

STILTS - Starlink Tables Infrastructure Library Tool Set

Version 3.0-7



*Starlink User Note*256
Mark Taylor
10 June 2016

Abstract

STILTS is a set of command-line tools for processing tabular data. It has been designed for, but is not restricted to, use on astronomical data such as source catalogues. It contains both generic (format-independent) table processing tools and tools for processing VOTable documents. Facilities offered include crossmatching, format conversion, format validation, column calculation and rearrangement, row selection, sorting, plotting, statistical calculations and metadata display. Calculations on cell data can be performed using a powerful and extensible expression language.

The package is written in pure Java and based on STIL, the Starlink Tables Infrastructure Library. This gives it high portability, support for many data formats (including FITS, VOTable, text-based formats and SQL databases), extensibility and scalability. Where possible the tools are written to accept streamed data so the size of tables which can be processed is not limited by available memory. As well as the tutorial and reference information in this document, detailed on-line help is available from the tools themselves.

The STILTS application is available under the GNU General Public License (GPL) though most parts of the library code may alternatively be used under the GNU Lesser General Public License (LGPL).

Contents

| | |
|---|----------|
| Abstract..... | 1 |
| 1 Introduction..... | 7 |
| 2 The <code>stilts</code> command..... | 9 |

| | |
|---|-----------|
| 2.1 Stilts flags..... | 9 |
| 2.2 Task Names..... | 10 |
| 2.3 Task Arguments..... | 11 |
| 2.4 Getting Help..... | 12 |
| 3 Invocation..... | 15 |
| 3.1 Class Path..... | 16 |
| 3.2 Java Flags..... | 16 |
| 3.3 System Properties..... | 17 |
| 3.4 JDBC Configuration..... | 18 |
| 4 JyStilts - STILTS from Python..... | 20 |
| 4.1 Running JyStilts..... | 21 |
| 4.2 Table I/O..... | 22 |
| 4.3 Table objects..... | 23 |
| 4.4 Table filter commands (cmd_*)...... | 25 |
| 4.5 Table output modes (mode_*)...... | 26 |
| 4.6 Tasks..... | 26 |
| 4.7 Calculation Functions..... | 27 |
| 5 Table I/O..... | 28 |
| 5.1 Table Locations..... | 28 |
| 5.2 Table Formats..... | 29 |
| 5.2.1 Input Formats..... | 29 |
| 5.2.2 Output Formats..... | 30 |
| 6 Table Pipelines..... | 33 |
| 6.1 Processing Filters..... | 33 |
| 6.1.1 addcol | 34 |
| 6.1.2 addpixsample | 34 |
| 6.1.3 addresolve | 35 |
| 6.1.4 addskycoords | 35 |
| 6.1.5 assert | 36 |
| 6.1.6 badval | 36 |
| 6.1.7 cache | 36 |
| 6.1.8 check | 36 |
| 6.1.9 clearparams | 37 |
| 6.1.10 colmeta | 37 |
| 6.1.11 delcols | 37 |
| 6.1.12 every | 37 |
| 6.1.13 explodeall | 37 |
| 6.1.14 explodecols | 38 |
| 6.1.15 fixcolnames | 38 |
| 6.1.16 head | 38 |
| 6.1.17 keepcols | 38 |
| 6.1.18 meta | 39 |
| 6.1.19 progress | 39 |
| 6.1.20 random | 39 |
| 6.1.21 randomview | 40 |
| 6.1.22 repeat | 40 |
| 6.1.23 replacecol | 40 |
| 6.1.24 replaceval | 40 |
| 6.1.25 rowrange | 41 |
| 6.1.26 select | 41 |
| 6.1.27 seqview | 41 |
| 6.1.28 setparam | 41 |
| 6.1.29 sort | 41 |
| 6.1.30 sorthead | 42 |
| 6.1.31 stats | 42 |

| | | |
|----------|---|-----------|
| 6.1.32 | tablename | 43 |
| 6.1.33 | tail | 43 |
| 6.1.34 | transpose | 43 |
| 6.1.35 | uniq | 44 |
| 6.2 | Specifying a Single Column..... | 44 |
| 6.3 | Specifying a List of Columns..... | 45 |
| 6.4 | Output Modes..... | 45 |
| 6.4.1 | cgi | 45 |
| 6.4.2 | count | 46 |
| 6.4.3 | discard | 46 |
| 6.4.4 | gui | 46 |
| 6.4.5 | meta | 46 |
| 6.4.6 | out | 46 |
| 6.4.7 | plastic | 47 |
| 6.4.8 | samp | 47 |
| 6.4.9 | stats | 48 |
| 6.4.10 | topcat | 48 |
| 6.4.11 | tosql | 49 |
| 7 | Crossmatching..... | 50 |
| 7.1 | Match Criteria..... | 50 |
| 7.1.1 | sky: Sky Matching..... | 51 |
| 7.1.2 | skyerr: Sky Matching with Per-Object Errors..... | 52 |
| 7.1.3 | skyellipse: Sky Matching of Elliptical Regions..... | 53 |
| 7.1.4 | sky3d: Spherical Polar Matching..... | 53 |
| 7.1.5 | exact: Exact Matching..... | 54 |
| 7.1.6 | 1d, 2d, ...: Isotropic Cartesian Matching..... | 54 |
| 7.1.7 | 2d_anisotropic, ...: Anisotropic Cartesian Matching..... | 55 |
| 7.1.8 | 2d_cuboid, ...: Cuboid Cartesian Matching..... | 55 |
| 7.1.9 | 1d_err, 2d_err, ...: Cartesian Matching with Per-Object Errors..... | 56 |
| 7.1.10 | 2d_ellipse: Cartesian Matching of Elliptical Regions..... | 57 |
| 7.1.11 | Custom Matchers..... | 57 |
| 7.1.12 | Matcher Combinations..... | 57 |
| 7.2 | Multi-Object Matches..... | 58 |
| 8 | Plotting..... | 59 |
| 8.1 | Plot Parameters..... | 59 |
| 8.1.1 | Global Parameters..... | 60 |
| 8.1.2 | Layer Parameters..... | 60 |
| 8.1.3 | Animation..... | 61 |
| 8.2 | Surface Types..... | 62 |
| 8.3 | Layer Types..... | 63 |
| 8.3.1 | mark | 63 |
| 8.3.2 | size | 65 |
| 8.3.3 | sizexy | 67 |
| 8.3.4 | xyvector | 70 |
| 8.3.5 | xyerror | 72 |
| 8.3.6 | xyellipse | 75 |
| 8.3.7 | link2 | 77 |
| 8.3.8 | mark2 | 79 |
| 8.3.9 | line | 81 |
| 8.3.10 | linearfit | 82 |
| 8.3.11 | label | 84 |
| 8.3.12 | contour | 87 |
| 8.3.13 | density | 88 |
| 8.3.14 | fill | 90 |
| 8.3.15 | histogram | 92 |

| | |
|--|------------|
| 8.3.16 kde | 94 |
| 8.3.17 knn | 97 |
| 8.3.18 densogram | 101 |
| 8.3.19 function | 104 |
| 8.3.20 skyvector | 105 |
| 8.3.21 skyellipse | 107 |
| 8.3.22 skydensity | 110 |
| 8.3.23 xyzvector | 112 |
| 8.3.24 xyzerror | 114 |
| 8.3.25 yerror | 117 |
| 8.3.26 spectrogram | 119 |
| 8.4 Shading Modes..... | 120 |
| 8.4.1 auto | 120 |
| 8.4.2 flat | 121 |
| 8.4.3 translucent | 121 |
| 8.4.4 transparent | 122 |
| 8.4.5 density | 122 |
| 8.4.6 aux | 124 |
| 8.4.7 weighted | 124 |
| 8.5 Output Modes..... | 125 |
| 8.5.1 swing | 126 |
| 8.5.2 out | 126 |
| 8.5.3 cgi | 126 |
| 8.5.4 discard | 126 |
| 8.5.5 auto | 126 |
| 8.6 Export Formats..... | 126 |
| 9 Old-Style Plotting..... | 129 |
| 9.1 Parameter Suffixes..... | 129 |
| 10 Algebraic Expression Syntax..... | 132 |
| 10.1 Referencing Column Values..... | 132 |
| 10.2 Referencing Parameter Values..... | 133 |
| 10.3 Null Values..... | 134 |
| 10.4 Operators..... | 135 |
| 10.5 Functions..... | 136 |
| 10.5.1 Tilings..... | 136 |
| 10.5.2 Arithmetic..... | 137 |
| 10.5.3 Conversions..... | 139 |
| 10.5.4 Distances..... | 141 |
| 10.5.5 KCorrections..... | 143 |
| 10.5.6 Times..... | 147 |
| 10.5.7 TrigDegrees..... | 149 |
| 10.5.8 Maths..... | 150 |
| 10.5.9 Arrays..... | 153 |
| 10.5.10 Fluxes..... | 156 |
| 10.5.11 Strings..... | 158 |
| 10.5.12 Formats..... | 161 |
| 10.5.13 CoordsRadians..... | 162 |
| 10.5.14 Coverage..... | 165 |
| 10.5.15 Lists..... | 166 |
| 10.5.16 CoordsDegrees..... | 167 |
| 10.6 Examples..... | 168 |
| 10.7 Advanced Topics..... | 170 |
| 10.7.1 Expression evaluation..... | 170 |
| 10.7.2 Instance Methods..... | 170 |
| 10.7.3 Adding User-Defined Functions..... | 171 |

| | |
|---|------------|
| 11 Programmatic Invocation..... | 173 |
| Appendix A: Commands By Category..... | 175 |
| Appendix B: Command Reference..... | 177 |
| B.1 calc : Evaluates expressions..... | 177 |
| B.2 cdsskymatch : Crossmatches table on sky position against VizieR/SIMBAD table..... | 178 |
| B.3 coneskymatch : Crossmatches table on sky position against remote cone service..... | 183 |
| B.4 funcs : Browses functions used by algebraic expression language..... | 190 |
| B.5 pixfoot : Generates Multi-Order Coverage maps..... | 191 |
| B.6 pixsample : Samples from a HEALPix pixel data file..... | 193 |
| B.7 plot2plane : Draws a plane plot..... | 197 |
| B.8 plot2sky : Draws a sky plot..... | 207 |
| B.9 plot2cube : Draws a cube plot..... | 216 |
| B.10 plot2sphere : Draws a sphere plot..... | 227 |
| B.11 plot2time : Draws a time plot..... | 235 |
| B.12 plot2d : Old-style 2D Scatter Plot..... | 245 |
| B.13 plot3d : Old-style 3D Scatter Plot..... | 254 |
| B.14 plothist : Old-style Histogram..... | 263 |
| B.15 regquery : Queries the VO registry..... | 269 |
| B.16 server : Runs an HTTP server to perform STILTS commands..... | 271 |
| B.17 sqlclient : Executes SQL statements..... | 274 |
| B.18 sqlskymatch : Crossmatches table on sky position against SQL table..... | 276 |
| B.19 sqlupdate : Updates values in an SQL table..... | 281 |
| B.20 taplint : Tests TAP services..... | 283 |
| B.21 tapquery : Queries a Table Access Protocol server..... | 286 |
| B.22 tapresume : Resumes a previous query to a Table Access Protocol server..... | 290 |
| B.23 tapskymatch : Crossmatches table on sky position against TAP table | 293 |
| B.24 tcat : Concatenates multiple similar tables..... | 298 |
| B.25 tcatn : Concatenates multiple tables..... | 302 |
| B.26 tcopy : Converts between table formats..... | 305 |
| B.27 tcube : Calculates N-dimensional histograms..... | 307 |
| B.28 tloop : Generates a single-column table from a loop variable..... | 310 |
| B.29 tjoin : Joins multiple tables side-to-side..... | 312 |
| B.30 tmatch1 : Performs a crossmatch internal to a single table..... | 315 |
| B.31 tmatch2 : Crossmatches 2 tables using flexible criteria..... | 319 |
| B.32 tmatchn : Crossmatches multiple tables using flexible criteria..... | 325 |
| B.33 tmulti : Writes multiple tables to a single container file..... | 330 |
| B.34 tmultin : Writes multiple processed tables to single container file..... | 332 |
| B.35 tpipe : Performs pipeline processing on a table..... | 334 |
| B.36 tskymatch2 : Crossmatches 2 tables on sky position..... | 339 |
| B.37 votcopy : Transforms between VOTable encodings..... | 343 |
| B.38 votlint : Validates VOTable documents..... | 347 |
| Appendix C: Release Notes..... | 351 |
| C.1 Acknowledgements..... | 351 |
| C.2 Version History..... | 352 |

1 Introduction

STILTS provides a number of command-line applications which can be used for manipulating tabular data. Conceptually it sits between, and uses many of the same classes as, the packages STIL, which is a set of Java APIs providing table-related functionality, and TOPCAT, which is a graphical application providing the user with an interactive platform for exploring one or more tables. This document is mostly self-contained - it covers some of the same ground as the STIL and TOPCAT user documents (SUN/252 and SUN/253 respectively).

Currently, this package consists of commands in the following categories:

Generic table manipulation

`tcopy`, `tpipe`, `tmulti`, `tmultin`, `tcat`, `tcatn`, `tloop`, `tjoin` and `tcube` (see Section 6).

Crossmatching

`tmatch1`, `tmatch2`, `tmatchn` and `tskymatch2` (see Section 7).

Plotting

`plot2plane`, `plot2sky`, `plot2cube`, `plot2sphere` and `plot2time` (also deprecated old-style plot commands `plot2d`, `plot3d` and `plothist`) (see Section 8).

Sky Pixel Operations

`pixfoot` and `pixsample`.

VOTable

`votcopy` and `votlint`.

Virtual Observatory access

`cdsskymatch`, `coneskymatch`, `tapquery`, `tapresume`, `tapskymatch`, `taplint` and `regquery`.

SQL databases

`sqlclient`, `sqlupdate` and `sqlskymatch`.

Miscellaneous

`calc` (Appendix B.1), `funcs` (Appendix B.4) and `server` (Appendix B.16).

See Appendix A for an expanded version of this list.

There are many ways you might want to use these tools; here are a few possibilities:

In conjunction with TOPCAT

you can identify a set of processing steps using TOPCAT's interactive graphical facilities, and construct a script using the commands provided here which can perform the same steps on many similar tables without further user intervention.

Format conversion

If you have a separate table processing engine and you want to be able to output the results in a somewhat different form, for instance converting it from FITS to VOTable or from TABLEDATA-encoded to BINARY-encoded VOTable, or to perform some more scientifically substantial operation such as changing units or coordinate systems, substituting bad values etc, you can pass the results through one of the tools here. Since on the whole operation is streaming, such conversion can easily and efficiently be done on the fly.

Server-side operations

The tools provided here are suitable for use on servers, either to generate files as part of a web service (perhaps along the lines of the **Format conversion** item above) or as configurable components in a server-based workflow system. The `server` command may help, but is not required, for use in these situations.

Quick look

You might want to examine the metadata, or a few rows, or a statistical summary of a table

without having to load the whole thing into TOPCAT or some other table viewer application.

2 The `stilts` command

All the functions available in this package can be used from a single command, which is usually referred to in this document simply as "`stilts`". Depending on how you have installed the package, you may just type "`stilts`", or something like

```
java -jar some/path/stilts.jar
```

or

```
java -classpath topcat-lite.jar uk.ac.starlink.ttools.Stilts
```

or something else - this is covered in detail in Section 3.

In general, the form of a command is

```
stilts <stilts-flags> <task-name> <task-args>
```

The forms of the parts of this command are described in the following subsections, and details of each of the available tasks along with their arguments are listed in the command reference (Appendix B) at the end of this document. Some of the commands are highly configurable and have a variety of parameters to define their operation. In many cases however, it's not complicated to use them. For instance, to convert the data in a FITS table to VOTable format you might write:

```
stilts tcopy cat.fits cat.vot
```

2.1 Stilts flags

Some flags are common to all the tasks in the STILTS package, and these are specified after the `stilts` invocation itself and before the task name. They generally have the same effect regardless of which task is running. These generic flags are as follows:

-help

Prints a usage message for the `stilts` command itself and exits. The message contains a listing of all the known tasks.

