**SUN/245.0**

Starlink Project
Starlink User Note 245.0

A. J. Chipperfield

21 Jan 2002



---

# HDSTOOLS
# Tools To Display and Edit HDS Objects
# Version 1.0
# User's manual

---

## Abstract

The HDSTOOLS package contains a number of tools to edit and display HDS objects. The tools originated in the ASTERIX package but have now been modified to run as normal Starlink tasks.

# Contents

# 1 Introduction

The Starlink Hierarchical Data System, HDS, allows data files to be constructed with structured and primitive components. They are usually written and read by scientific application programs but it is sometimes useful to create or edit them by hand or to inspect them with no meaning attached to the components.

The HDSTOOLS package contains a number of generally useful tools for this purpose. The tools originated in the Asterix package, which was written at Leicester and Birmingham Universities for analysis of X-Ray data. Asterix is no longer supported so the tools have been extracted to form the HDSTOOLS package, which will be supported by Starlink. HDSTRACE remains the program of choice for displaying the content of HDS objects.

The applications should run as before but the underlying parameter system has been simplified (to the normal Starlink parameter system) and some bugs have been fixed and deficiencies addressed.

# 2 Running the Applications

Startup scripts are provided to define commands for running HDSTOOLS programs direct from the Unix shell or from ICL. The following instructions assume your site has been set up in the standard Starlink way.

To initialise the HDSTOOLS package for running from the Unix shell, type:

```
% hdstools
```

This will `source` the file `/star/bin/hdstools/hdstools.csh` to set up aliases for the tools.

Similarly, to run from ICL, type:

```
ICL> hdstools
```

This will 'load' the ICL script `/star/bin/hdstools/hdstools.icl` to 'define' the required commands.

In both cases you may also want to set up the environment variables AST_LIST_SPOOL, OLDFILE and/or NEWFILE (see HDSTOOLS Output Devices, Section 4).

In addition to the normal commands, `hcopy`, `hcreate` *etc.*, commands with '`hdt_`' replacing the initial '`h`' (`hdt_copy`. *hdt_create etc.*) are defined to allow for the possibility of name conflict with other packages.

# 3 Supplying Parameters

## 3.1 Command Line Parameters

The Starlink (ADAM) parameter system used by the HDSTOOLS application programs allows parameter values to be specified on the command line by 'positional' or 'keyword' parameter

specifiers.

Keyword parameter specifiers have the syntax `keyword=value` where `name` is the parameter keyword. (For HDSTOOLS the keyword is always the same as the parameter name.) Keyword specifiers may appear in any order on the command line and the case of the keyword is not significant.

Parameter specifiers which are not keyword specifiers are positional – the first of them applies to the parameter whose position is defined as 1, the second to the parameter whose position is defined as 2 *etc.* For example:

```
% hcreate file dims=10 _REAL
```

The parameter whose position is 1 takes the value `file`, the parameter whose keyword is DIMS takes the value `10` and the parameter whose position is 2 takes the value `_REAL`. Precisely how the values are interpreted depends upon the type of the parameter.

Parameters of type _CHAR take a character string as their value. If a string containing space or comma is specified on the command line, the parameter system requires that it is enclosed in double quotes. (Positional specifiers must also be quoted if they contain'='.)

Parameters of type _LOGICAL take the value YES, Y, NO, N, TRUE, T, FALSE or F (regardless of case). Alternatively the keyword specifiers '*KEYWORD*' or 'NO*KEYWORD*', where *KEYWORD* is the parameter's keyword may be given to set TRUE or FALSE respectively.

Note that when programs are run from the Unix shell and values are specified on the command line, it is often necessary to protect characters from the shell by inserting \ or enclosing them in single quotes. Characters which are likely to cause trouble are: ()[]*!". This is not a problem when replying to prompts or when running from ICL.

## 3.2 Vpaths and Prompts

If a parameter value is not specified on the command line, its value may be obtained by following a 'value-path' (known as the vpath) until a value is found. The path consists of a list of possible 'default' sources, including prompting the user (which will always produce a value).

