :

CONVERT: HDS2IDL.

D IRAF Versions

The CONVERT utilities NDF2IRAF and IRAF2NDF are built using copies of relevant IRAF libraries (which are included in the CONVERT release) so they exhibit the same behaviour as the IRAF version from which the libraries were extracted. (There are also some IRAF dependencies in the so-called SPP routines of CONVERT – these originate written in the IRAF SPP language and include header files defining the layout of the IRAF image.) The versions of IRAF2NDF and NDF2IRAF which you use must therefore be compatible with the version of IRAF which you are using.

A new version of the IRAF image format was developed for IRAF Version 2.11 onwards. IRAF Version 2.11 onwards will read either the old or new image formats but will produce the new format by default. (It can be made to produce old-format images by setting environment variable `oifversion=1`.)

CONVERT contains IRAF V2.11 compatible versions of the SPP routines and the IRAF libraries. If you are still running IRAF V2.10, set environment variable `oifversion=1` before running NDF2IRAF. (This includes when running Starlink programs from IRAF cl if an output image is produced by 'on-the-fly' conversion.)

Details of which IRAF version libraries are used in CONVERT will be given in the release notes.

E Release Notes

E.1 Release Notes – V1.8

NDF2FITS has a new literal parameter called AXISORDER to specify the order of WCS axes in the output FITS header. This is introduced so that the generated FITS file may be interpeted correctly by those FITS readers that assume certain axis orders. For example some expect equatorial co-ordinates to be ordered right ascension then declination, whereas the FITS standard lets you permute WCS axes in any order.

IRAF release V2.11.3 libraries are used in this release for building NDF2IRAF and IRAF2NDF.

F Notes from Previous Few Releases

F.1 Release Notes – V1.5

SPECX2NDF has been revamped so that it now uses the new AST SpecFrame functionality, allowing translation between spectral frames without re-running SPECX2NDF. The SPECTRUM parameter (and associated parameters SOR, DOPPLER) are no longer required since they can be changed after conversion using WCSATTRIB application. *Scripts using SPECX2NDF may need modification.* In addition to dealing with map files SPECX2NDF has now been extended to deal

with SPECX data files; each spectrum in the file is translated to an NDF spectrum in the output HDS container file.

FITS2NDF supports INES archive IUE spectra.

Improved propagation of existing world co-ordinate system (WCS) headers in NDF2FITS partially from improvements to the AST subroutine library. For example, long-slit spectra with a three-dimensional WCS, but stored in a two-dimensional image, retain their three-dimensional WCS headers. Comments are preserved where values have not changed significantly.

The references to the old VMS-only tasks have been removed from the documentation because the residual VMS service no longer exists.

F.1.1 Release Notes – V1.5-4

Added IRAF compatibility libs for Linux systems. CONVERT should now build on any ix86 Linux platform.

F.1.2 Release Notes – V1.5-5

Added FITS-AIPS++ and FITS-CLASS encodings to FITS2NDF and NDF2FITS.

Clarified the description of UNF2NDF parameter TYPE so that it is clear that the type given should also be of the input data, not just the output NDF.

SPECX2NDF now creates NDF files using the new double-sideband SpecFrame. See the AST documentation for more details on double-sideband spectra.

F.1.3 Release Notes – V1.5-6

FITSGZ is a new on-the-fly conversion format for GZIP-compressed FITS files. The recognised extensions are `fits.gz`, `fit.gz`, and `fts.gz`.

FITS2NDF supports the new AAO Instruments (AAOMEGA and FMOS) that use the 2dF data structures.

F.1.4 Release Notes – V1.5-7

FITS2NDF has a new TYPE group parameter to set the data type of the NDF, overriding the value propagated from the FITS BITPIX, or BSCALE and BZERO precision when FMTCNV is `TRUE`.

NDF2FITS now supports multi-NDFs HDS container files through the new CONTAINER and MERGE parameters.

F.1.5 Release Notes – V1.5-8

FITS2NDF Parameter FMTCNV has a new allowed value of `"Native"` requesting that there is no format conversion, and the array of numbers stored in the FITS file are copied to a scaled array within the NDF. This preserves the data compression. Parameter TYPE continues to control the data type of the true unscaled values.

NDF2FITS Parameter BITPIX has a new allowed value of `"Native"`. This requests that should any scaled-form arrays be converted, then the data type of the corresponding output FITS array is set to the scaled-form array's data type, and that the scale and offset coefficients for the format conversion are taken from the NDF's scaled array too. This new facility preserves the data compression of large files. In the absence of a scaled array, the application behaves as if BITPIX=`-1` were specified.

F.1.6 Release Notes – V1.5-9

NDF2FITS Parameter ENCODING has a new allowed value of `"FITS-WCS(CD)"`. This is the same as `"FITS-WCS"` except that it uses the old CD matrix formalism to describe the data array's rotation and scaling.

F.1.7 Release Notes – V1.5-10

FITS2NDF supports externally and internally compressed FITS files.