-version

Prints the STILTS version number and exits.

-verbose

Causes more verbose information to be written during operation. Specifically, what this does is to boost the logging level by one notch. It may be specified multiple times to increase verbosity further.

-allowunused

Causes unused parameter settings on the command line to be tolerated. Normally, any unused parameters on the command line cause a usage message to be output and the command to fail, on the assumption that if you've supplied a parameter setting that's not doing anything it is probably a mistake and you should be given a chance to correct it. But if this flag is set, you just get a warning through the logging system about any unused parameters, and the command is executed as if they weren't there.

-prompt

Most of the STILTS commands have a number of parameters which will assume sensible defaults if you do not give them explicit values on the command line. If you use the `-prompt` flag, then you will be prompted for every parameter you have not explicitly specified to give you an opportunity to enter a value other than the default.

-bench

Outputs the elapsed time taken by the task to standard error on successful completion.

-debug

Sets up output suitable for debugging. The most visible consequence of this is that if an error occurs then a full stacktrace is output, rather than just a user-friendly report.

-batch

Some parameters will prompt you for their values, even if they offer legal defaults. If you use the `-batch` flag, then you won't be prompted at all.

-memory

Encourages the command to use java heap memory for caching large amounts of data rather than using temporary disk files. The default is to use memory for small tables, and disk for large ones. This flag is in most cases equivalent to specifying the system property `-Dstartable.storage=memory`.

-disk

Encourages the command to use temporary files on disk for caching table data. The default is to use memory for small tables, and disk for large ones. Using this flag may help if you are running out of memory. This flag is in most cases equivalent to specifying the system property `-Dstartable.storage=disk`.

-memgui

Displays a graphical window while the command is running which summarises used and available heap memory. May be useful for profiling or understanding resource constraints.

-checkversion <vers>

Requires that the version is exactly as given by the string `<vers>`. If it is not, STILTS will exit with an error. This can be useful when executing in certain controlled environments to ensure that the correct version of the application is being picked up.

-stdout <file>

Sends all normal output from the run to the given file. By default this goes to the standard output stream. Supplying an empty string or `"-"` for `<file>` will restore this default behaviour.

-stderr <file>

Sends all error output from the run to the given file. By default this goes to the standard error stream. Supplying an empty string or `"-"` for `<file>` will restore this default behaviour.

If you are submitting an error report, please include the result of running `stilts -version` and the output of the troublesome command with the `-debug` flag specified.

2.2 Task Names

The `<task-name>` part of the command line is the name of one of the tasks listed in Appendix B - currently the available tasks are:

- `calc`
- `cdsskymatch`
- `coneskymatch`
- `funcs`
- `pixfoot`
- `pixsample`
- `plot2cube`
- `plot2plane`
- `plot2sphere`
- `plot2sky`
- `plot2time`
- `plot2d`

- `plot3d`
- `plothist`
- `regquery`
- `server`
- `sqlclient`
- `sqlskymatch`
- `sqlupdate`
- `taplint`
- `tapquery`
- `tapresume`
- `tapskymatch`
- `tcat`
- `tcatn`
- `tcopy`
- `tcube`
- `tjoin`
- `tloop`
- `tmatch1`
- `tmatch2`
- `tmatchn`
- `tmulti`
- `tmultin`
- `tpipe`
- `tskymatch2`
- `votcopy`
- `votlint`

2.3 Task Arguments

The `<task-args>` part of the command line is a list of parameter assignments, each giving the value of one of the named parameters belonging to the task which is specified in the `<task-name>` part.

The general form of each parameter assignment is

```
<param-name>=<param-value>
```

If you want to set the parameter to the null value, which is legal for some but not all parameters, use the special string "null", or just leave the value blank ("`<param-name>=`"). In some cases you can optionally leave out the `<param-name>` part of the assignment (i.e. the parameter is positionally determined); this is indicated in the task's usage description if the parameter is described like [`<param-name>=`]`<param-value>` rather than `<param-name>=<param-value>`. If the `<param-value>` contains spaces or other special characters, then in most cases, such as from the Unix shell, you will have to quote it somehow. How this is done depends on your platform, but usually surrounding the whole value in single quotes will do the trick.

Tasks may have many parameters, and you don't have to set all of them explicitly on the command line. For a parameter which you don't set, two things can happen. In many cases, it will default to some sensible value. Sometimes however, you may be prompted for the value to use. In the latter case, a line like this will be written to the terminal:

```
matcher - Name of matching algorithm [sky]:
```

This is prompting you for the value of the parameter named `matcher`. "Name of matching algorithm" is a short description of what that parameter does. "sky" is the default value (if there is no default, no value will appear in square brackets). At this point you can do one of four things:

- Hit return - this will select the default value if there is one. If there is no default, this is equivalent to entering "null".
- Enter a value for the parameter explicitly. The special value "null" means the null value, which is legal for some, but not all parameters. If the value you enter is not legal, you will see an error message and you will be invited to try again.
- Enter "help" or a question mark "?". This will output a message giving a detailed description of the parameter and prompt you again.
- Bail out by hitting ctrl-C or whatever is usual on your platform.

Under normal circumstances, most parameters which have a legal default value will default to it if they are not set on the command line, and you will only be prompted for those where there is no default or the program thinks there's a good chance you might not want to use it. You can influence this however using flags to the `stilts` command itself (see Section 2.1). If you supply the `-prompt` flag, then you will be prompted for every parameter you have not explicitly set. If you supply `-batch` on the other hand, you won't be prompted for any parameters (and if you fail to set any without legal default values, the task will fail).

If you want to see the actual values of the parameters for a task as it runs, including prompted values and defaulted ones which you haven't specified explicitly, you can use the `-verbose` flag after the `stilts` command:

```
% stilts -verbose tcopy cat.fits cat.vot ifmt=fits
INFO: tcopy in=cat.fits out=cat.vot ifmt=fits ofmt=(auto)
```

If you make a parameter assignment on the command line for a parameter which is not used by the task in question, STILTS will issue an error message and the task will fail. Note some parameters are only used dependent on the presence or values of other parameters, so even supplying a parameter which is documented in the task's usage can have this effect. This is done on the assumption that if you have supplied a spurious parameter it's probably a mistake and you should be given the opportunity to correct it. But if you want to be free to make these mistakes without the task failing, you can supply the `-allowunused` flag as described in Section 2.1, in which case they will just result in a warning.

Extensive help is available from `stilts` itself about task and its parameters, as described in the next section.

2.4 Getting Help

As well as the command descriptions in this document (especially the reference section Appendix B) you can get help for STILTS usage from the command itself. Typing

```
stilts -help
```

results in this output:

```
Usage:
  stilts [-help] [-version] [-verbose] [-allowunused] [-prompt] [-bench]
        [-debug] [-batch] [-memory] [-disk] [-memgui]
        [-checkversion <vers>] [-stdout <file>] [-stderr <file>]
        <task-name> <task-args>
```

```
stilts <task-name> help[=<param-name>|*]
```

Known tasks:

```
calc
cdsskymatch
coneskymatch
funcs
pixfoot
pixsample
plot2d
```

```

plot3d
plothist
regquery
server
sqlclient
sqlskymatch
sqlupdate
taplint
tapquery
tapresume
tapskymatch
tcat
tcatn
tcopy
tcube
tjoin
tloop
tmatch1
tmatch2
tmatchn
tmulti
tmultin
tpipe
tskymatch2
votcopy
votlint
plot2plane
plot2sky
plot2cube
plot2sphere
plot2time

```

For help on the individual tasks, including their parameter lists, you can supply the word `help` after the task name, so for instance

```
stilts tcopy help
```

results in

```

Usage: tcopy ifmt=<in-format> ofmt=<out-format>
       [in=]<table> [out=]<out-table>

```

Finally, you can get help on any of the parameters of a task by writing `help=<param-name>`, like this:

```
stilts tcopy help=in
```

gives

```
Help for parameter IN in task TCOPY
```

```
-----
Name:
  in
```

```
Usage:
  [in=]<table>
```

```
Summary:
  Location of input table
```

```
Description:
  The location of the input table. This may take one of the following
  forms:

  * A filename.
  * A URL.
  * The special value "-", meaning standard input. In this case the
    input format must be given explicitly using the ifmt parameter.
    Note that not all formats can be streamed in this way.
```

```
* A system command line with either a "<" character at the start, or
  a "|" character at the end ("<syscmd" or "syscmd|"). This
  executes the given pipeline and reads from its standard output.
  This will probably only work on unix-like systems.
```

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

```
Type:
uk.ac.starlink.table.StarTable
```

If you use "*" instead of a parameter name in this usage, help for all the parameters will be printed. Note that in most shells you will probably need to quote the asterisk, so you should write

```
stilts tcopy help='*'
```

In some cases, as described in Section 2.3, you will be prompted for the value of a parameter with a line something like this:

```
matcher - Name of matching algorithm [sky]:
```

In this case, if you enter "help" or a question mark, then the parameter help entry will be printed to the screen, and the prompt will be repeated.

For more detailed descriptions of the tasks, which includes explanatory comments and examples as well as the information above, see the full task descriptions in the Command Reference (Appendix B).

3 Invocation

There are a number of ways of invoking the `stilts` command, depending on how you have installed the package. This section describes how to invoke it from the command line. An alternative, using it from Jython (the Java implementation of the Python language), is described in Section 4.

If you're using a Unix-like operating system, the easiest way is to use the `stilts` script. If you have a full `starjava` installation it is in the `starjava/bin` directory. Otherwise you can download it separately from wherever you got your STILTS installation in the first place, or find it at the top of the `stilts.jar` or `topcat-*.jar` that contains your STILTS installation, so do something like

```
unzip stilts.jar stilts
chmod +x stilts
```

to extract it (if you don't have `unzip`, try `jar xvf stilts.jar stilts`). `stilts` is a simple shell script which just invokes `java` with the right classpath and the supplied arguments.

To run using the `stilts` script, first make sure that both the `java` executable and the `stilts` script itself are on your path, and that the `stilts.jar` or `topcat-*.jar` jar file is in the same directory as `stilts`. Then the form of invocation is:

```
stilts <java-flags> <stilts-flags> <task-name> <task-args>
```

A simple example would be:

```
stilts votcopy format=binary t1.xml t2.xml
```

in this case, as often, there are no `<java-flags>` or `<stilts-flags>`. If you use the `-classpath` argument or have a `CLASSPATH` environment variable set, then classpath elements thus specified will be added to the classpath required to run the command. The examples in the command descriptions below use this form for convenience.

If you don't have a Unix-like shell available however, you will need to invoke Java directly with the appropriate classes on your classpath. If you have the file `stilts.jar`, in most cases you can just write:

```
java <java-flags> -jar stilts.jar <stilts-flags> <task-name> <task-args>
```

which in practice would look something like

```
java -jar /some/where/stilts.jar votcopy format=binary t1.xml t2.xml
```

In the most general case, Java's `-jar` flag might be no good, for one of the following reasons:

1. You have the classes in some form other than the `stilts.jar` file (such as `topcat-full.jar`)
2. You need to specify some extra classes on the classpath, which is required e.g. for use with JDBC (Section 3.4) or if you are extending the commands (Section 10.7.3) using your own classes at runtime

In this case, you will need an invocation of this form:

```
java <java-flags> -classpath <class-path>
    uk.ac.starlink.ttools.Stilts <stilts-flags> <task-name> <task-args>
```

The example above in this case would look something like:

```
java -classpath /some/where/topcat-full.jar uk.ac.starlink.ttools.Stilts
    votcopy format=binary t1.xml t2.xml
```

Finally, as a convenience, it is possible to run STILTS from a TOPCAT installation by using its `-stilts` flag, like this:

```
topcat <java-flags> -stilts <stilts-flags> <task-name> <task-args>
```

This is possible because TOPCAT is built on top of STILTS, so contains a superset of its code.

The `<stilts-flags>`, `<task-name>` and `<task-args>` parts of these invocations are explained in Section 2, and the `<class-path>` and `<java-flags>` parts are explained in the following subsections.

3.1 Class Path

The classpath is the list of places that Java looks to find the bits of compiled code that it uses to run an application. Depending on how you have done your installation the core STILTS classes could be in various places, but they are probably in a file with one of the names `stilts.jar`, `topcat-lite.jar` or `topcat-full.jar`. The full pathname of one of these files can therefore be used as your classpath. In some cases these files are self-contained and in some cases they reference other jar files in the filesystem - this means that they may or may not continue to work if you move them from their original location.

Under certain circumstances the tools might need additional classes, for instance:

- JDBC drivers (see Section 3.4)
- Providing extended algebraic functions (see Section 10.7.3)
- Installing I/O handlers for new table formats (see SUN/252)

In this case the classpath must contain a list of all the jar files in which the required classes can be found, separated by colons (unix) or semicolons (MS Windows). Note that even if all your jar files are in a single directory you can't use the name of that directory as a class path - you must name each jar file, separated by colons/semicolons.

3.2 Java Flags

In most cases it is not necessary to specify any additional arguments to the Java runtime, but it can be useful in certain circumstances. The two main kinds of options you might want to specify directly to Java are these:

System properties

System properties are a way of getting information into the Java runtime from the outside, rather like environment variables. There is a list of the ones which have significance to STILTS in Section 3.3. You can set them from the command line using a flag of the form `-Dname=value`. So for instance to ensure that temporary files are written to the `/home/scratch` directory, you could use the flag

```
-Djava.io.tmpdir=/home/scratch
```

Memory size

Java runs with a fixed amount of 'heap' memory; this is typically 64Mb by default. If one of the tools fails with a message that says it's out of memory then this has proved too small for the job in hand. You can increase the heap memory with the `-Xmx` flag. To set the heap memory size to 256 megabytes, use the flag

```
-Xmx256M
```

(don't forget the 'M' for megabyte). You will probably find performance is dreadful if you

specify a heap size larger than the physical memory of the machine you're running on.

You can specify other options to Java such as tuning and profiling flags etc, but if you want to do that sort of thing you probably don't need me to tell you about it.