Prompts may include a 'suggested value' obtained by following a 'ppath' similar to the vpath. In response to a prompt, you can type in a value, terminated by the <return> key, or the suggested value may be used by just hitting the <return> key, or placed in the input area by hitting the <tab> key. The input area may be edited (see SUN/144 for details).

Other possible responses to a prompt are:

- ! The 'null' response. This will usually terminate the program but has a special meaning for some parameters (see the program description).
- !! The 'abort' response. This will invariably abort an HDSTOOLS program.
- ? The 'help' response. This will display a one-line help message about the parameter and then re-prompt.
- ?? The 'fullhelp' response. This will put the user into the full help system, displaying the help for the parameter. On exit from the help system the user will be re-prompted.

### 3.3 Specifying HDS Objects

An HDS object is specified as a pathname comprising a 'container file' optionally followed by a number of component names separated by '.' – all but the last component must be structures. For example:

```
image.data_array.data
```

specifies component DATA of a structured component DATA_ARRAY of the container file `image.sdf`. Note that the file extension (`.sdf` is not given and the case of component names is not significant.

Usually, where an object is required, a cell or slice of an array object may be specified instead. Cells and slices are specified in parentheses after the component name and consist of ranges for each dimension, separated by ','. Ranges take the form: *ll:ul*, where *ll* and *ul* are lower and upper limits. Either limit may be omitted implying the beginning or end. If the ':' is also omitted the upper and lower limits will be the same *i.e.* the one number given.

For example:

```
container(2).data_array.data(:50,100:150)
```

specifies elements 1 to 50 in the first dimension and 100 to 150 in the second the DATA component of the DATA_ARRAY structure of the second structure in the top-level structure array in container file `container.sdf`.

Note that when running direct from the Unix shell, the parentheses will need protecting.

### 3.4 'Special' Keywords

In addition to the parameter keywords, there are several 'special keywords'. You may find the following two useful:

| | |
|---|---|
| PROMPT | Forces a prompt for any values not defined on the command line, regardless of what is on the vpath. |
| ACCEPT | Forces the suggested value to be used if a prompt would otherwise occur. |

For example:

```
% hcreate prompt
```

will cause a prompt for all the parameters of `hcreate`, including the dimensions which would otherwise be defaulted to produce a scalar.

### 3.5 Global Parameters

Parameters may be 'associated' with a 'global' parameter and thus values may be shared between different programs. The association may be to write the parameter's value on successful completion and/or use its value on the vpath or ppath. Several HDSTOOLS programs use the global parameter HDSOBJ to remember the last-used HDS object and/or offer it as a suggested value at a prompt. In this way a sequence of operations may be easily performed on the same object.

### 3.6 Interface Files

Each program's parameters and their positions, keywords, vpaths, ppaths and associations *etc.* are defined in an 'Interface File' (see SUN/115)

Interface files are provided for each program but it is possible to use private customised files instead.

## 4 HDSTOOLS Output Devices

The HDSTOOLS programs for displaying objects require an output device to be specified. The value of the device parameter may be set to one of the following values:

| | |
|---|---|
| TERMINAL | The user terminal. |
| PRINTER | The printer. Output is written to a temporary file, `ast_io.temp`*nn* (where *nn* are two digits to provide a unique name) and then spooled using the command specified in environment variable AST_LIST_SPOOL (which must be set). |
| NEWFILE[=*fnm*] | Write to a new file whose name is specified by the environment variable NEWFILE (default *fnm* else `ast_print.lis`). |
| OLDFILE[=*fnm*] | Append to the file whose name is specified by the environment variable OLDFILE (default *fnm* else `ast_print.lis`). |
| other | Anything else is taken to be a new filename to be written. |

CONSOLE, STDOUT and blank are synonyms for TERMINAL. The case of the value is not significant and it may be abbreviated (except for filenames). When a new file is to be written, any existing file of the same name will be saved as its filename with ∼ appended. Any existing `filename`∼ will be overwritten.

If the NEWFILE or OLDFILE=filename form is used on the command line, the '=' must be protected from the parameter system which would assume you were trying to specify a parameter called NEWFILE or OLDFILE. There are two ways to do this:

- Enclose it all in double quotes `"NEW=filename"` (but note that the quotes will need protecting from the shell so you end up with `'"NEW=filename"'`, for example).
- Use the parameter keyword form `DEV=NEW=filename`.