NDF2FITS writes integrity check keywords CHECKSUM and DATASUM at the end of each header if new Parameter CHECKSUMS is `TRUE`.

F.1.8 Release Notes – V1.5-11

NDF2FITS writes the correct BUNIT keyword value in the IMAGE extension storing the VARIANCE component. The BUNIT keyword is absent for a QUALITY array.

There is a new DUPLEX parameter. When set `TRUE` (and PROFITS is also `TRUE`), it permits the FITS airlock headers to appear also in the IMAGE extensions for the VARIANCE and QUALITY arrays.

F.1.9 Release Notes – V1.5-12

NDF2FITS now makes special provision for the JCMT SMURF-package extension. It treats the extension contents as NDFs rather than arbitrary HDS structures.

F.1.10 Release Notes – V1.5-13

NDF2FITS supports the propagation of provenance information to FITS headers. There is a choice of generic propagation that attempts to propagate all the information, or to write CADC-specific headers, or to exclude provenance (the default), governed by the new PROVENANCE parameter.

NDF2FITS now handles extensions containing only NDFs by adding a dummy FITS sub-file that retains the name and type of the wrapper structure.

FITS2NDF processes SMURF-package data better, permitting a roundtrip via FITS, perserving the original data structures, save for some additional FITS headers.

F.1.11 Release Notes – V1.5-14

The creation of AXIS structures by SPECX2NDF can now be suppressed using its new AXIS parameter.

NDF2FITS is now much quicker when writing out WCS information that includes a large table of values.

F.1.12 Release Notes – V1.5-15

The CADC provenance processing now also modifies the PRODUCT keyword's value in FITS extensions to the original PRODUCT string followed by underscore and the extension name in lowercase. Also the CADC provenance now only records unique OBSIDD values.

F.1.13 Release Notes – V1.5-16

The FITS converters use the new NDG provenance system that inherits history records too. NDF2FITS also ignores hidden ancestors.

NDF2FITS can now handle files names with multiple fullstops in the path.

The formatting of history records written by FITSNDF has been improved, in particular the indentation after the first line. This preserves the formatting after an NDF→FITS→NDF cycle.

Various internal improvements were made, particularly to reuse KAPLIBS equivalents of CONVERT subroutines.

Correct a bug in FITS2NDF affecting the import of SMURF data containing VARIANCE or QUALITY that was formerly in an NDF.

F.1.14 Release Notes – V1.5-17

The bash CONVERT initialisation script has been changed so that it now works from any Bourne-compatible shell.

F.1.15 Release Notes – V1.5-18

FITS2NDF recognises the new `"Group:"` heading in HISTORY headers that originally came from an exported NDF. [This lists all the NDFs stored within a GRP group used in an application.] Thus the original paragraph and indentation may be preserved after an NDF→FITS→NDF cycle.

NDF2FITS allows three-letter permutations for Parameter COMP. There is now no automatic quality masking of data and variance whenever the quality array is exported to the FITS file too. A new subsection of the NDF2FITS reference documentation explains this in more detail.

Removed default history recording in FITS2NDF's output NDF since you can control automatic history creation via the NDF_AUTO_HISTORY environment variable.

F.1.16 Release Notes – V1.5-19

Propagate LABEL keyword to NDF label in FITS2NDF. FITS2NDF can now recreate data that were formerly in a UKIRT_HDS container file.

NDF2FITS now recognises the modified data type of the SMURF extension. It handles component names that are too long to fit in a single EXTNAME header by creating an EXTNAMEF keyword that uses the Long-string convention. It also attempts to use the FIT-WCS encoding before any other.

Added support for the SMURF_EXT type for the SMURF extension, and for extensions within the SMURF extension's NDFs.

SPECX2NDF converts the GSD bad value into the Starlink equivalent.

The IDL scripts READ_NDF and WRITE_NDF have been fixed to use STARLINK_DIR to specify the correct IDL library path.

F.1.17 Release Notes – V1.6

Support for FITS -TAB projection type added.

Add WCSCOMP parameter to FITS2NDF. It controls where world co-ordinate information is propagated in the output NDF.

Add USEAXIS parameter to NDF2FITS. It controls whether AXIS co-ordinate information can be stored in an alternate representation in the FITS headers, that may include a -TAB extension. The default behaviour is to not write alternate AXIS co-ordinates if a WCS component exists in the NDF.

Changed the default ORIGIN keyword from `Starlink Project, U.K.` to `Starlink Software` in NDF2FITS.

F.1.18 Release Notes – V1.7

Support for 64-bit integers is now available in ASCII2NDF, DA2NDF, FITS2NDF, NDF2ASCII, NDF2DA, NDF2FITS, NDF2UNF, and UNF2NDF.

NDF2FITS now avoids scaling when the range of data values lies within the scaled type's range, such as double to single precision.

NDF2FITS has a new boolean parameter called ALLOWTAB to permit or disable storage of tabular co-ordinates using the FITS WCS TAB algorithm.

SPECX2NDF has a new SYSTEM parameter that specifies the NDF's co-ordinate system.