If you are using the `stilts` command-line script, any flags to it starting `-D` or `-x` are passed directly to the `java` executable. You can pass other flags to Java with the `stilts` script's `-J` flag; for instance:

```
stilts -Xmx4M -J-verbose:gc calc 'mjdToIso(0)'
```

is equivalent to

```
java -Xmx4M -verbose:gc -jar stilts.jar calc 'mjdToIso(0)'
```

3.3 System Properties

System properties are a way of getting information into the Java runtime - they are a bit like environment variables. There are two ways to set them when using STILTS: either on the command line using arguments of the form `-Dname=value` (see Section 3.2) or in a file in your home directory named `.starjava.properties`, in the form of a `name=value` line. Thus submitting the flag

```
-Dvotable.strict=true
```

on the command line is equivalent to having the following in your `.starjava.properties` file:

```
# Force strict interpretation of the VOTable standard.
votable.strict=true
```

The following system properties have special significance to STILTS:

http.proxyHost

Can be used to force HTTP access to go via a named proxy; may be required if you are attempting access to remote data or services from behind a firewall configured to block direct HTTP connections. See java documentation for this property for more details.

java.awt.headless

May need to be set to "true" if running the plotting tasks on a headless server. You only need to worry about this if you see error messages complaining about headlessness.

java.io.tmpdir

The directory in which STILTS will write any temporary files it needs. This is usually only done if the `-disk` flag has been specified (see Section 2.1).

jdbc.drivers

Can be set to a (colon-separated) list of JDBC driver classes using which SQL databases can be accessed (see Section 3.4).

jel.classes

Can be set to a (colon-separated) list of classes containing static methods which define user-provided functions for synthetic columns or subsets. (see Section 10.7.3).

mark.workaround

If set to "true", this will work around a bug in the `mark()/reset()` methods of some java `InputStream` classes. These are rather common, including in Sun's J2SE system libraries. Use this if you are seeing errors that say something like "Resetting to invalid mark". Currently defaults to "false".

service.maxparallel

Raises the maximum number of concurrent queries that may be made during a multi-cone operation. You should only increase this value **with great care** since you risk overloading servers and becoming unpopular with data centres. As a rule, you should only increase this value if you have obtained permission from the data centres whose services on which you will be using the increased parallelism.

star.basicauth.user

star.basicauth.password

If set, these will provide username and password for HTTP Basic Authentication. Any time the application attempts to access an HTTP URL and is met by a 401 Unauthorized response, it will try again supplying these user credentials. This is a rather blunt instrument, since the same identity is supplied regardless of which URL is being accessed, but it may be of some use in accessing basic-authentication protected services.

startable.readers

Can be set to a (colon-separated) list of custom table format input handler classes (see SUN/252).

startable.storage

Can be set to determine the default storage policy. Setting it to "disk" has basically the same effect as supplying the "-disk" argument on the command line (see Section 2.1). Other possible values are "adaptive", "memory", "sideways" and "discard"; see SUN/252. The default is "adaptive", which means storing smaller tables in memory, and larger ones on disk.

startable.unmap

Determines whether and how unmapping of memory mapped buffers is done. Possible values are "sun" (the default) or "none". In most cases you are advised to leave this alone, but in the event of unmapping-related JVM crashes (not expected!), setting it to none may help.

startable.writers

Can be set to a (colon-separated) list of custom table format output handler classes (see SUN/252).

votable.namespacing

Determines how namespacing is handled in input VOTable documents. Known values are "none" (no namespacing, xmlns declarations in VOTable document will probably confuse parser), "lax" (anything that looks like it is probably a VOTable element will be treated as a VOTable element) and "strict" (VOTable elements must be properly declared in one of the correct VOTable namespaces). May also be set to the classname of a uk.ac.starlink.votable.Namespacing implementation. The default is "lax".

votable.strict

Set true for strict enforcement of the VOTable standard when parsing VOTables. This prevents the parser from working round certain common errors, such as missing arraysize attributes on FIELD or PARAM elements with datatype="char". False by default.

votable.version

Selects the version of the VOTable standard which output VOTables will conform to by default. May take the values "1.0", "1.1", "1.2" or "1.3". By default, version 1.2 VOTables are written.

3.4 JDBC Configuration

This section describes additional configuration which must be done to allow the commands to access SQL-compatible relational databases for reading or writing tables. If you don't need to talk to SQL-type databases, you can ignore the rest of this section. The steps described here are the standard ones for configuring JDBC (which sort-of stands for Java Database Connectivity), described in more detail on Sun's JDBC web page.

To use STILTS with SQL-compatible databases you must:

- Have access to an SQL-compatible database locally or over the network
- Have a JDBC driver appropriate for that database
- Install that driver for use with STILTS
- Know the format the driver uses for URLs to access database tables
- Have appropriate privileges on the database to perform the desired operations

Installing the driver consists of two steps:

1. Ensure that the classpath you are using includes this driver class as described in Section 3.1
2. Set the `jdbc.drivers` system property to the name of the driver class as described in Section 3.3

These steps are all standard for use of the JDBC system. See SUN/252 for information about JDBC drivers known to work with STIL (the short story is that at least MySQL and PostgreSQL will work).

Here is an example of using `tcopy` to write the results of an SQL query on a table in a MySQL database as a VOTable:

```
stilts -classpath /usr/local/jars/mysql-connector-java.jar \
-Djdbc.drivers=com.mysql.jdbc.Driver \
tcopy \
in="jdbc:mysql://localhost/db1#SELECT id, ra, dec FROM gsc WHERE mag < 9" \
ofmt=votable gsc.vot
```

or invoking Java directly:

```
java -classpath stilts.jar:/usr/local/jars/mysql-connect-java.jar \
-Djdbc.drivers=com.mysql.jdbc.Driver \
uk.ac.starlink.ttools.Stilts tcopy \
in="jdbc:mysql://localhost/db1#SELECT id, ra, dec FROM gsc WHERE mag < 9" \
ofmt=votable out=gsc.vot
```

You have to exercise some care to get the arguments in the right order here - see Section 3.

Alternatively, you can set some of this up beforehand to make the invocation easier. If you set your `CLASSPATH` environment variable to include the driver jar file (and the STILTS classes if you're invoking Java directly rather than using the scripts), and if you put the line

```
jdbc.drivers=com.mysql.jdbc.Driver
```

in the `.starjava.properties` file in your home directory, then you could avoid having to give the `-classpath` and `-Djdbc.drivers` flags respectively.

4 JyStilts - STILTS from Python

Most of the discussions and examples in this document describe using STILTS as a standalone java application from the command line; in this case, scripting can be achieved by executing one STILTS command, followed by another, followed by another, perhaps controlled from a shell script, with intermediate results stored in files.

However, it is also possible to invoke STILTS commands from within the Jython environment. Jython is a pure-java implementation of the widely-used Python scripting language. Using Jython is almost exactly the same as using the more usual C-based Python, except that it is not possible to use extensions which use C code. This means that if you are familiar with Python programming, it is very easy to string STILTS commands together in Jython.

This approach has several advantages over the conventional command-line usage:

- You can make use of python programming constructions like loops, functions and variables
- Python syntax can be used to put together parameter values (especially referencing quoted strings or values containing embedded spaces) in a way which is often less painful than doing it from the shell
- Intermediate processing stages can be kept in memory (in a python variable) rather than having to write them out to a file and read them in for the next command; this can be much more efficient
- Because of the previous point, there are separate read, filter, processing and write commands, which means command lines can be shorter and less confusing
- The java startup overhead (typically a couple of seconds) happens only once when entering jython, not once for every STILTS command

Note however that you will *not* be able to introduce JyStilts commands into your larger existing Python programs if those rely on C-based extensions, such as NumPy and SciPy, since JyStilts will only run in JPython, while C-based extensions will only run in CPython. (See however JNumeric for some of the Numpy functionality from Jython.)

Usage from jython has syntax which is similar to command-line STILTS, but with a few changes. The following functions are defined by JyStilts:

- A function `tread`, which reads a table from a file or URL and turns it into a table object in jython
- A table method `write` which takes a table object and writes it to file
- A table method for each STILTS filter (e.g. `cmd_head`, `cmd_select`, `cmd_addcol`)
- A table method for each STILTS output mode (e.g. `mode_out`, `mode_meta`, `mode_samp`),
- A function for each STILTS task (e.g. `tmatch2`, `tcat`, `plot2sky`)
- A number of table methods which make table objects integrate nicely into the python environment

Reasonably detailed documentation for these is provided in the usual Python way ("*doc strings*"), and can be accessed using the Python "`help`" command, however for full documentation and examples you should refer to this document.

In JyStilts the input, processing, filtering and output are done in separate steps, unlike in command-line STILTS where they all have to be combined into a single line. This can make the flow of execution easier to follow. A typical sequence will involve:

1. Reading one or more tables from file using the `tread` function
2. Perhaps filtering the input table(s) using one or more of the `cmd_*` filter methods
3. Performing core processing such as crossmatching
4. Perhaps filtering the result using one or more of the `cmd_*` filter methods
5. If running interactively, perhaps examining the intermediate results using one of the `mode_*`

output modes

6. Writing the final result to a file using the `write` method

Here is an example command line invocation for crossmatching two tables:

```
stilts tskymatch2 in1=survey.fits \
    icmd1='addskycoords fk4 fk5 RA1950 DEC1950 RA2000 DEC2000' \
    in2=mycat.csv ifmt2=csv \
    icmd2='select VMAG>18' \
    ra1=ALPHA dec1=DELTA ra2=RA2000 dec2=DEC2000 \
    error=10 join=2not1 \
    out=matched.fits
```

and here is what it might look like in JyStilts:

```
>>> import stilts
>>> t1 = stilts.tread('survey.fits')
>>> t1 = t1.cmd_addskycoords(t1, 'fk4', 'fk5', 'RA1950', 'DEC1950', 'RA2000', 'DEC2000')
>>> t2 = stilts.tread('mycat.csv', 'csv')
>>> t2 = t2.cmd_select('VMAG>18')
>>> tm = stilts.tskymatch2(in1=t1, in2=t2, ra1='ALPHA', dec1='DELTA',
...                        error=10, join='2not1')
>>> tm.write('matched.fits')
```

When running interactively, it can be convenient to examine the intermediate results before processing or writing as well, for instance:

```
>>> tm.mode_count()
columns: 19 rows: 2102
>>> tm.cmd_keeppcols('ID ALPHA DELTA').cmd_head(4).write()
+-----+-----+-----+
| ID      | ALPHA      | DELTA      |
+-----+-----+-----+
| 262     | 149.82439  | -0.11249   |
| 263     | 150.14438  | -0.11785   |
| 265     | 149.92944  | -0.11667   |
| 273     | 149.93185  | -0.12566   |
+-----+-----+-----+
```

More detail about how to run JyStilts and its usage is given in the following subsections.

4.1 Running JyStilts

The easiest way to run JyStilts is to download the standalone `jystilts.jar` file from the STILTS web page, and simply run

```
java -jar jystilts.jar
```

This file includes jython itself and all the STILTS and JyStilts classes. To use the JyStilts commands, you will need to import the `stilts` module using a line like `"import stilts"` from Jython in the usual Python way.

Alternatively, you can run JyStilts from an existing Jython installation using just the `stilts.jar` file. First, make sure that Jython is installed; it is available from <http://www.jython.org/>, and comes as a self-installing jar file. JyStilts has been tested, and appears to work, on versions 2.5.0 and 2.5.1; it's recommended to use the latest version if you don't have some reason to use one of the others. Some earlier versions of JyStilts worked with jython 2.2.1, but that no longer seems to be the case; it might be possible to reinstate this if there is some pressing need.

To use JyStilts, you then just need to start jython with the `stilts.jar` file on your classpath, for instance like this:

```
jython -J-classpath /some/where/stilts.jar
```

or (C-shell):

```
setenv CLASSPATH /some/where/stilts.jar
jython
```

Optionally, you can extract the `stilts.py` module from the `stilts.jar` file (using a command like "unzip `stilts.jar` `stilts.py`") and put it in a directory on your `jython sys.path` (e.g. `jythondir/Lib`); this may cause `jython` to compile it to bytecode (`stilts$py.class`) and thus improve startup time. Note that in this case you will still need the `stilts.jar` file on your classpath as above.

4.2 Table I/O

The `tread` function reads tables from an external location into `JyStilts`. Its arguments are as follows:

```
tread(location, fmt='(auto)', random=False)
```

and its return value is a table object, which can be interrogated directly, or used in other `JyStilts` commands. Usually, the `location` argument should be a string which gives the filename or URL at which a table can be found. You can alternatively use a readable python file (or file-like) object for the `location`, but be aware that this may be less efficient on memory. As with command-line `STILTS`, the `fmt` argument is one of the options in Section 5.2.1, but may be left as the default if the format is auto-detectable, which currently means if the file is in `VOTable`, `FITS`, `CDF` or `GBIN` format. The `random` argument can be used to ensure that the returned file has random (i.e. not sequential-only) access; for some table formats the default way of reading them in means that their rows can only be accessed in sequence. Depending on what processing you are doing, that may or may not be satisfactory.

Examples of reading a table are:

```
>>> import stilts
>>> t1 = stilts.tread('cat.fits')
>>> t2 = stilts.tread(open('cat.fits', 'rb'))          # less efficient
>>> t3 = stilts.tread('data.csv', fmt='csv', random=True)
```

The most straightforward way to write a table (presumably the result of one or a sequence of `JyStilts` commands) is using the `write` table method:

```
write(self, location=None, fmt='(auto)')
```

The `location` gives either a string which is a filename, or a writable python file (or file-like) object. Again, use of a filename is preferred as it may(?) be more efficient. If no `location` is supplied, the table will be written to standard output (useful for inspection, but a bad idea for binary formats or very large tables). The `fmt` argument is one of the output formats in Section 5.2.2, but may be left as the default if the format can be guessed from the filename.

Examples of writing a table are:

```
>>> table.write('out.fits')
>>> table.write(open('out.fits', 'wb'))              # less efficient?
>>> table.write('catalogue.dat', fmt='csv')
>>> table.write()                                    # display to stdout
```

Often it's convenient to combine examining the table with filtering steps, for instance:

```
>>> table.every(100).write()
```

would write only every hundredth row, and

```
>>> (table.cmd_sorthhead(10, 'BMAG')
...     .cmd_select('!NULL_VMAG')
...     .cmd_keeppcols('BMAG VMAG')
...     .write())
```

would write only the BMAG and VMAG columns for the ten rows in which VMAG is non-null with the lowest BMAG values.

You can also read and write multiple tables, if you use a table format for which that is appropriate. This generally means FITS (which can store tables in multiple extensions) or VOTable (which can store multiple TABLE elements in one document). This is done using the `treads` and `twrites` functions. The functions look like this:

```
treads(location, fmt='(auto)', random=False)
twrites(tables, location=None, fmt='(auto)')
```

These are similar to the `tread` and `twrite` functions, except that `treads` returns a list of tables rather than a single table, and `twrites`'s `tables` argument is an iterable over tables rather than a single table. Here is an example of reading multiple tables from a multi-extension FITS file, counting the rows in each, and then writing them out to a multi-TABLE VOTable file:

```
import stilts
tables = stilts.treads('multi.fits')
print([t.getRowCount() for t in tables])
stilts.twrites(tables, 'multi.vot', fmt='votable')
```

4.3 Table objects

The tables read by the `tread` function and produced by operating on them within JyStilts have a number of methods defined on them. These are explained below.

First, a number of special methods are defined which allow a table to behave in python like a sequence of rows:

`__iter__`

This special method means that the table can be treated as an *iterable*, so that for instance "`for row in table:`" will iterate over all rows.

`__len__` (*random-access tables only*)

This special method means that you can use the expression "`len(table)`" to count the number of rows. This method is not available for tables with sequential access only.

`__getitem__` (*random-access tables only*)

Returns a row at a given index in the table. This special method means that you can use indexing expressions like "`table[3]`" or "`table[0:10]`" to obtain the row or rows corresponding to a given row index or slice. This method is not available for tables with sequential access only.

`__add__`, `__mul__`, `__rmul__`

These special methods allow the addition and multiplication operators "+" and "*" to be used with the sense of concatenation. Thus "`table1+table2`" will produce a new table with the rows of `table1` followed by the rows of `table2`. Note this will only work if both tables have compatible columns. Similarly "`table*3`" would produce a table like `table` but with all its rows repeated three times.

In all of these cases, each row object that is accessed is a tuple of the column values for that row of the table. The tuple items (table cells) may be accessed using a key which is a numeric index or

slice in the usual way, or with a key which is a column name, or one of the `ColumnInfo` objects returned by `columns()`.

Sometimes, the result of a table operation will be a table which does not have random access. For such tables you can iterate over the rows, but not get their row values by indexing. Non-random-access tables are also peculiar in that `getRowCount` returns a negative value. To take a table which may not have random access and make it capable of random access, use the `random` filter: `"table=table.cmd_random()"`.

To a large extent it is possible to duplicate the functions of the various STILTS commands by writing your own python code based on these python-friendly table access methods. Note however that such python-based processing is likely to be *much* slower than the STILTS equivalents. If performance is important to you, you should try in most cases to use the various `cmd_*` commands etc for table processing.