## 5 MSG Tuning

HDSTOOLS applications output messages to the user via the Starlink MSG library and interrogate the environment variables to determine any MSG tuning parameters (see SUN/104).

For example, on Unix,

```
% setenv MSG_FILTER 1
```

will cause the application to run in 'QUIET' mode and output of the application version number and other purely informational messages will be prevented.

# 6 Acknowledgements

The following people from the University of Birmingham were involved in producing the original Asterix tools: David Allan, Bob Vallance, Richard Beard, Jim Peden, Trevor Ponman, Dick Willingale, Ray Forbes.

# 7 Specifications of the HDSTOOLS Applications

## 7.1 General Notes

### 7.1.1 The Usage Section

The command name and parameters are listed using the following conventions:

***par*** Positional parameter specifier for parameter *par*, in position order – a prompt will be issued if it is not specified,

**[***par***]** Optional positional parameter specifier for parameter *par*, in position order – a default will be used if it is not specified.

**[***par=***]** Optional keyword parameter specifier – a default will be used if it is not specified.

Note that the order of keyword specifiers is not significant. Positional parameters may also be specified in keyword form but that precludes the use of positional parameter specifications for parameters with a higher position.

### 7.1.2 The Parameters Section

Each parameter is listed in the form

*PAR* = *TYPE* (*access*)

Description

*PAR* is the parameter name – this also serves as the keyword for HDSTOOLS parameters.

*TYPE* is the parameter type. Type UNIV means that the type of the parameter will depend on the type of the argument given – an unquoted string is assumed to be an HDS object name (except for 'INTERNAL' parameters). If you want the type to be a string, you must enclose it in double quotes (and possibly protect them from the shell with further single quotes – HCOPY

has an example of this). Conversely, if the parameter type is '_CHAR' and you want to specify an HDS object containing the value, you must force the string to be interpreted as an object name by preceding it with @ (HGET has an example of this).

(In fact the parameter system treats all parameters as HDS objects. If you specify a primitive value on the command line, an HDS object is created to hold the value and that object is used by the parameter system.)

*Access* is 'Read' or 'Write' (HDSTOOLS does not use 'Update'). Note that this refers to the access made to the parameter and not to the access to any file or object represented by the value. For example the OUT parameter of HCOPY is read but its value is the name of an object which is written, whilst HGET actually writes a value to its ATTR parameter.

The parameter description gives a simple description of the parameter. (More information may be provided in the description of the application.) If there is an association with a global parameter, this is indicated by the global parameter name preceded by $<$ if it is used as a suggested value, and followed by $>$ if the global parameter value updated with this parameter's value on successful completion. If a default will be used without prompting, the default is appended enclosed in [ ]. These defaults will be given as the suggested value if prompting is forced.

### 7.1.3 The Examples Section

Examples are given, in most cases assuming the command is run from the Unix shell – the % represents the Unix prompt and is not typed by the user. Running from ICL will be similar but quotes, parentheses and square brackets *etc.* do not need special protection.

# HCOPY
## Copy HDS data objects

**Description:**

Copies a data object from one place to another. The copy is recursive, so if a structure is specified as input then everything below that level will be copied to the output destination.

A suitable output object will be created if necessary (replacing any existing unsuitable one of the same name) but any structure above the specified object must already exist. Any content of an existing component is lost.

**Usage:**

```
hcopy in out
```

**Parameters:**

**INP=UNIV (Read)**
Object being copied from, structured or primitive.

**OUT=CHAR (Read)**
Object being copied to - takes shape and type of input.

**Examples:**

```
% hcopy file1 file2
```

Copy complete container file.

```
% hcopy file.more.asterix.grafix.coltab tab1
```

Extract an object into its own container file.

```
% hcopy file1.data_array file2.data_array
```

Copy the component DATA_ARRAY in file1 to DATA_ARRAY in file2. The DATA_ARRAY component of file2 will be created, but file2 must already exist.

```
% hcopy '[1 2 3 4 5]' file.array
```

Copy the explicit values given into a component ARRAY within file.

```
% hcopy '"Counts/sec"' 'file.axis(2).units'
```

A more practical example of the above.

```
% hcopy 'file.axis(1)' 'file.axis(2)'
```

Copy the first element of the AXIS array to the second - works for structure or primitive arrays.