Second, some additional utility methods are defined:

count_rows()

Returns the number of rows in the table in the most efficient way possible. If the table is random-access or otherwise knows its row count without further calculation, that value is returned. Otherwise, the rows are iterated over without reading, which may take some time but should be much more efficient than iterating over the table as an iterable, since the row cell data itself is not retrieved.

columns()

Returns a tuple of the column descriptors for the table. Each item in the tuple is an instance of the `ColumnInfo` class; useful methods include `getName()`, `getUnitString()`, `getUCD()`. `str(column)` will return its name.

coldata(key)

Returns a sequence of the values for the given column. The sequence will have the same number of elements as the number of rows in the table. The `key` argument may be either an integer column index (if negative, counts backwards from the end), or the column name or info object. The returned value will always be iterable (has `__iter__`), but will only be indexable (has `__len__` and `__getitem__`) if the table is random access.

parameters()

Returns a name to value mapping of the table parameters (per-table metadata). This does not include all the available information about those parameters, for instance unit and UCD information is not included. For more detailed information, use the `StarTable` methods. Note that as currently implemented, changing the values in the returned mapping will not change the actual table parameter values.

write(location=None, fmt=None)

Outputs the table. The optional `location` argument gives a filename or writable file object, and the optional `fmt` argument gives a format, one of the options listed in Section 5.2.1. If `location` is not supplied, output is to standard output, so in an interactive session it will be printed to the terminal. If `fmt` is not supplied, an attempt will be made to guess a suitable format based on the location.

Third, a set of `cmd_*` methods corresponding to the STILTS filters are available; these are described in Section 4.4.

Fourth, a set of `mode_*` methods corresponding to the STILTS output modes are available; these are described in Section 4.5.

Finally, tables are also instances of the `StarTable` interface defined by STIL, which is the table I/O

layer underlying STILTS. The full documentation can be found in the user manual and javadocs on the STIL page, and all the java methods can be used from JyStilts, but in most cases there are more pythonic equivalents provided, as described above.

Here are some examples of these methods in use:

```
>>> import stilts
>>> xsc = stilts.tread('/data/table/2mass_xsc.xml') # read table
>>> xsc.mode_count() # show rows/column count
columns: 6 rows: 1646844
>>> print xsc.columns() # full info on columns
(id(String), ra(Double)/degrees, dec(Double)/degrees, jmag(Double)/mag, hmag(Double)/mag, kmag(Double)/mag)
>>> print [str(col) for col in xsc.columns()] # column names only
['id', 'ra', 'dec', 'jmag', 'hmag', 'kmag']
>>> row = xsc[1000000] # examine millionth row
>>> print row
(u'19433000+4003190', 295.875, 40.055286, 14.449, 13.906, 13.374)
>>> print row[0] # cell by index
19433000+4003190
>>> print row['ra'], row['dec'] # cells by col name
295.875 40.055286
>>> print len(xsc) # count rows, maybe slow
1646844
>>> print xsc.count_rows() # count rows efficiently
1646844L
>>> print (xsc+xsc).count_rows() # concatenate
3293688L
>>> print (xsc*10000).count_rows()
16468440000L
>>> for row in xsc: # select rows using python commands
...     if row[4] - row[3] > 3.0:
...         print row[0]
...
11165243+2925509
20491597+5119089
04330238+0858101
01182715-1013248
11244075+5218078
>>> # same thing using stilts (50x faster)
>>> (xsc.cmd_select('hmag - jmag > 3.0')
...     .cmd_keepcols('id')
...     .write())
+-----+
| id |
+-----+
| 11165243+2925509 |
| 20491597+5119089 |
| 04330238+0858101 |
| 01182715-1013248 |
| 11244075+5218078 |
+-----+
```

The following are all ways to obtain the value of a given cell in the table from the previous example.

```
xsc.getCell(99, 0)
xsc[99][0]
xsc[99]['id']
xsc.coldata(0)[99]
xsc.coldata('id')[99]
```

Some of these methods may be more efficient than others. Note that none of these methods will work if the table has sequential-only access.

4.4 Table filter commands (cmd_*)

The STILTS table filters documented in Section 6.1 are available in JyStilts as table methods which start with the "cmd_" prefix. The return value when calling the method on a table object is another

table object. The arguments, which are the same as those required for the command-line version, are supplied as a list of unnamed arguments of the `cmd_*` function. In general the arguments are strings, but numbers are accepted where appropriate. Use the python `help` command to see the usage of each method.

So, to use the `tail` filter to select only the last ten lines of a table, you can write:

```
table.cmd_tail(10)
```

To set units of "Hz" for some columns using the `colmeta` filter write:

```
table.cmd_colmeta('-units', 'Hz', 'AFREQ BFREQ CFREQ')
```

Note that where a filter argument is a space-separated list it must appear as a single argument in the filter invocation, just as in command-line STILTS.

The filter commands are also available as module functions. This means that

```
stilts.cmd_head(table, 10)
```

and

```
table.cmd_head(10)
```

have exactly the same meaning. It's a matter of taste which you prefer.

4.5 Table output modes (`mode_*`)

The STILTS table output modes documented in Section 6.4 are available in JyStilts as table methods which start with the "`mode_`" prefix. These methods have no return value, but cause something to happen, in some cases output to be written to standard output. Some of these methods have named arguments, others have no arguments. Use the python `help` command to see the usage of each method.

These methods are straightforward to use. The following example calculates statistics for a table and writes the results to standard output:

```
>>> table.mode_stats()
```

and this one attempts to send the table via the SAMP communications protocol to a running instance of TOPCAT:

```
>>> table.mode_samp(client='topcat')
```

The output modes are also available as module functions. This means that

```
stilts.mode_samp(table, client='topcat')
```

and

```
table.mode_samp(client='topcat')
```

have exactly the same meaning. It's a matter of taste which you prefer.

4.6 Tasks

The STILTS tasks documented in Appendix B can be used under their usual names if they are imported from the `stilts` module. STILTS parameters as are supplied as named arguments of the python functions. In general they are either table objects for table input parameters or strings, but in

some cases python arrays are accepted, and numbers may be used where appropriate. The STILTS input format (`ifmt`, `istream`), filter (`cmd/icmd/ocmd`) and output mode (`omode`) parameters are not used however; instead perform filtering directly on the table inputs and outputs using the python `cmd_*` and `mode_*` table methods or functions.

Here is an example of concatenating two similar tables together and writing the result:

```
>>> from stilts import tread, tcat
>>> t1 = tread('data1.csv', fmt='csv')
>>> t2 = tread('data2.csv', fmt='csv')
>>> t12 = tcat([t1,t2], seqcol='seq')
>>> t12.write('t12.csv', fmt='csv')
```

Note that for those tasks which have a parameter named "in" in command-line STILTS, it has been renamed as "in_" for the python version, to avoid a name clash with the python reserved word. In most cases, the `in` parameter is the first, mandatory parameter in any case, and so can be referenced by position as in the previous example (we could have written `tcat(in_[t1,t2])` instead).

4.7 Calculation Functions

The various functions from the expression language listed in Section 10.5 are available directly from JyStilts. Each of the subsections in that section is a class in the `stilts` module namespace, with unbound functions representing the functions.

This means you can use them like this:

```
>>> import stilts
>>> print stilts.Times.mjdToIso(54292)
2007-07-11T00:00:00
```

or like this:

```
>>> from stilts import CoordsDegrees
>>> dist = CoordsDegrees.skyDistanceDegrees(ra1, dec1, ra2, dec2)
```

5 Table I/O

Most of the tools in this package either read one or more tables as input, or write one or more tables as output, or both. This section explains what kind of tables the tools can read and write, and how you tell them where to find the tables to operate on.

In most cases input and output table specifications are given by parameters with the following names (or similar ones):

| | |
|-------------|------------------------------|
| in | Location of the input table |
| ifmt | Format of the input table |
| out | Location of the output table |
| ofmt | Format of the output table |

The values of these parameters are discussed in more detail below.

5.1 Table Locations

The location of tables for input and output are usually given using the `in` and `out` parameters respectively. These are often, but not always, filenames. The possibilities are these:

Filename

Very often, you will simply specify a filename as location, and the tool will just read from/write to it in the usual way.

URL

Tables can be read from URLs directly, and in some cases written to them as well. Some non-standard URL protocols are supported as well as the usual ones. The list is:

http:
Read from HTTP resources.

ftp:
Read from anonymous FTP resources.

file:
Read from local files. This is not particularly useful since you can do much the same using just the filename. There is a difference: using this form forces reads to be sequential rather than random access, which may allow you to experience a different set of different performance characteristics and bugs.

jar:
Specialised protocol for looking inside Java Archive files - see `JarURLConnection` documentation.

myspace:
Accesses files in the AstroGrid "MySpace" virtual file store. These URLs look something like `"myspace:/survey/iras_psc.xml"`, and can access files in the myspace are that the user is currently logged into. These URLs can be used for both input and output of tables. To use them you must have an AstroGrid account and the AstroGrid WorkBench or similar must be running; if you're not currently logged in a dialogue will pop up to ask you for name and password.

ivo:

Understands ivo-type URLs which signify files in the AstroGrid "MySpace" virtual file store. These URLs look something like "ivo://uk.ac.le.star/filemanager#node-2583". These URLs can be used for both input and output of tables. To use them you must have an AstroGrid account and the AstroGrid WorkBench or similar must be running; if you're not currently logged in a dialogue will pop up to ask you for name and password.

jdbc:

Used for communicating with SQL-compliant relational databases. These are a bit different to normal URLs - see section Section 3.4.

Minus sign ("-")

The special location "-" (minus sign) indicates standard input (for reading) or standard output (for writing). This allows you to use STILTS commands in a normal Unix pipeline.

System command ("*syscmd*" or "*syscmd* |")

If the location starts with a "<" character or ends with a "|" character, the rest of the string is taken as a command line to be executed by the system shell. For instance a location like "<cat header.txt data.txt" (or equivalently "cat header.txt data.txt|") could be used to prepend a header line to an ASCII data file before it is passed to the STILTS ASCII-format input handler. Note this syntax will probably only work on Unix-like systems.

In any of these cases, for input locations compression is taken care of automatically. That means that you can give the filename or URL of a file which is compressed using `gzip`, `bzip2` or Unix `compress` and the program will uncompress it on the fly.

5.2 Table Formats

The generic table commands in STILTS (currently `tpipe`, `tcopy`, `tmulti`, `tmultin`, `tcat`, `tcatn`, `tloop`, `tjoin`, `tcube`, `tmatch1`, `tmatch2`, `tmatchn`, `tskymatch2`, `pixfoot`, `pixsample`, `plot2cube`, `plot2plane`, `plot2sky`, `plot2sphere`, `plot2time`, `plot2d`, `plot3d`, `plothist`, `cdsskymatch`, `coneskymatch`, `sqlskymatch`, `tapquery`, `tapresume`, `tapskymatch` and `regquery`) have no native format for table storage, they can process data in a number of formats equally well. STIL has its own model of what a table consists of, which is basically:

- Some per-table metadata (parameters)
- A number of columns
- Some per-column metadata
- A number of rows, each containing one entry per column

Some table formats have better facilities for storing this sort of thing than others, and when performing conversions STILTS does its best to translate between them, but it can't perform the impossible: for instance there is nowhere in a Comma-Separated Values file to store descriptions of column units, so these will be lost when converting from VOTable to CSV formats.

The formats the package knows about are dependent on the input and output handlers currently installed. The ones installed by default are listed in the following subsections. More may be added in the future, and it is possible to install new ones at runtime - see the STIL documentation for details.

Some formats can be used to hold multiple tables in a single file, and others can only hold a single table per file.

5.2.1 Input Formats

Some of the tools in this package ask you to specify the format of input tables using the `ifmt`

parameter. The following list gives the values usually allowed for this (matching is case-insensitive):

fits

FITS format - FITS binary or ASCII tables can be read. For commands which take a single input table, by default the first table HDU in the file will be used, but this can be altered for multi-extension FITS files by supplying an identifier after a '#' sign. The identifier can be either an HDU index or the extension name (EXTNAME header, possibly followed by "-" and the EXTVER header), so "table.fits#3" means the third HDU extension, and "table.fits#UV_DATA" means the HDU with the value "UV_DATA" for its EXTNAME header card.

colfits

Column-oriented FITS format. This is where a table is stored as a BINTABLE extension which contains a single row, each cell of the row containing a whole column of the table it represents. This has different performance characteristics from normal FITS tables; in particular it may be considerably more efficient for very large, and especially very wide tables where not all of the columns are required at any one time. Only likely to be efficient for uncompressed files on disk.

votable

VOTable format - any legal version 1.0, 1.1, 1.2 or 1.3 format VOTable documents, and many illegal ones, can be read. For commands which take a single input table, by default the first TABLE element in the document is used, but this can be altered by supplying the 0-based index after a '#' sign, so "table.xml#4" means the fifth TABLE element in the document.

cdf

NASA Common Data Format. CDF is described at <http://cdf.gsfc.nasa.gov/>.

ascii

Plain text file with one row per column in which columns are separated by whitespace.

csv

Comma-Separated Values format, using approximately the conventions used by MS Excel.

gbin

Special-interest GBIN format for internal use by the DPAC consortium in relation to the Gaia astrometry satellite. Additional classes (data model and GaiaTools GBIN reader) are required on the classpath at runtime to use this format (e.g. `stilts -classpath MDBExplorerStandalone.jar` or `java -classpath stilts.jar:MDBExplorerStandalone.jar uk.ac.starlink.ttools.Stilts`).

tst

Tab-Separated Table format, as used by Starlink's GAIA and ESO's SkyCat amongst other tools.

ipac

IPAC Table Format.

wdc

World Datacentre Format (experimental).

For more details on these formats, see the descriptions in SUN/253.

In some cases (when using VOTable, FITS, CDF or GBIN format tables) the tools can detect the table format automatically, and no explicit specification is necessary. If this isn't the case and you omit the format specification, the tool will fail with a suitable error message. It is always safe to specify the format explicitly; this will be slightly more efficient, and may lead to more helpful error messages in the case that the table can't be read correctly.

5.2.2 Output Formats

Some of the tools ask you to specify the format of output tables using the `ofmt` parameter. The following list gives the values usually allowed for this; in some cases as you can see there are several variants of a given format. You can abbreviate these names, and the first match in the list below will be used, so for instance specifying `votable` is equivalent to specifying `votable-tabledata` and `fits` is equivalent to `fits-plus`. Matching is case-insensitive.

fits-plus

FITS file; primary HDU contains a VOTable representation of the metadata, subsequent extensions contain one or more FITS binary tables (behaves the same as `fits-basic` for most purposes)

fits-basic

FITS file; primary HDU is data-less, subsequent extensions contain a FITS binary table

colfits-plus

FITS file containing a BINTABLE with a single row; each cell of the row contains a whole column's worth of data. The primary HDU also contains a VOTable representation of the metadata.

colfits-basic

FITS file containing a BINTABLE with a single row; each cell of the row contains a whole column's worth of data. The primary HDU contains nothing.

votable-tabledata

VOTable document with TABLEDATA (pure XML) encoding

votable-binary-inline

VOTable document with BINARY-encoded data inline within a `STREAM` element. If VOTable 1.3 output is in force (see `votable.version` system property), `votable-binary2-inline` is provided instead.

votable-binary-href

VOTable document with BINARY-encoded data in a separate file (only if not writing to a stream). If VOTable 1.3 output is in force (see `votable.version` system property), `votable-binary2-href` is provided instead.

votable-fits-href

VOTable document with FITS-encoded data in a separate file (only if not writing to a stream)

votable-fits-inline

VOTable document with FITS-encoded data inline within a `STREAM` element

ascii

Simple space-separated ASCII file format

text

Human-readable plain text (with headers and column boundaries marked out)

csv

Comma-Separated Value format. The first line is a header which contains the column names.

csv-noheader

Comma-Separated Value format with no header line.

ipac

IPAC Table Format.

tst

Tab-Separated Table format.

html

Standalone HTML document containing a `TABLE` element

html-element

HTML `TABLE` element

latex

LaTeX `tabular` environment

latex-document

LaTeX standalone document containing a `tabular` environment

mirage

Mirage input format

For more details on these formats, see the descriptions in SUN/253.