**Deficiencies :**

Error reporting could be more helpful.

# HCREATE
## Create an HDS data object of specified type and dimensions

**Description:**

Creates an HDS data object of specified type and dimensions. It will either create a completely new container file or a new object within an existing structure. An existing container file will be overwritten but an existing component within a file will not.

By default the object created will be a scalar (dimension 0). If you want to create an object of different shape then either supply the dimensions on the command line or force prompting with the PROMPT keyword.

Primitives are not given values and this action must be performed subsequently by HMODIFY, HFILL or HCOPY.

**Usage:**

```
hcreate inp type [dims]
```

**Parameters:**

**INP = UNIV (Read)**
Name of object. GLOBAL.HDSOBJ>

**TYPE = CHAR (Read)**
Type of object to be created.

**DIMS∗(∗) = INTEGER (Read)**
Dimensions of object, comma or space separated and enclosed in [ ] if more than one ([ ] optional in response to a prompt). [0]

**Examples:**

```
% hcreate file1 _integer '[10,25]'
```

Creates an HDS container file, file1, containing a 10x25 _INTEGER array. (Note that the square brackets have to be protected from the shell.)

```
% hcreate file1 struc
```

Creates an HDS container file, file1, containing a structure of type STRUC.

```
% hcreate file1.array _real '[10,25]'
```

Creates a 10x25 array named ARRAY of type _REAL in the HDS structure created in the previous example

**Valid Types :**

HDS divides objects into two classes, primitive and structured. The former contain simple data such as numbers or characters, whereas the latter contain collections of other objects, structured or primitive. The valid primitive types recognised by HDS are:

| Type | Equiv Fortran | Range |
|---|---|---|
| _LOGICAL | LOGICAL∗4 | .TRUE., .FALSE. |
| _UBYTE | not supported | 0..255 |
| _BYTE | BYTE | -128..127 |
| _UWORD | not supported | 0..65535 |
| _WORD | INTEGER∗2 | -32768..32767 |
| _INTEGER | INTEGER∗4 | |
| _REAL | REAL∗4 | |
| _DOUBLE | DOUBLE PRECISION | |
| _CHAR∗n | CHARACTER∗(n) | |

Any type not in the above list will be assumed to be a structured type.

# HDELETE
## Delete an HDS object

**Description:**

Deletes a named HDS data object. Everything below the level of the object specified is also deleted.

**Usage:**

```
hdelete inp
```

**Parameters:**

**INP = UNIV (Read)**

Name of object to be deleted.

**Examples:**

```
% hdelete file1.data
```

Deletes component 'data' from the container file's top-level structure.

```
% hdelete file1
```

Deletes container file file1.sdf

# HDIR
## Produce a simple summary of an HDS object

**Description:**

Produces a simple summary of a named HDS data object to a selected output. For a primitive object the name, type and value are given if scalar, or dimensions if non-scalar. For a structure the components are listed.

**Usage:**

```
hdir inp [dev]
```

**Parameters:**

**INP = UNIV (Read)**
HDS data object object to be summarized. <GLOBAL.HDSOBJ>

**DEV = _CHAR (Read)**
Output device (TERMINAL, PRINTER, OLDFILE, NEWFILE etc.). [TERMINAL]

**Examples:**

```
% hdir file1
```

Produces a summary of file1.sdf on the terminal.

```
% hdir file1 summary.lis
```

Produces a summary of file1.sdf in text file summary.lis.

```
% hdir file2 '"O=summary.lis"'
```

Appends a summary of file2.sdf to text file summary.lis.

```
% hdir file1.array O
```

Appends a summary of component ARRAY of file1.sdf to text file ast_print.lis, assuming environment variable OLDFILE is not set.

# HDISPLAY
## Display the contents of a primitive HDS object

**Description:**

This application outputs the contents of a specified primitive HDS data object of up to 7 dimensions, to a selected output.

A subset of the object (which may itself be a subset) may be specified. For integer data hex, octal or decimal formats may be chosen.

**Usage:**

```
hdisplay inp [dev] [slice=] [fmt=] [width=]
```

**Parameters:**

**INP = UNIV**
HDS data object to be displayed. <GLOBAL.HDSOBJ>

**DEV = _CHAR (Read)**
Output device (TERMINAL, PRINTER, OLDFILE, NEWFILE etc.). [TERMINAL]

**SLICE = _CHAR**
Description of data subset (range in 1st dimension {,range in 2nd dimension},{...}) eg. 5:10,1:5 for 2-D, 10:20 for 1-D. The range specifier for each dimension follows the same convention as the FORTRAN substring specifier so ":50" "20:" "5:10,:" are all valid. [∗ (whole object)]

**FMT = _CHAR**
A Fortran FORMAT string to be used for formatting numbers. [! (use an appropriate general format for the type)]

**WIDTH = _INTEGER**
Output page width. [! (Use width appropriate for device)]

**Examples:**

```
% hdisplay file.data_array
```

Display whole of specified array on terminal with default format.

```
% hdisplay img.quality dev=p slice='"100:120,250:300"' fmt=z1
```

Outputs the specified slice of a quality array in hex format to the printer.

```
% hdisplay accept width=132
```

Outputs the current HDS object to the terminal with page width 132.

# HFILL
## Fill an HDS data object with a specified value

**Description:**

This application allows primitive data object of any shape and size to be filled with a single specified value.

Every effort is made to convert the value to the required type, using HDS rules. Error DAT__CONER is reported if this is not possible.

The defined object may be a slice of a larger array. If the array was previously undefined, it becomes defined - other elements are initialised to zero (or blank for type _CHAR).

**Usage:**

```
% hfill inp value
```

**Parameters:**

**INP = UNIV (Read)**
Name of object. <GLOBAL.HDSOBJ>

**VALUE = UNIV (Read)**
The value to be used.

**Examples:**

```
% hfill cfile.real 0
```

Puts the value 0.0 into every element of component REAL of container file cfile.sdf.

```
% hfill 'cfile.real(2,1:)'  5.5
```

Puts value 5.5 in each element of the specified slice of OBJECT (assuming the slice specification is valid for OBJECT).

```
% hfill cfile.real '"a"'
```

Fails as the value 'a' cannot be converted to _REAL.

```
% hfill cfile.chars '" "'
```

Writes a blank string into every element of component CHARS of container file cont.sdf

**Deficiencies :**

There is a limit of 80 characters on the size of a character value which may be given. A value larger than this will cause a DAT__TRUNC error. A smaller value too large to fit in the specified object's elements will be silently truncated.

# HGET
# Return information about an object

**Description:**

This application returns many different pieces of information depending on the value of the ITEM parameter.

| Item code | Returned type | Description |
|---|---|---|
| PRIMITIVE | _LOGICAL | True if Object primitive |
| STRUCTURED | _LOGICAL | True if Object structured |
| NDIM | _INTEGER | Dimensionality |
| DIMS | _CHAR | Dimensions separated by commas |
| NELM | _INTEGER | Total number of elements |
| TYPE | _CHAR | Object type |
| VALUE | Object type | Object value (primitive scalar only) |
| MIN, MAX | _REAL | Min,max values in numeric array |

The 'internal' parameter ATTR is set to the requested value and, if ECHO is TRUE, the value is displayed on the user's terminal.

For items MIN and MAX, internal parameter INDEX is also set (but not displayed).

The application is particularly useful from ICL, where the values of the ATTR and INDEX parameters can be returned into ICL variables and thus used to control applications. Note however that a character value cannot be returned into a variable that has already been defined as a numeric type.