In some cases the tools may guess what output format you want by looking at the extension of the output filename you have specified.

6 Table Pipelines

Several of the tasks available in STILTS take one or more input tables, do something or other with them, and produce one or more output tables. This is a pretty obvious way to go about things, and in the most straightforward case that's exactly what happens: you name one or more input tables, specify the processing parameters, and name an output table; the task then reads the input tables from disk, does the processing and writes the output table to disk.

However, many of the tasks in STILTS allow you to do pre-processing of the input tables before the main job, post-processing of the output table after the main job, and to decide what happens to the final tabular result, without any intermediate storage of the data. Examples of the kind of pre-processing you might want to do are to rearrange the columns so that they have the right units for the main task, or replace 'magic' values such as -999 with genuine blank values; the kind of post-processing you might want to do is to sort the rows in the output table or delete some of the columns you're not interested in. As for the destination of the final table, you might want to write it to disk, but equally you might not want to store it anywhere, but only be interested in counting the number of rows, or seeing the minima/maxima of a few of the columns, or you might want to send it straight to TOPCAT or some other table viewing application for interactive analysis.

Clearly, you could achieve the same effect by running multiple applications: preprocess your original input tables to write intermediate files on disk, run the main processing application which reads those files from disk and writes a new output file, run another application to postprocess the output file and write a new final output file, and finally do something with this such as counting the rows in it or viewing it in TOPCAT. However, by doing it all within a single task instead, no intermediate results have to be stored, and the whole sequence can be very much more efficient. You can think of this (if it helps) like a Unix pipeline, except what is being streamed from the start to the end of the pipe is not bytes, but table metadata and data. In most cases, the table data is streamed through the pipeline a row at a time, meaning that the amount of memory required is small (though in some cases, for instance row sorting and crossmatching, this is not possible).

Tasks which allow this pre/post-processing, or "filtering", have parameters with names like "cmd" which you use to specify processing steps. Tasks with multiple input tables (`tmatch2`, `tskymatch2`, `tcatsn`, `tjoin`) may have parameters named `icmd1`, `icmd2`, ... for preprocessing the different input tables and `ocmd` for postprocessing the output table. `tpipe` does nothing except filtering, so there is no distinction between pre- and post-processing, and its filter parameter is just named `cmd`. `tpipe` additionally has a `script` parameter which allows you to use a text file to write the commands in, to prevent the command line getting too long. In both cases there is a parameter named `omode` which defines the "output mode", that is, what happens to the post-processed output table that comes out of the end of the pipeline.

Section 6.1 lists the processing steps available, and explains how to use them, Section 6.2 and Section 6.3 describe the syntax used in some of these filter commands for specifying columns, and Section 6.4 describes the available output modes. See the examples in the command reference, and particularly the `tpipe` examples (Appendix B.35.2), for some examples putting all this together.

6.1 Processing Filters

This section lists the filter commands which can be used for table pipeline processing, in conjunction with `cmd-` or `script-` type parameters.

You can string as many of these together as you like. On the command line, you can repeat the `cmd` (or `icmd1`, or `ocmd...`) parameter multiple times, or use one `cmd` parameter and separate different filter specifiers with semicolons (`;`). The effect is the same.

It's important to note that each command in the sequence of processing steps acts on the table at that

point in the sequence. Thus either of the two identical invocations:

```
stilts tpipe cmd='delcols 1; delcols 1; delcols 1'
stilts tpipe cmd='delcols 1' cmd='delcols 1' cmd='delcols 1'
```

has the same effect as

```
stilts tpipe cmd='delcols "1 2 3"'
```

since in the first case the columns are shifted left after each one is deleted, so the table seen by each step has one fewer column than the one before. Note also the use of quotes in the latter of the examples above, which is necessary so that the `<colid-list>` of the `delcols` command is interpreted as one argument not three separate words.

The available filters are described in the following subsections.

6.1.1 addcol

Usage:

```
addcol [-after <col-id> | -before <col-id>]
        [-units <units>] [-ucd <ucd>] [-utype <utype>] [-desc <descrip>]
        <col-name> <expr>
```

Add a new column called `<col-name>` defined by the algebraic expression `<expr>`. By default the new column appears after the last column of the table, but you can position it either before or after a specified column using the `-before` or `-after` flags respectively. The `-units`, `-ucd` `-utype` and `-desc` flags can be used to define metadata values for the new column.

Syntax for the `<expr>` and `<col-id>` arguments is described in the manual.

6.1.2 addpixsample

Usage:

```
addpixsample [-radius <expr-rad>] [-systems <in-sys> <pix-sys>]
              <expr-lon> <expr-lat> <healpix-file>
```

Samples pixel data from an all-sky image file in HEALPix format. The `<healpix-file>` argument must be the filename of a table containing HEALPix pixel data. The URL of such a file can be used instead, but local files are likely to be more efficient.

The `<expr-lon>` and `<expr-lat>` arguments give expressions for the longitude and latitude in degrees for each row of the input table; this is usually just the column names. The long/lat must usually be in the same coordinate system as that used for the HEALPix data, so if the one is in galactic coordinates the other must be as well. If this is not the case, use the `-systems` flag to give the input long/lat and healpix data coordinate system names respectively. The available coordinate system names are:

- `icrs`: ICRS (Hipparcos) (Right Ascension, Declination)
- `fk5`: FK5 J2000.0 (Right Ascension, Declination)
- `fk4`: FK4 B1950.0 (Right Ascension, Declination)
- `galactic`: IAU 1958 Galactic (Longitude, Latitude)
- `supergalactic`: de Vaucouleurs Supergalactic (Longitude, Latitude)
- `ecliptic`: Ecliptic (Longitude, Latitude)

The `<expr-rad>`, if present, is a constant or expression giving the radius in degrees over which pixels will be averaged to obtain the result values. Note that this averaging is somewhat

approximate; pixels partly covered by the specified disc are weighted the same as those fully covered. If no radius is specified, the value of the pixel covering the central position will be used.

The `<healpix-file>` file is a table with one row per HEALPix pixel and one or more columns representing pixel data. A new column will be added to the output table corresponding to each of these pixel columns. This type of data is available in FITS tables for a number of all-sky data sets, particularly from the LAMBDA (<http://lambda.gsfc.nasa.gov/>) archive; see for instance the page on foreground products (including dust emission, reddening etc) or WMAP 7 year data. If the filename given does not appear to point to a file of the appropriate format, an error will result. Note the LAMBDA files mostly (all?) use galactic coordinates, so coordinate conversion using the `-systems` flag may be appropriate, see above.

Syntax for the `<expr-lon>`, `<expr-lat>` and `<expr-rad>` arguments is described in the manual.

This filter is somewhat experimental, and its usage may be changed or replaced in a future version.

Note: you may prefer to use the `pixsample` command instead.

6.1.3 addresolve

Usage:

```
addresolve <col-id-objname> <col-name-ra> <col-name-dec>
```

Performs name resolution on the string-valued column `<col-id-objname>` and appends two new columns `<col-name-ra>` and `<col-name-dec>` containing the resolved Right Ascension and Declination in degrees.

Syntax for the `<col-id-objname>` argument is described in Section 6.2.

UCDs are added to the new columns in a way which tries to be consistent with any UCDs already existing in the table.

Since this filter works by interrogating a remote service, it will obviously be slow. The current implementation is experimental; it may be replaced in a future release by some way of doing the same thing (perhaps a new STILTS task) which is able to work more efficiently by dispatching multiple concurrent requests.

This is currently implemented using the Simbad service operated by CDS.

6.1.4 addskycoords

Usage:

```
addskycoords [-epoch <expr>] [-inunit deg|rad|sex] [-outunit deg|rad|sex]
              <insys> <outsys> <col-id1> <col-id2> <col-name1> <col-name2>
```

Add new columns to the table representing position on the sky. The values are determined by converting a sky position whose coordinates are contained in existing columns. The `<col-id>` arguments give identifiers for the two input coordinate columns in the coordinate system named by `<insys>`, and the `<col-name>` arguments name the two new columns, which will be in the coordinate system named by `<outsys>`. The `<insys>` and `<outsys>` coordinate system specifiers are one of

- `icrs`: ICRS (Hipparcos) (Right Ascension, Declination)
- `fk5`: FK5 J2000.0 (Right Ascension, Declination)
- `fk4`: FK4 B1950.0 (Right Ascension, Declination)

- `galactic`: IAU 1958 Galactic (Longitude, Latitude)
- `supergalactic`: de Vaucouleurs Supergalactic (Longitude, Latitude)
- `ecliptic`: Ecliptic (Longitude, Latitude)

The `-inunit` and `-outunit` flags may be used to indicate the units of the existing coordinates and the units for the new coordinates respectively; use one of `degrees`, `radians` or `sexagesimal` (may be abbreviated), otherwise `degrees` will be assumed. For `sexagesimal`, the two corresponding columns must be string-valued in forms like `hh:mm:ss.s` and `dd:mm:ss.s` respectively.

For certain conversions, the value specified by the `-epoch` flag is of significance. Where significant its value defaults to 2000.0.

Syntax for the `<expr>`, `<col-id1>` and `<col-id2>` arguments is described in the manual.

6.1.5 `assert`

Usage:

```
assert <expr>
```

Check that a boolean expression is true for each row. If the expression `<expr>` does not evaluate true for any row of the table, execution terminates with an error. As long as no error occurs, the output table is identical to the input one.

The exception generated by an assertion violation is of class `uk.ac.starlink.ttools.filter.AssertException` although that is not usually obvious if you are running from the shell in the usual way.

Syntax for the `<expr>` argument is described in Section 10.

6.1.6 `badval`

Usage:

```
badval <bad-val> <colid-list>
```

For each column specified in `<colid-list>` any occurrence of the value `<bad-val>` is replaced by a blank entry.

Syntax for the `<colid-list>` argument is described in Section 6.3.

6.1.7 `cache`

Usage:

```
cache
```

Stores in memory or on disk a temporary copy of the table at this point in the pipeline. This can provide improvements in efficiency if there is an expensive step upstream and a step which requires more than one read of the data downstream. If you see an error like "Can't re-read data from stream" then adding this step near the start of the filters might help.

The result of this filter is guaranteed to be random-access.

See also the `random` filter, which caches only when the input table is not random-access.

6.1.8 check

Usage:

```
check
```

Runs checks on the table at the indicated point in the processing pipeline. This is strictly a debugging measure, and may be time-consuming for large tables.

6.1.9 clearparams

Usage:

```
clearparams <pname> ...
```

Clears the value of one or more named parameters. Each of the <pname> values supplied may be either a parameter name or a simple wildcard expression matching parameter names. Currently the only wildcarding is a "*" to match any sequence of characters. `clearparams *` will clear all the parameters in the table.

It is not an error to supply <pname>s which do not exist in the table - these have no effect.

6.1.10 colmeta

Usage:

```
colmeta [-name <name>] [-units <units>] [-ucd <ucd>] [-utype <utype>]  
        [-desc <descrip>]  
        <colid-list>
```

Modifies the metadata of one or more columns. Some or all of the name, units, ucd, utype and description of the column(s), identified by <colid-list> can be set by using some or all of the listed flags. Typically, <colid-list> will simply be the name of a single column.

Syntax for the <colid-list> argument is described in Section 6.3.

6.1.11 delcols

Usage:

```
delcols <colid-list>
```

Delete the specified columns. The same column may harmlessly be specified more than once.

Syntax for the <colid-list> argument is described in Section 6.3.

6.1.12 every

Usage:

```
every <step>
```

Include only every <step>'th row in the result, starting with the first row.

6.1.13 explodeall

Usage:

```
explodeall [-ifndim <ndim>] [-ifshape <dims>]
```

Replaces any columns which is an N-element arrays with N scalar columns. Only columns with fixed array sizes are affected. The action can be restricted to only columns of a certain shape using the flags.

If the `-ifndim` flag is used, then only columns of dimensionality `<ndim>` will be exploded. `<ndim>` may be 1, 2,

If the `-ifshape` flag is used, then only columns with a specific shape will be exploded; `<dims>` is a space- or comma-separated list of dimension extents, with the most rapidly-varying first, e.g. '2 5' to explode all 2 x 5 element array columns.

6.1.14 explodecols**Usage:**

```
explodecols <colid-list>
```

Takes a list of specified columns which represent N-element arrays and replaces each one with N scalar columns. Each of the columns specified by `<colid-list>` must have a fixed-length array type, though not all the arrays need to have the same number of elements.

Syntax for the `<colid-list>` argument is described in Section 6.3.

6.1.15 fixcolnames**Usage:**

```
fixcolnames
```

Renames all columns and parameters in the input table so that they have names which have convenient syntax for STILTS. For the most part this means replacing spaces and other non-alphanumeric characters with underscores. This is a convenience which lets you use column names in algebraic expressions and other STILTS syntax.

6.1.16 head**Usage:**

```
head <nrows>
```

Include only the first `<nrows>` rows of the table. If the table has fewer than `<nrows>` rows then it will be unchanged.

6.1.17 keepcols**Usage:**

```
keepcols <colid-list>
```

Select the columns from the input table which will be included in the output table. The output table will include only those columns listed in `<colid-list>`, in that order. The same column may be listed more than once, in which case it will appear in the output table more than once.

Syntax for the `<colid-list>` argument is described in Section 6.3.

6.1.18 meta

Usage:

```
meta [<item> ...]
```

Provides information about the metadata for each column. This filter turns the table sideways, so that each row of the output corresponds to a column of the input. The columns of the output table contain metadata items such as column name, units, UCD etc corresponding to each column of the input table.

By default the output table contains columns for the following items:

- Index: Position of column in table
- Name: Column name
- Class: Data type of objects in column
- Shape: Shape of array values
- ElSize: Size of each element in column (mostly useful for strings)
- Units: Unit string
- Description: Description of data in the column
- UCD: Unified Content Descriptor
- Utype: Type in data model

as well as any table-specific column metadata items that the table contains.

However, the output may be customised by supplying one or more `<item>` headings. These may be selected from the above as well as the following:

- UCD_desc: Textual description of UCD

as well as any table-specific metadata. It is not an error to specify an item for which no metadata exists in any of the columns (such entries will result in empty columns).

Any table parameters of the input table are propagated to the output one.

6.1.19 progress

Usage:

```
progress
```

Monitors progress by displaying the number of rows processed so far on the terminal (standard error). This number is updated every second or thereabouts; if all the processing is done in under a second you may not see any output. If the total number of rows in the table is known, an ASCII-art progress bar is updated, otherwise just the number of rows seen so far is written.

Note under some circumstances progress may appear to complete before the actual work of the task is done since part of the processing involves slurping up the whole table to provide random access on it. In this case, applying the `cache` upstream may help.

6.1.20 random

Usage:

```
random
```

Ensures that random access is available on this table. If the table currently has random access, it has no effect. If only sequential access is available, the table is cached so that downstream steps will see the cached, hence random-access, copy.

6.1.21 randomview

Usage:

```
randomview
```

Ensures that steps downstream only use random access methods for table access. If the table is sequential only, this will result in an error. Only useful for debugging.

6.1.22 repeat

Usage:

```
repeat [-row|-table] <count>
```

Repeats the rows of a table multiple times to produce a longer table. The output table will have `<count>` times as many rows as the input table.

The optional flag determines the sequence of the output rows. If `<count>=2` and there are three rows, the output sequence will be 112233 for `-row` and 123123 for `-table`. The default behaviour is currently `-table`.

The `<count>` value will usually be a constant integer value, but it can be an expression evaluated in the context of the table.