**Usage:**

```
hget inp item [attr] [index] [echo=] [version=]
```

**Parameters:**

**INP = UNIV (Read)**
Object to be interrogated. <GLOBAL.HDSOBJ>

**ITEM = _CHAR (Read)**
Item wanted (see Description). Case is not significant and the value may be abbreviated. <VALUE>

**ATTR = UNIV (Write)**
Attribute value.

**INDEX = _INTEGER (Write)**
The index position of the max or min value (treating arrays as vectors).

**ECHO = _LOGICAL (Read)**
Echo attribute value to standard output stream? [TRUE]

**VERSION = _LOGICAL (Read)**
Whether application version number is to be output. If the MSG_FILTER environment variable is set to 1, output will not occur anyway. [FALSE]

**Examples:**

```
% hget 'numvec(2)' value
```

Displays the value of the second element of object numvec

```
% hget file.data_array.data max attr=@info.max noecho
```

Writes the maximum value of the DATA component of the DATA_ARRAY component of file into the MAX component of file info. The value will not be displayed.

```
ICL> hget numvec min (minval) (index) echo=f



ICL> =index
```

Sets the ICL variable minval to the minimum value in vector numvec, and variable index to the position of the minimum value within numvec. The minimum value will not be displayed but the second ICL command will display the index.

**Notes:**

Internal parameters (ATTR and INDEX) are not saved in the task's parameter file. Their values can be written to HDS objects by specifying the name of an existing object of a suitable type on the command line. The object name must be preceded by @ for the ATTR parameter but this is not necessary for INDEX (see Example 2).

# HHELP
# Gives help about HDSTOOLS

**Description:**

Displays help about HDSTOOLS. It describes individual commands in detail.

See the Section "Navigating the Help Library" for details how to move around the help information, and to select the topics you want to view.

**Usage:**

```
hhelp [topic] [subtopic] [subsubtopic] [subsubsubtopic]
```

**Parameters:**

**TOPIC = LITERAL (Read)**
Topic for which help is to be given. [" "]

**SUBTOPIC = LITERAL (Read)**
Subtopic for which help is to be given. [" "]

**SUBSUBTOPIC = LITERAL (Read)**
Subsubtopic for which help is to be given. [" "]

**SUBSUBSUBTOPIC = LITERAL (Read)**
Subsubsubtopic for which help is to be given. [" "]

**Examples:**

```
hhelp
```

No parameter is given so the introduction and the top-level help index is displayed.

```
hhelp application
```

This gives help about the specified application.

```
hhelp application subtopic
```

This lists help about a subtopic of the specified application or topic. The hierarchy of topics has a maximum of four levels.

**Navigating the Help Library :**

The help information is arranged hierarchically. You can move around the help information whenever HHELP prompts. This occurs when it has either presented a screen's worth of text or has completed displaying the previously requested help. The information displayed by HHELP on a particular topic includes a description of the topic and a list of subtopics that further describe the topic.

At a prompt you may enter:

- a topic and/or subtopic name(s) to display the help for that topic or subtopic, so for example, "hdisplay parameters dev" gives help on DEV, which is a subtopic of Parameters, which in turn is a subtopic of HDISPLAY;
- a <RETURN> to see more text at a "Press RETURN to continue ..." request;
- a <RETURN> at topic and subtopic prompts to move up one level in the hierarchy, and if you are at the top level it will terminate the help session;
- a CTRL/D (pressing the CTRL and D keys simultaneously) in response to any prompt will terminate the help session;
- a question mark "?" to redisplay the text for the current topic, including the list of topic or subtopic names; or
- an ellipsis "..." to display all the text below the current point in the hierarchy. For example, "HDISPLAY..." displays information on the HDISPLAY topic as well as information on all the subtopics under HDISPLAY.

You can abbreviate any topic or subtopic using the following rules.

- Just give the first few characters, e.g. "PARA" for Parameters.
- Some topics are composed of several words separated by underscores. Each word of the keyword may be abbreviated, e.g. "Colour_Set" can be shortened to "C_S".
- The characters "%" and "∗" act as wildcards, where the percent sign matches any single character, and asterisk matches any sequence of characters. Thus to display information on all available topics, type an asterisk in reply to a prompt.
- If a word contains, but does end with an asterisk wildcard, it must not be truncated.
- The entered string must not contain leading or embedded spaces.

Ambiguous abbreviations result in all matches being displayed.

**Implementation Status:**

- Uses the portable help system.

# HMODIFY
## Modify the value of an HDS object

**Description:**

Allows the value of an HDS primitive data object to be changed. The new value(s) may be entered directly at the terminal or may be taken from another HDS object. Information about the object is displayed.

Every effort is made to convert the value(s) to the required type, using HDS rules. Error DAT__CONER is reported if this is not possible.

For _CHAR values, the character length of the value object need not match that of the object to be updated. If the used length of any element of the given value exceeds that of the output object, it will be truncated and a warning message displayed, trailing spaces will be silently truncated.

**Usage:**

```
hmodify inp value
```

**Parameters:**

**INP=UNIV (Read)**

Name of object to be modified - must be primitive. <GLOBAL.HDSOBJ>

**VAL=UNIV (Read)**

Value to be given to object - this can be an explicit value (including an array) entered at at the terminal or the name of another object. The value must be of the same size and shape as the object to be modified but will be converted to the correct type if possible. Character values must be quoted.

**Examples:**

```
% hmodify spectrum.data.temp 20000
```

Writes the value 20000 into the specified component

```
% hmodify 'ds.axis(1).units' '"Counts/s"'
```

Writes the value "Counts/s" into the UNITS component of first element of the AXIS array of structures in container file ds.sdf.

```
% hmodify 'ds.data_array(1:5)' '[1 2 3 4 5]'
```

Writes the given values into the specified slice of component DATA_ARRAY in container file ds.sdf.

```
% hmodify ds.data_array ds2.data_array
```

Replace the values of the first DATA_ARRAY with those in the second. They must be the same size.

# HREAD
## Read a file into an HDS object

**Description:**

Values are read, one per record, from the file and written to the specified HDS object. The number of values to be read is calculated from the size of the HDS object. If fewer values are found in the file, an error is reported but the given values will have been written, with the remainder unspecified. If the file is ASCII, it is read using a general format appropriate for the type of the specified object. The HDS object must exist and be primitive - it is written as if it were a vector.

**Usage:**

```
hread file out [binary=]
```

**Parameters:**

**FILE = _CHAR (Read)**
Name of file to be read.

**OUT = UNIV (Read)**
HDS object to receive data. <GLOBAL.HDSOBJ>

**BINARY = _LOGICAL (Read)**
Whether file is binary. [NO]

**Examples:**

```
% hread values.dat cfile.structure.data
```

Reads values from ASCII file values.dat and writes them to object STRUCTURE.DATA in file cfile.sdf.

```
% hread values.dat cfile.structure.data binary
```

Reads values from binary file values.dat and writes them to object STRUCTURE.DATA in file cfile.sdf.

# HRENAME
## Rename an HDS data object

**Description:**

The object is renamed.

Note that if INP is specified only as a container filename, the name of the top-level object contained will be changed but not the name of the file.

If the new name is too long, DAT__TRUNC is reported and a new value requested.

**Usage:**

```
hrename inp to
```

**Parameters:**

**INP = UNIV (Read)**
The object to be renamed.

**TO = _CHAR (Read)**
The new name - must be a valid HDS component name (not a pathname).

**Examples:**

```
% rename cfile.structure.data array
```

Component STRUCTURE.DATA becomes STRUCTURE.ARRAY

```
% rename cfile container
```

The top-level component of container file cfile.sdf (probably named CFILE) is renamed to CONTAINER

**Method :**

Uses subroutine DAT_RENAME.

# HRESET
## Change state of a primitive HDS object to undefined

**Description:**

The state of the specified object is set to 'undefined'. All subsequent read operations will fail until the object is written to (re-defined). An attempt to reset a slice of an object will reset the whole object, an attempt to reset a structure object will have no effect.

**Usage:**

```
hreset inp
```

**Parameters:**

**INP = UNIV (Read)**
The name of a primitive object.

**Examples:**

```
% hreset cfile.data_array.data
```

Resets the DATA component of structure DATA_ARRAY in file cfile.

```
% hreset 'cfile.data_array.data(1:10,1:10)'
```

Resets the whole of object cfile.data_array.data

```
% hreset cfile.data_array
```

No effect if DATA_ARRAY is a structure

**Method :**

Calls HDS subroutine DAT_RESET

# HRESHAPE
## Reshape an HDS object

**Description:**

Changes the dimensions of the specified object. The number of dimensions may be decreased but the total number of elements must remain the same unless only the length of the last (or only) dimension is changed. If the size is decreased, elements will be discarded; if it is increased, the value of additional elements is not defined. The operation will fail if the object is a structure array and any truncated elements contain components.