6.1.23 replacecol

Usage:

```
replacecol [-name <name>] [-units <units>] [-ucd <ucd>] [-utype <utype>]
           [-desc <descrip>]
           <col-id> <expr>
```

Replaces the content of a column with the value of an algebraic expression. The old values are discarded in favour of the result of evaluating `<expr>`. You can specify the metadata for the new column using the `-name`, `-units`, `-ucd`, `-utype` and `-desc` flags; for any of these items which you do not specify, they will take the values from the column being replaced.

It is legal to reference the replaced column in the expression, so for example `"replacecol pixsize pixsize*2"` just multiplies the values in column `pixsize` by 2.

Syntax for the `<col-id>` and `<expr>` arguments is described in the manual.

6.1.24 replaceval

Usage:

```
replaceval <old-val> <new-val> <colid-list>
```

For each column specified in `<colid-list>` any instance of `<old-val>` is replaced by `<new-val>`. The value string 'null' can be used for either `<old-value>` or `<new-value>` to indicate a blank value (but see also the `badval` filter).

Syntax for the `<colid-list>` argument is described in Section 6.3.

6.1.25 rowrange

Usage:

```
rowrange <first> <last>|+<count>
```

Includes only rows in a given range. The range can either be supplied as "`<first> <last>`", where row indices are inclusive, or "`<first> +<count>`". In either case, the first row is numbered 1.

Thus, to get the first hundred rows, use either "`rowrange 1 100`" or "`rowrange 1 +100`" and to get the second hundred, either "`rowrange 101 200`" or "`rowrange 101 +100`".

6.1.26 select

Usage:

```
select <expr>
```

Include in the output table only rows for which the expression `<expr>` evaluates to true. `<expr>` must be an expression which evaluates to a boolean value (true/false).

Syntax for the `<expr>` argument is described in Section 10.

6.1.27 seqview

Usage:

```
seqview
```

Ensures that steps downstream see the table as sequential access. Any attempts at random access will fail. Only useful for debugging.

6.1.28 setparam

Usage:

```
setparam [-type byte|short|int|long|float|double|boolean|string]
          [-desc <descrip>] [-unit <units>] [-ucd <ucd>] [-utype <utype>]
          <pname> <pexpr>
```

Sets a named parameter in the table to a given value. The parameter named `<pname>` is set to the value `<pexpr>`, which may be a literal value or an expression involving mathematical operations and other parameter names (using the `param$<name>` syntax). By default, the data type of the parameter is determined by the type of the supplied expression, but this can be overridden using the `-type` flag. The parameter description, units, UCD and Utype attributes may optionally be set using the other flags.

6.1.29 sort

Usage:

```
sort [-down] [-nullsfirst] <key-list>
```

Sorts the table according to the value of one or more algebraic expressions. The sort key expressions appear, as separate (space-separated) words, in `<key-list>`; sorting is done on the first

expression first, but if that results in a tie then the second one is used, and so on.

Each expression must evaluate to a type that it makes sense to sort, for instance numeric. If the `-down` flag is used, the sort order is descending rather than ascending.

Blank entries are by default considered to come at the end of the collation sequence, but if the `-nullsfirst` flag is given then they are considered to come at the start instead.

Syntax for the `<key-list>` argument is described in Section 10.

6.1.30 `sorthead`

Usage:

```
sorthead [-tail] [-down] [-nullsfirst] <nrows> <key-list>
```

Performs a sort on the table according to the value of one or more algebraic expressions, retaining only `<nrows>` rows at the head of the resulting sorted table. The sort key expressions appear, as separate (space-separated) words, in `<key-list>`; sorting is done on the first expression first, but if that results in a tie then the second one is used, and so on. Each expression must evaluate to a type that it makes sense to sort, for instance numeric.

If the `-tail` flag is used, then the last `<nrows>` rows rather than the first ones are retained.

If the `-down` flag is used the sort order is descending rather than ascending.

Blank entries are by default considered to come at the end of the collation sequence, but if the `-nullsfirst` flag is given then they are considered to come at the start instead.

This filter is functionally equivalent to using `sort` followed by `head`, but it can be done in one pass and is usually cheaper on memory and faster, as long as `<nrows>` is significantly lower than the size of the table.

Syntax for the `<key-list>` argument is described in Section 10.

6.1.31 `stats`

Usage:

```
stats [<item> ...]
```

Calculates statistics on the data in the table. This filter turns the table sideways, so that each row of the output corresponds to a column of the input. The columns of the output table contain statistical items such as mean, standard deviation etc corresponding to each column of the input table.

By default the output table contains columns for the following items:

- Name: Column name
- Mean: Average
- StDev: Population Standard deviation
- Minimum: Numeric minimum
- Maximum: Numeric maximum
- NGood: Number of non-blank cells

However, the output may be customised by supplying one or more `<item>` headings. These may be selected from the above as well as the following:

- **NBad**: Number of blank cells
- **Variance**: Population Variance
- **SampStDev**: Sample Standard Deviation
- **SampVariance**: Sample Variance
- **MedAbsDev**: Median Absolute Deviation
- **ScMedAbsDev**: Median Absolute Deviation * 1.4826
- **Skew**: Gamma 1 skewness measure
- **Kurtosis**: Gamma 2 peakedness measure
- **Sum**: Sum of values
- **MinPos**: Row index of numeric minimum
- **MaxPos**: Row index of numeric maximum
- **Cardinality**: Number of distinct values in column; values >100 ignored
- **Median**: Middle value in sequence
- **Quartile1**: First quartile
- **Quartile2**: Second quartile
- **Quartile3**: Third quartile

Additionally, the form "Q.nn" may be used to represent the quantile corresponding to the proportion 0.nn, e.g.:

- **Q.25**: First quartile
- **Q.625**: Fifth octile

Any parameters of the input table are propagated to the output one.

Note that quantile calculations (including median and quartiles) can be expensive on memory. If you want to calculate quantiles for large tables, it may be wise to reduce the number of columns to only those you need the quantiles for earlier in the pipeline. No interpolation is performed when calculating quantiles.

6.1.32 **tablename**

Usage:

```
tablename <name>
```

Sets the table's name attribute to the given string.

6.1.33 **tail**

Usage:

```
tail <nrows>
```

Include only the last <nrows> rows of the table. If the table has fewer than <nrows> rows then it will be unchanged.

6.1.34 **transpose**

Usage:

```
transpose [-namecol <col-id>]
```

Transposes the input table so that columns become rows and vice versa. The `-namecol` flag can be used to specify a column in the input table which will provide the column names for the output table. The first column of the output table will contain the column names of the input table.

Syntax for the `<col-id>` argument is described in Section 6.2.

6.1.35 `uniq`

Usage:

```
uniq [-count] [<colid-list>]
```

Eliminates adjacent rows which have the same values. If used with no arguments, then any row which has identical values to its predecessor is removed.

If the `<colid-list>` parameter is given then only the values in the specified columns must be equal in order for the row to be removed.

If the `-count` flag is given, then an additional column with the name `DupCount` will be prepended to the table giving a count of the number of duplicated input rows represented by each output row. A unique row has a `DupCount` value of 1.

Syntax for the `<colid-list>` argument is described in Section 6.3.

6.2 Specifying a Single Column

If an argument is specified in the help text for a command with the symbol `<col-id>` it means you must give a string which identifies one of the existing columns in a table.

There are three ways you can specify a column in this context:

Column Name

The name of the column may be used if it contains no spaces and doesn't start with a minus character ('-'). It is usually matched case insensitively. If multiple columns have the same name, the first one that matches is selected.

Column Index or \$ID

The index of the column may always be used; this is a useful fallback if the column name isn't suitable for some reason. The first column is '1', the second is '2' and so on. You may alternatively use the forms '\$1', '\$2' etc.

Tip: if counting which column has which index is giving you a headache, running `tpipe` with `omode=meta` or `omode=stats` on the table may help.

Column `ucd$` specifier

If the column has a Unified Content Descriptor (this will usually only be the case for VOTable or possibly FITS format tables) you can refer to it using an identifier of the form "`ucd$<ucd-spec>`". Depending on the version of UCD scheme used, UCDs can contain various punctuation marks such as underscores, semicolons and dots; for the purpose of this syntax these should all be represented as underscores ("_"). So to identify a column which has the UCD "`phot.mag;em.opt.R`", you should use the identifier "`ucd$phot_mag_em_opt_r`". Matching is not case-sensitive. Furthermore, a trailing underscore acts as a wildcard, so that the above column could also be referenced using the identifier "`ucd$phot_mag_`". If multiple columns have UCDs which match the given identifier, the first one will be used.

Column `utype$` specifier

If the column has a **Utype** (this will usually only be the case for VOTable or possibly FITS format tables) you can refer to it using an identifier of the form "`utype$<utype-spec>`". Utypes may contain various punctuation marks such as colons and dots; for the purpose of this syntax these should all be represented as underscores ("_"). So to identify a column which has the Utype "`ssa:Access.Format`", you should use the identifier "`utype$ssa_Access_format`". Matching is not case-sensitive. If multiple columns have Utypes which match the given

identifier, the first one will be used.

6.3 Specifying a List of Columns

If an argument is specified in the help text for a command with the symbol `<colid-list>` it means you must give a string which identifies a list of zero, one or more of the existing columns in a table. The string you specify is separated into separate tokens by whitespace, which means that you will normally have to surround it in single or double quotes to ensure that it is treated as a single argument and not several of them.

Each token in the `<colid-list>` string may be one of the following:

Column Name

The name of a column may be used if it contains no spaces and doesn't start with a minus character ('-'). It is usually matched case insensitively. If multiple columns have the same name, the first one that matches is selected.

Column Index or \$ID

The index of the column may always be used; this is a useful fallback if the column name isn't suitable for some reason. The first column is '1', the second is '2' and so on. You may alternatively use the forms '\$1', '\$2' etc.

Tip: if counting which column has which index is giving you a headache, running `tpipe` with `omode=meta` or `omode=stats` on the table may help.

Wildcard Expression

You can use a simple form of wildcard expression which expands to any columns in the table whose names match the pattern. Currently, the only special character is an asterisk '*' which matches any sequence of characters. To match an unknown sequence at the start or end of the string an asterisk must be given explicitly. Other than that, matching is usually case insensitive. The order of the expanded list is the same as the order in which the columns appear in the table.

Thus `"col*"` will match columns named `col1`, `Column2` and `COL_1024`, but not `decOld`. `"*MAG*"` will match columns named `magnitude`, `ABS_MAG_U` and `JMAG`. `"*"` on its own expands to a list of all the columns of the table in order.

Specifying a list which contains a given column more than once is not usually an error, but what effect it has depends on the function you are executing.

6.4 Output Modes

This section lists the output modes which can be used as the value of the `omode` parameter of `tpipe` and other commands. Typically, having produced a result table by pipeline processing an input one, you will write it out by specifying `omode=out` (or not using the `omode` parameter at all - `out` is the default). However, you can do other things such as calculate statistics, display metadata, etc. In some of these cases, additional parameters are required. The different output modes, with their associated parameters, are described in the following subsections.

6.4.1 cgi

Usage:

```
omode=cgi ofmt=<out-format>
```

Writes a table to standard output in a way suitable for use as output from a CGI (Common Gateway

Interface) program. This is very much like `out` mode but a short CGI header giving the MIME Content-Type is prepended to the output

Additional parameters for this output mode are:

`ofmt = <out-format> (String)`

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters).

[Default: `votable`]

6.4.2 `count`

Usage:

`omode=count`

Counts the number of rows and columns and writes the result to standard output.

6.4.3 `discard`

Usage:

`omode=discard`

Reads all the data in the table in sequential mode and discards it. May be useful in conjunction with the `assert` filter.

6.4.4 `gui`

Usage:

`omode=gui`

Displays the table in a scrollable window.

6.4.5 `meta`

Usage:

`omode=meta`

Prints the table metadata to standard output. The name and type etc of each column is tabulated, and table parameters are also shown.

See the `meta` filter for more flexible output of table metadata.

6.4.6 `out`

Usage:

`omode=out out=<out-table> ofmt=<out-format>`

Writes a new table.

Additional parameters for this output mode are:

out = <out-table> (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

[Default: -]

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

[Default: (auto)]

6.4.7 plastic

Usage:

```
omode=plastic transport=string|file client=<app-name>
```

Broadcasts the table to any registered Plastic-aware applications. PLASTIC, the PLatform for Astronomical Tool InterConnection, is a tool interoperability protocol. A *Plastic hub* must be running in order for this to work.

Additional parameters for this output mode are:

transport = string|file (*String*)

Determines the method (PLASTIC *message*) used to perform the PLASTIC communication. The choices are

- **string**: VOTable serialized as a string and passed as a call parameter (ivo://votech.org/votable/load). Not suitable for very large files.
- **file**: VOTable written to a temporary file and the filename passed as a call parameter (ivo://votech.org/votable/loadFromURL). The file ought to be deleted once it has been loaded. Not suitable for inter-machine communication.

If no value is set (null) then a decision will be taken based on the apparent size of the table.

client = <app-name> (*String*)

Gives the name of a PLASTIC listener application which is to receive the broadcast table. If a non-null value is given, then only the first registered application which reports its application name as that value will receive the message. If no value is supplied, the broadcast will be to all listening applications.

6.4.8 samp

Usage:

```
omode=samp format=<value> client=<name-or-id>
```

Sends the table to registered SAMP-aware applications subscribed to a suitable table load MType. SAMP, the Simple Application Messaging Protocol, is a tool interoperability protocol. A *SAMP Hub* must be running for this to work.

Additional parameters for this output mode are:

format = <value> (*String[]*)

Gives one or more table format types for attempting the table transmission over SAMP. If multiple values are supplied, they should be separated by spaces. Each value supplied for this parameter corresponds to a different MType which may be used for the transmission. If a single value is used, a SAMP broadcast will be used. If multiple values are used, each registered client will be interrogated to see whether it subscribes to the corresponding MTypes in order; the first one to which it is subscribed will be used to send the table. The standard options are

- `votable`: use MType `table.load.votable`
- `fits`: use MType `table.load.fits`

If any other string is used which corresponds to one of STILTS's known table output formats, an attempt will be made to use an ad-hoc MType of the form `table.load.format`.

[Default: `votable fits`]

`client = <name-or-id> (String)`

Identifies a registered SAMP client which is to receive the table. Either the client ID or the (case-insensitive) application name may be used. If a non-null value is given, then the table will be sent to only the first client with the given name or ID. If no value is supplied the table will be sent to all suitably subscribed clients.

6.4.9 stats

Usage:

`omode=stats`

Calculates and displays univariate statistics for each of the numeric columns in the table. The following entries are shown for each column as appropriate:

- mean
- population standard deviation
- minimum
- maximum
- number of non-null entries

See the `stats` filter for more flexible statistical calculations.

6.4.10 topcat

Usage:

`omode=topcat`

Attempts to display the output table directly in TOPCAT. If a TOPCAT instance is already running on the local host, an attempt will be made to open the table in that. A variety of mechanisms are used to attempt communication with an existing TOPCAT instance. In order:

1. SAMP using existing hub (TOPCAT v3.4+ only, requires SAMP hub to be running)
2. PLASTIC using existing hub (requires PLASTIC hub to be running)
3. SOAP (requires TOPCAT to run with somewhat deprecated `-soap` flag, may be limitations on table size)
4. SAMP using internal, short-lived hub (TOPCAT v3.4+ only, running hub not required, but may be slow. It's better to start an external hub, e.g. `topcat -exthub`)

Failing that, an attempt will be made to launch a new TOPCAT instance for display. This only works if the TOPCAT classes are on the class path.

If large tables are involved, starting TOPCAT with the `-disk` flag is probably a good idea.

6.4.11 `tosql`

Usage:

```
omode=tosql protocol=<jdbc-protocol> host=<value> db=<db-name>
           dbname=<table-name> write=create|dropcreate|append
           user=<username> password=<passwd>
```

Writes a new table to an SQL database. You need the appropriate JDBC drivers and `-Djdbc.drivers` set as usual (see Section 3.4).