**Usage:**

```
hreshape inp dims
```

**Parameters:**

**INP = CHAR (Read)**

The name of the object to be re-shaped. <GLOBAL.HDSOBJ>

**DIMS∗(∗) = INTEGER (Read)**

New dimensions of object, comma or space separated and enclosed in [ ] if more than one ([ ] optional in response to a prompt).

**Examples:**

```
Assuming numarr is a 50x100 array of numbers:
```

```
% hreshape numarr 5000
```

Changes numvarr to a 5000 element vector.

```
% hreshape numarr '[50,50]'
```

Produces a 50x50 array of numbers. Elements [:50-100] are discarded.

```
% hreshape numarr '[50,100]'
```

After the last example this would restore the original shape of numarr but the previously discarded values may be lost.

**Method :**

Uses HDS subroutines DAT_ALTER or DAT_MOULD as appropriate.

# HRETYPE
## Change the type of an HDS structure object

**Description:**

Changes the type of an HDS structure object - the type of a primitive object cannot be changed.

**Usage:**

```
hretype inp newtype
```

**Parameters:**

**INP = UNIV (Read)**
The name of the object. <GLOBAL.HDSOBJ>

**TYPE = CHAR (Read)**
New type - a valid HDS non-primitive type.

**Examples:**

```
% hretype cfile.structure data_array
```

Sets the type of cfile.structure to DATA_ARRAY

```
% hretype cfile.structure.data _REAL
```

Error - cannot change the type of a primitive object.

**Method :**

CALLS DAT_RETYPE

# HTAB
## Display one or more vector objects in table form

**Description:**

Provides the facility to display vector HDS data objects simultaneously in a tabular form on a selected output. The range to be output is selectable but by default the whole of each object is output.

Note that the input object names must either be given on the command line or the PROMPT keyword must be used.

**Usage:**

```
htab in1 in2 ...  [dev=] [slice=] [width=]
```

**Parameters:**

**INPn = UNIV (Read)**
nth object in table. Up to six objects may be specified, <GLOBAL.HDSOBJ> for INP1, [! (no more objects)] for INP2 - 6.

**DEV = _CHAR (Read)**
Output device (TERMINAL, PRINTER, OLDFILE, NEWFILE etc.). [TERMINAL]

**SLICE = _CHAR (Read)**
Range of data to be output in form "n1:n2" ":n2" or "n1:". [∗ (whole object)]

**WIDTH = _INTEGER (Read)**
Output page width. A null (!) will result in a width appropriate for the chosen text output device. [!]

**Examples:**

```
% htab vec1 vec2 vec3 dev=n=vec.lis
```

Tabulate the three vectors to file vec.lis

```
% htab vec1 vec2 vec3 dev=printer slice=1:10
```

Tabulate the first ten elements of the given vectors on printer,

```
% htab vec1 vec2 vec3 vec4 vec5 vec6 width=132
```

Tabulate six vectors to terminal in 132 column mode

# HWRITE
## Write an HDS object into formatted/unformatted file

**Description:**

Writes values from an HDS primitive object into an ASCII (default) or binary file. One value is written per record.

For ASCII file output a default format is used unless explicitly overridden.

**Usage:**

```
hwrite inp file [binary=] [fmt=]
```

**Parameters:**

**INP = UNIV (Read)**
Object to be input and read. <GLOBAL.HDSOBJ>

**FILE = _CHAR (Read)**
Output filename.

**BINARY = _LOGICAL (Read)**
Whether file to be binary (Fortran unformatted). [NO]

**FMT = _CHAR (Read)**
Output format for ASCII file. If null (!) is specified, a default format is used for each different data type. [!]

**Examples:**

```
% hwrite cfile.structure.data values.dat
```

Write component DATA to ASCII file values.dat with default format

```
% hwrite cfile.structure.data values.dat fmt=F12.6
```

As above but with specified format

```
% hwrite cfile.structure.data values.dat binary
```

Write component DATA in binary form into sequential, unformatted file.