Additional parameters for this output mode are:

protocol = <jdbc-protocol> (*String*)

The driver-specific sub-protocol specifier for the JDBC connection. For MySQL's Connector/J driver, this is `mysql`, and for PostgreSQL's driver it is `postgresql`. For other drivers, you may have to consult the driver documentation.

host = <value> (*String*)

The host which is acting as a database server.

[Default: localhost]

db = <db-name> (*String*)

The name of the database on the server into which the new table will be written.

dbname = <table-name> (*String*)

The name of the table which will be written to the database.

write = create|dropcreate|append (*WriteMode*)

Controls how the values are written to a table in the database. The options are:

- `create`: Creates a new table before writing. It is an error if a table of the same name already exists.
- `dropcreate`: Creates a new database table before writing. If a table of the same name already exists, it is dropped first.
- `append`: Appends to an existing table. An error results if the named table has the wrong structure (number or types of columns) for the data being written.

[Default: create]

user = <username> (*String*)

User name for the SQL connection to the database.

[Default: mbt]

password = <passwd> (*String*)

Password for the SQL connection to the database.

7 Crossmatching

STILTS offers flexible and efficient facilities for crossmatching tables. Crossmatching is identifying different rows, which may be in the same or different tables, that refer to the same item. In an astronomical context such an item is usually, though not necessarily, an astronomical source or object. This operation corresponds to what in database terminology is called a *join*.

There are various complexities to specifying such a match. In the first place you have to define what is the condition that must be satisfied for two rows to be considered matching. In the second place you must decide what happens if, for a given row, more than one match can be found. Finally, you have to decide what to do having worked out what the matched rows are; the result will generally be presented as a new output table, but there are various choices about what columns and rows it will consist of. Some of these issues are discussed in this section, and others in the reference sections on the tools themselves in Appendix B.

Matching can in general be a computationally intensive process. The algorithm used by the `tmatch*` tasks in STILTS, except in pathological cases, scales as $O(N \log(N))$ or thereabouts, where N is the total number of rows in all the tables being matched. No preparation (such as sorting) is required on the tables prior to invoking the matching operation. It is reasonably fast; for instance an RA, Dec positional match of two 10^5 -row catalogues takes of the order of 60 seconds on current (2005 laptop) hardware. Attempting matches with large tables can lead to running out of memory; the calculation just mentioned required a java heap size of around 200Mb (`-Xmx200M`).

In the current release of STILTS the following tasks are provided for crossmatching between local tables:

`tmatch2`

Generic crossmatching between two tables.

`tskymatch2`

Crossmatching between two tables where the matching criterion is a fixed separation on the sky. This is simply a stripped-down version of `tmatch2` provided for convenience when the full generality is not required.

`tmatch1`

Generic crossmatching internal to a single table. The basic task this performs is to identify groups of rows within a single table which match each other.

`tmatchn`

Generic crossmatching between multiple (>2) tables.

`tjoin`

Trivial join operation between multiple tables in which no row re-ordering is required. This barely warrants the term "crossmatch" and the concepts explained in the rest of this section are not relevant to it.

7.1 Match Criteria

Determining whether one row represents the same item as another is done by comparing the values in certain of their columns to see if they are the same or similar. The most common astronomical case is to say that two rows match if their celestial coordinates (right ascension and declination) are within a given small radius of each other on the sky. There are other possibilities; for instance the coordinates to compare may be in a Cartesian space, or have a higher (or lower) dimensionality than two, or the match may be exact rather than within an error radius....

If you just need to match two tables according to sky position with fixed errors you are

recommended to use the simplified `tskymatch2` task. For other cases, this section describes how to specify much more flexible match criteria for use with `tmatch1`, `tmatch2` or `tmatchn` by setting the following parameters:

matcher

Name of the match criteria type.

params

Fixed value(s) giving the parameters of the match (typically an error radius). If more than one value is required, the values should be separated by spaces.

values*

Expressions to be compared between rows. This will typically contain the names of one or more columns, but each element may be an algebraic expression (see Section 10) rather than just a column name if required. If more than one value is required, the values should be separated by spaces. There is one of these parameters for each table taking part in the match, so for `tmatch2` you must specify both `values1` and `values2`.

tuning

Fixed value(s) supplying tuning parameters for the match algorithm. If there is more than one value, they should be separated by spaces. This value will have a sensible default, so you do not need to supply it, but providing adjusted values may make your match run faster or require less memory (or the reverse). Adjusting tuning parameters will not change the result of any match, only the resources required to run it. Looking at the progress output of a match will indicate what tuning values have been used; adjusting the value a bit up or down is a good way to experiment.

For example, suppose we wish to locate objects in two tables which are within 3 arcseconds of each other on the sky. One table has columns RA and DEC which give coordinates in degrees, and the other has columns RArad and DECrad which give coordinates in radians. These are the arguments which would be used to tell `tmatch2` what the match criteria are:

```
matcher=sky
params=3
values1='RA DEC'
values2='radiansToDegrees(RArad) radiansToDegrees(DECrad)'
```

It is clearly important that corresponding values are comparable (in the same units) between the tables being matched, and in geometrically sensitive cases such as matching on the sky, it's important that they are the units expected by the matcher as well. To determine what those units are, either consult the roster below, or run the following command:

```
stilts tmatch2 help=matcher
```

which will tell you about all the known matchers and their associated `params`, `values*` and `tuning` parameters.

The following subsections list the basic `matcher` types and the requirements of their associated `params`, `values*` and `tuning` parameters. The units of the required values are given where significant.

7.1.1 sky: Sky Matching

```
matcher=sky values*='<ra/degrees> <dec/degrees>'
params='<max-error/arcsec>'
tuning='<healpix-k>'
```

values*:

- `ra/degrees`: Right Ascension
- `dec/degrees`: Declination

params:

- `max-error/arcsec`: Maximum separation along a great circle

tuning:

- `healpix-k`: Controls sky pixel size. Legal range 0 - 20. 0 is 60deg, 20 is 0.2".

The `sky` matcher compares positions on the celestial sphere with a fixed error radius. Rows are considered to match when the two (`ra`, `dec`) positions are within `max-error` arcseconds of each other along a great circle.

In fact this matching is not restricted to equatorial coordinates - the `ra` and `dec` parameters may represent any longitude-like and latitude-like coordinates in degrees, since the spherical geometry for the matching is unchanged under such transformations.

7.1.2 `skyerr`: Sky Matching with Per-Object Errors

```
matcher=skyerr values*='<ra/degrees> <dec/degrees> <error/arcsec>'
               params='<scale/arcsec>'
               tuning='<healpix-k>'
```

values*:

- `ra/degrees`: Right Ascension
- `dec/degrees`: Declination
- `error/arcsec`: Per-object error radius along a great circle

params:

- `scale/arcsec`: Rough average of per-object error distance; just used for tuning to set default pixel size

tuning:

- `healpix-k`: Controls sky pixel size. Legal range 0 - 20. 0 is 60deg, 20 is 0.2".

The `skyerr` matcher compares positions on the celestial sphere using error radii which can be different for each row. Rows are considered to match when the separation between the two `ra`, `dec` positions is no larger than the sum of the two per-row `error` values.

The `scale` parameter should be a rough average value of the error distances. It is used only to set a sensible default for `healpix-k` tuning parameter, and its value does not affect the result. If you set `healpix-k` directly, its value is ignored.

As with `sky` matching, other longitude/latitude coordinate pairs may be used in place of right ascension and declination.

Note: the semantics of this matcher have changed slightly at version 2.4 of STILTS. In earlier versions the single parameter was named `max-error` and provided an additional constraint on the maximum accepted separation between matched objects. For most uses, the old and new behaviours

are expected to give the same results, but in cases of difference, the new behaviour is more likely what you want.

7.1.3 `skyeclipse`: Sky Matching of Elliptical Regions

```
matcher=skyeclipse values*='<ra/degrees> <dec/degrees> <primary-radius/arcsec>
                           <secondary-radius/arcsec>
                           <position-angle/degrees>'
                      params='<scale/arcsec>'
                      tuning='<healpix-k>'
```

values*:

- `ra/degrees`: Right ascension of centre
- `dec/degrees`: Declination of centre
- `primary-radius/arcsec`: Length of ellipse semi-major axis
- `secondary-radius/arcsec`: Length of ellipse semi-minor axis
- `position-angle/degrees`: Position angle - measured from north pole to primary axis, in direction of positive RA

params:

- `scale/arcsec`: Rough average of ellipse major radius; just used for tuning to set default pixel size

tuning:

- `healpix-k`: Controls sky pixel size. Legal range 0 - 20. 0 is 60deg, 20 is 0.2".

The `skyeclipse` matcher compares elliptical regions on the sky for overlap. Each row has to provide five values, giving the centre, the major and minor radii, and the position angle of an ellipse. Rows are considered to match if there is any overlap between the ellipses. The goodness of match is a normalised generalisation of the symmetrical case used by the `skyerr` matcher, in which the best possible match is two concentric ellipses, and the worst allowable match is when the circumferences just touch.

The calculations are approximate since in some cases they rely on projecting the ellipses onto a Cartesian tangent plane before evaluating the match, so for larger ellipses the criterion will be less exact. For objects the size of most observed stars or galaxies, this approximation is not expected to be problematic.

The `scale` parameter must be supplied, and should be a rough average value of the major radii. it is used only to set a sensible default for the `healpix-k` tuning parameter, and its value does not affect the result. If you set `healpix-k` directly, the value of `scale` is ignored.

7.1.4 `sky3d`: Spherical Polar Matching

```
matcher=sky3d values*='<ra/degrees> <dec/degrees> <distance>'
                params='<error/units of distance>'
                tuning='<bin-factor>'
```

values*:

- `ra/degrees`: Right Ascension
- `dec/degrees`: Declination
- `distance`: Distance from origin

params:

- `error/units` of distance: Maximum Cartesian separation for match

tuning:

- `bin-factor`: Scaling factor to adjust bin size; larger values mean larger bins

The `sky3d` matcher compares positions in the volume of the sky taking account of distance from the observer. The position in three-dimensional space is calculated for each row using the `ra`, `dec` and `distance` as spherical polar coordinates, where `distance` is the distance from the observer along the line of sight. Rows are considered to match when their positions in this space are within `error` units of each other. The units of `error` are the same as those of `distance`.

As with `sky` matching, other longitude/latitude coordinate pairs may be used in place of right ascension and declination.

7.1.5 `exact`: Exact Matching

```
matcher=exact values*='<matched-value>'
```

values*:

- `matched-value`: Value for exact match

The `exact` matcher compares arbitrary key values for exact equality. Rows are considered to match only if the values in their `matched-value` columns are exactly the same. These values can be strings, numbers, or anything else. A blank value never matches, not even with another blank one. Since the `params` parameter holds no values, it does not have to be specified. Note that the values must also be of the same type, so for instance a Long (64-bit) integer value will not match an Integer (32-bit) value.

7.1.6 `1d`, `2d`, ...: Isotropic Cartesian Matching

```
matcher=1d values*='<x>'
           params='<error>'
           tuning='<bin-factor>'
```

values*:

- `x`: Cartesian co-ordinate #1

params:

- `error`: Maximum Cartesian separation for match

tuning:

- `bin-factor`: Scaling factor to adjust bin size; larger values mean larger bins

```
matcher=2d values*='<x> <y>'
           params='<error>'
```

```
tuning='<bin-factor>'

values*:

- x: Cartesian co-ordinate #1
- y: Cartesian co-ordinate #2

params:

- error: Maximum Cartesian separation for match

tuning:

- bin-factor: Scaling factor to adjust bin size; larger values mean larger bins

```

The `1d` matcher compares positions in 1-dimensional Cartesian space. Rows are considered to match if their `x` column values differ by no more than `error`.

The `2d` matcher compares positions in 2-dimensional Cartesian space. Rows are considered to match if the difference in their `(x,y)` positions reckoned using Pythagoras is less than `error`.

Matching in any number of Cartesian dimensions can be done by extending this syntax in the obvious way.

7.1.7 `2d_anisotropic, ...`: Anisotropic Cartesian Matching

```
matcher=2d_anisotropic values*='<x> <y>'
                        params='<error-in-x> <error-in-y>'
                        tuning='<bin-factor>'
```

```
values*:

- x: Cartesian co-ordinate #1
- y: Cartesian co-ordinate #2

params:

- error-in-x: Axis length of error ellipse in Cartesian co-ordinate #1 direction
- error-in-y: Axis length of error ellipse in Cartesian co-ordinate #2 direction

tuning:

- bin-factor: Scaling factor to adjust bin size; larger values mean larger bins

```

The `2d_anisotropic` matcher compares positions in 2-dimensional Cartesian space using an anisotropic metric. Rows are considered to match if their `(x,y)` positions fall within an error ellipse with axis lengths `error-in-x`, `error-in-y` of each other. This kind of match will typically be used for non-'spatial' spaces, for instance (magnitude,redshift) space, in which the metrics along different axes are not related to each other.

Matching in any number of dimensions of Cartesian space using an anisotropic metric can be done by extending this syntax in the obvious way.

7.1.8 `2d_cuboid, ...`: Cuboid Cartesian Matching

```
matcher=2d_cuboid values*='<x> <y>'
                  params='<error-in-x> <error-in-y>'
                  tuning='<bin-factor>'
```

values*:

- x: Cartesian co-ordinate #1
- y: Cartesian co-ordinate #2

params:

- error-in-x: Half length of cuboid in Cartesian co-ordinate #1 direction
- error-in-y: Half length of cuboid in Cartesian co-ordinate #2 direction

tuning:

- bin-factor: Scaling factor to adjust bin size; larger values mean larger bins

The `2d_cuboid` matcher compares positions in 2-dimensional Cartesian space in cuboidal cells. Rows are considered to match if their (x,y) positions fall within an error cuboid with half-axis lengths `error-in-x`, `error-in-y` of each other. This kind of match is suitable for grouping items into pixels, though it's not a very efficient way of doing that.

Matching in any number of dimensions using N-dimensional hyper-cuboids can be done by extending this syntax in the obvious way.

7.1.9 `1d_err`, `2d_err`, ...: Cartesian Matching with Per-Object Errors

```
matcher=2d_err values*='<x> <y> <error>'
                params='<scale>'
                tuning='<bin-factor>'
```

values*:

- x: Cartesian co-ordinate #1
- y: Cartesian co-ordinate #2
- error: Per-object error radius

params:

- scale: Rough average of per-object error distance; just used for tuning in conjunction with bin factor

tuning:

- bin-factor: Scaling factor to adjust bin size; larger values mean larger bins

The `1d_err`, `2d_err`, ... matchers compare positions in N-dimensional Cartesian space like the `1d`, `2d` matchers described in Section 7.1.6, except that the match radius can be different for each row. Rows are considered to match when the separation reckoned by Pythagoras between the x, y, ... positions is no larger than the sum of the two per-row `error` values. Matching in any number of Cartesian dimensions can be done by extending this syntax in the obvious way.

The `scale` parameter must be supplied, and should be approximately the characteristic size of the per-object error values. In conjunction with the `bin-factor` tuning parameter its value affects the

performance of the match, but not the result.

7.1.10 2d_ellipse: Cartesian Matching of Elliptical Regions

```
matcher=2d_ellipse values*='<x> <y> <primary-radius> <secondary-radius>
                           <orientation-angle/degrees>'
                      params='<scale>'
                      tuning='<bin-factor>'
```

values*:

- `x`: X coordinate of centre
- `y`: Y coordinate of centre
- `primary-radius`: Length of ellipse semi-major axis
- `secondary-radius`: Length of ellipse semi-minor axis
- `orientation-angle/degrees`: Angle from X axis towards Y axis of semi-major axis

params:

- `scale`: Rough average of per-object error distance; just used for tuning in conjunction with bin factor

tuning:

- `bin-factor`: Scaling factor to adjust bin size; larger values mean larger bins

The `2d_ellipse` matcher compares elliptical regions in a 2d plane for overlap. Each row has to specify five values, giving the centre, the major and minor radii, and the orientation angle of an ellipse. Rows are considered to match if there is any overlap between the ellipses. The goodness of match is a normalised generalisation of the symmetrical case used by the isotropic matcher, in which the best possible match is two concentric ellipses, and the worst allowable match is when the circumferences just touch.

Note the orientation angle is measured anticlockwise from the horizontal, unlike the position angle used by the `skyellipse` matcher.

The `scale` parameter must be supplied, and should be approximately the characteristic size of the per-object major radius. In conjunction with the `bin-factor` tuning parameter its value affects the performance of the match, but not the result.

7.1.11 Custom Matchers

For advanced users, it is possible to supply the name of a class on the classpath which implements the `uk.ac.starlink.table.join.MatchEngine` interface and which has a no-arg constructor. This allows java programmers to write their own matchers using any match criteria and binning algorithms they choose.

7.1.12 Matcher Combinations

In addition to the matching criteria listed in the previous subsections, you can build your own by combining any of these. To do this, take the two (or more) matchers that you want to use, and separate their names with a "+" character. The `values*` parameters of the combined matcher should then hold the concatenation of the `values*` entries of the constituent matchers, and the same for the `params` parameter.

So for instance the matcher "sky+1d" could be used with the following syntax:

```
matcher=sky+1d values*='<ra/degrees> <dec/degrees> <x>'
                params='<max-error/arcsec> <error>'
                tuning='<healpix-k> <bin-factor>'
```

values*:

- ra/degrees: Right Ascension
- dec/degrees: Declination
- x: Cartesian co-ordinate #1

params:

- max-error/arcsec: Maximum separation along a great circle
- error: Maximum Cartesian separation for match

tuning:

- healpix-k: Controls sky pixel size. Legal range 0 - 20. 0 is 60deg, 20 is 0.2".
- bin-factor: Scaling factor to adjust bin size; larger values mean larger bins

This would compare positions on the sky with an additional scalar constraint. Rows are considered to match if *both* their ra, dec positions are within max-error arcseconds of each other along a great circle (as for matcher=sky) *and* their x values differ by no more than error (as for matcher=1d).

This example might be used for instance to identify objects from two catalogues which are within a couple of arcseconds and also 0.5 blue magnitudes of each other. Rolling your own matchers in this way can give you very flexible match constraints.

7.2 Multi-Object Matches

The generic matching in STILTS is determined by specified match criteria, as described in Section 7.1. These criteria give conditions for whether two items (table rows) count as matched with each other. In the case of a pair match, as provided by tmatch2, it is clear how this is to be interpreted.

However, some of the matching tasks (tmatchn in group mode and tmatch1) search for match groups which may have more than two members. This section explains precisely how STILTS applies the pair-wise matching criteria it is given to identifying multi-object groups.

In a multi-object match context, the matcher identifies a matched group as the largest possible group of objects in which each is linked by a pair match to *any* other object in the group - it is a group of "friends of friends". Formally, the set of matched groups is a set of disjoint graphs whose nodes are input table rows and whose edges are successful pair matches, where no successful pair match exists between nodes in different elements of that set. Thus the set has a minimal number of elements, and each of its elements is a matched group of maximal size. The important point to note is that for any particular pair in a matched group, there is no guarantee that the two objects match each other, only that you can hop from one to the other via pairs which do match.

So in the case of a multi-object sky match on a field which is very crowded compared to the specified error radius, it is quite possible for *all* the objects in the input table(s) to end up as part of the same large matching group. Results at or near this percolation threshold are (a) probably not useful and (b) likely to take a long time to run. Some care should therefore be exercised when specifying match criteria in multi-object match contexts.

8 Plotting

As of version 3.0 (October 2014), STILTS offers plotting commands corresponding to the new-style plots in version 4 of the TOPCAT application. The commands are currently:

- `plot2plane` (Appendix B.7): Draws a plane plot
- `plot2sky` (Appendix B.8): Draws a sky plot
- `plot2cube` (Appendix B.9): Draws a cube plot
- `plot2sphere` (Appendix B.10): Draws a sphere plot
- `plot2time` (Appendix B.11): Draws a time plot

(In previous versions the less capable commands `plot2d`, `plot3d` and `plothist` were available - these are now deprecated, but described in Section 9).

These commands all have a similar structure. The *plot surface*, or geometry of the plot, is defined by which command you use (for instance, if you want to plot longitude/latitude data on the celestial sphere, use `plot2sky`). Content is added to the plot by specifying zero or more *plot layers*, as described in Section 8.3 below. Section 8.4 describes the shading modes which affect how colouring is performed for some of the layer types. Once a plot has been specified, it can be displayed on the screen or exported in some way according to a selected output mode (Section 8.5) and perhaps export format (Section 8.6). Plots displayed to the screen are by default "live" - they can be resized and navigated around (pan, zoom, rotate, ...) using the mouse in the same way as in a TOPCAT window.

These commands allow you to make all the plots that can be produced with TOPCAT, in some cases with more flexibility in configuration. Unlike TOPCAT, the size of table you can plot is not limited by the size of table you can load into the application. In most cases, STILTS will generate plots from arbitrarily large data sets with fixed (and modest) memory requirements. Performance is of course highly dependent on the details of the plot, but for instance an all-sky density plot for 2 billion points can be produced in the order of 30 minutes.

8.1 Plot Parameters

The plotting commands offer a great deal of control over what is plotted and how it is represented, and thus unavoidably have lots of parameters. When looking at the command documentation in Appendix B the Usage sections may look rather daunting. However, the discussion below and the Examples sections should help. Generating a simple plot is straightforward and can be done with only four or five parameters; if you want to represent more complicated data or have specific preferences for appearance then you can consult the documentation for the additional options.

As a simple example, if a file "cat.fits" contains the columns RMAG and BMAG for red and blue magnitudes, you can draw a two-dimensional colour-magnitude scatter plot with the command:

```
stilts plot2plane layer_1=mark in_1=cat.fits x_1=BMAG-RMAG y_1=BMAG
```

Since an output file is not specified, the plot is shown in a window on the screen. This plot window is "live" - you can resize the window, or pan and zoom around it using the same mouse controls as in TOPCAT. To send the output to a PNG file, do instead:

```
stilts plot2plane layer_1=mark in_1=cat.fits x_1=BMAG-RMAG y_1=BMAG out=fig.png
```

We can adjust the plot by inverting the Y axis so it increases downwards instead of upwards:

```
stilts plot2plane
      yflip=true
      layer_1=mark in_1=cat.fits x_1=BMAG-RMAG y_1=BMAG
```

The parameters of the plot now fall into two groups. Global parameters, without suffixes, make global adjustments to the plot. In this example `yflip=true` inverts the Y axis. **Layer parameters**, with suffixes, are introduced by a `layer` parameter and grouped together by a given suffix. Each layer group defines a plot layer with content to be drawn on the plot surface. In this case the layer is of type `mark` (draw markers) and the suffix is `"_1"`. Global and Layer parameters are described separately in the following subsections.

8.1.1 Global Parameters

The global plot parameters are documented in the usage sections of the various plot commands (e.g. Appendix B.7.1). They deal with things like positioning the plot axes, fixing the data bounds, selecting font types and sizes, adjusting grids and tickmarks, configuring how interactive navigation works, managing data storage, and so on. They are all optional, since they all have sensible defaults, for instance data bounds will be determined from the supplied data if they are not given explicitly.

8.1.2 Layer Parameters

The layer parameters come in groups, each specifying the details of one plot layer. Each layer type has its own list of parameters. A plot layer is introduced on the command line with a parameter of the form

```
layer<suffix>=<layer-type>
```

and any other parameters with the same `<suffix>` are considered to apply to the same layer. In the basic example we considered:

```
stilts plot2plane layer_1=mark in_1=cat.fits x_1=BMAG-RMAG y_1=BMAG
```

the suffix is `"_1"` and the layer type associated with it is `mark` (plotting markers to make a scatter plot). The different layer types are documented in Section 8.3, and each has its own set of parameters, some of which are mandatory and some which are optional with sensible defaults. In the documentation, the suffix is represented as `"N"`. For instance the `mark` layer type requires you to specify an input table (`inN`) and point positions (`xN` and `yN`). Since the suffix we have used in the example for the `layerN` parameter is `"_1"`, we have written `in_1`, `x_1` and `y_1`. The `mark` layer has some optional style parameters as well, so we could adjust the plot's appearance by adding `shape_1=cross size_1=4 color_1=blue`.

You can have as many layers as you like (even none), so we could overplot two datasets from different input files like this:

```
stilts plot2plane
  layer_1=mark in_1=cat1.fits x_1=BMAG-RMAG y_1=BMAG color_1=magenta size_1=5
  layer_2=mark in_2=cat2.fits x_2=mag_b-mag_r y_2=mag_b color_2=cyan size_2=5
```

We have assigned different colours to the different layers and boosted the marker size to 5 pixels.

As a convenience, if the same value is used for all the layers, you can omit the suffix. So to avoid having to specify the same markers size for both layers, you can write instead:

```
stilts plot2plane
  size=5
  layer_1=mark in_1=cat1.fits x_1=BMAG-RMAG y_1=BMAG color_1=magenta
  layer_2=mark in_2=cat2.fits x_2=mag_b-mag_r y_2=mag_b color_2=cyan
```

Although the `size` parameter no longer has an explicit suffix, it's still a layer parameter, it just applies to multiple layers. This shorthand works for all layer parameters. Here is another example which also shows how you can use the `icmdN` parameter to pre-process input data prior to performing the plot. Here, we make two different selections of the input rows to plot two different data sets.

```
stilts plot2plane
  in=cat.fits x=BMAG-RMAG y=BMAG
  layer_1=mark icmd_1='select vel<1000' color_1=blue
  layer_2=mark icmd_2='select vel>=1000' color_2=red
```

The input tables and data values are the same for both datasets, so we can just supply the parameters `in`, `x` and `y`, rather than `in_1`, `in_2` etc.

Any string can be used as a suffix, including the empty string (though an empty string can cause confusion if there are multiple layers). The suffixing is also slightly more sophisticated than described above; to find parameters relating to a layer with a given suffix, the parameter looks first using the whole suffix, and strips single characters off it until it has none left. So if a layer is introduced with the parameter `layer_ab`, you can give the marker shape using any of the parameters `shape_ab`, `shape_a`, `shape_` or `shape`. If more than one of these is present, the first one in that list will be used (the order in which they appear on the command line is not significant). This can be used to group sets of layers.

By default, if multiple layers are specified, they are plotted in the order in which the introducing `layerN` parameters appear on the command line. This may be relevant, since layers plotted later sometimes obscure ones plotted earlier. You can alter the order of plotting with the `seq` (global) parameter, which is a comma-separated list of layer suffixes giving the sequence in which layers should be plotted. So adding "`seq=_2,_1`" would cause layer `_2` to be plotted before layer `_1`, instead of the other way round.

By default, if more than one layer is plotted, a legend will appear labelling the datasets. The dataset labels appearing in the legend are by default the layer suffixes specified on the command line. However, the labels can be given explicitly with the `legendN` parameter, so for instance in the example above `leglabel_1=Slow` `leglabel_2=Fast` would adjust the legend accordingly. Legend appearance and positioning can be adjusted by various `leg*` global parameters.

8.1.3 Animation

The plotting commands can be used to produce animations. This is done by supplying an *animation control table* using the `animate` parameter (which has associated `afmt` and `acmd` parameters for specifying its file format and applying filters). One output image is produced for each row of the control table. The columns of the table have names which correspond to plot command parameters, and for each row, the basic plot command is executed with the parameters on the command line supplied or replaced by those from the table. This is most commonly used for providing a movie of the kind of navigation you can do interactively with the mouse, but other applications are possible.

For instance, given the following animation control table with the name "bounds.txt", in ASCII format:

```
#  xmax  ymax
   4.0   2.0
   3.0   1.5
   2.0   1.0
   1.0   0.5
```

then this command:

```
stilts plot2plane xmin=0 ymin=0
  layer_1=mark in_1=gums_smc.fits x_1=ag y_1=av
  animate=bounds.txt afmt=ascii
```

would produce a 4-frame animation zooming in towards the origin.

If output is to the screen (`omode=swing`) the animation can be seen directly. If it is to an output file (`omode=out`) then a number of output files is written with sequence numbers, so adding the

parameter "out=x.png" to the above command would produce 4 files, x-1.png, x-2.png, x-3.png and x-4.png. Padding zeros are used to keep the files in alphanumeric sequence, so for instance in a 500-frame animation the first one would be named x-001.png. STILTS does not actually turn these files into a single animated output file, but you can use other tools to do this, for instance using ImageMagick:

```
convert x-*.png xmovie.gif
```

will produce an animated gif from the input frames.

You can create the animation control table any way you like, but you may find the `tloop` command convenient. For instance the above table can be written like this:

```
stilts tloop xmax 4 0 -1 ocmd='addcol ymax xmax*0.5' ofmt=ascii
```

You can pipe the output of `tloop` (or any other command) as the animation table on the unix command line by specifying `animate=-` (the "-" character stands for standard input). Note however that in this case you must explicitly give the file format (using the `afmt` parameter) and it must be a format which STILTS is capable of streaming (VOTable is suitable; ASCII is not).

A common requirement is to produce an animation of rotating a 3-d plot. Here's an example of how to do it from a unix shell:

```
stilts tloop phi 15 375 2 ofmt=votable \  
| stilts plot2sphere layer_1=mark in_1=hip_main.fits lon_1=radeg lat_1=dedeg r_1=plx \  
animate=- afmt=votable
```

The `phi` parameter controls the angle from which the 3D plot is viewed, and here it is incremented by 2 degrees for each frame. The same thing would work for `plot2cube` as well as `plot2sphere`.

Note that producing animations in this way is usually much more efficient than writing a shell script which invokes STILTS multiple times. The plot commands also employ multi-threading when animating to output files, so should make efficient use of multi-core machines (though currently animations to the screen are not multi-threaded).

8.2 Surface Types

The different `plot2*` commands correspond to different plot surface geometries. The different commands come with their own specific axis configuration parameters. Some of the plot layer types are specific to certain surface types. When supplying data from input tables to plot layers, the coordinate values you need to supply (and hence the corresponding parameter names) are determined not by the layer type, but by the surface type. For instance, point positions for layer N on a 2-d Cartesian surface (`plot2plane` command) are given using parameters `xN` and `yN`, but when plotting to the celestial sphere (`plot2sky` command) you supply `lonN` and `latN`.

The following list summarises the available surface types and their corresponding positional coordinates.

Plane (`plot2plane`)

2-dimensional Cartesian axes. Positional coordinates are supplied as `x`, `y` pairs. Note that this command can also be used to draw histograms.

Sky (`plot2sky`)

Celestial sphere. Positional coordinates are supplied as `lon`, `lat` pairs, giving longitude and latitude in decimal degrees. A number of different projections are available, and conversion between different celestial coordinate systems can also be performed. You could use it for other spherical coordinate systems too (like the surface of a planet).

Cube (`plot2cube`)

3-dimensional Cartesian axes. Positional coordinates are supplied as x, y, z triples.

Sphere (plot2sphere)

3-dimensional isotropic space with spherical polar coordinates. Positional coordinates are supplied as lon, lat, r triples, giving longitude and latitude in decimal degrees, and radius in an arbitrary unit. The plotting surface (space) is similar to Cube, except that the unit distance is always the same in all three directions.

Time (plot2time)

2-dimensional axes, but the horizontal axis represents time. The axis may be labelled in various ways (ISO-8601 dates, decimal year, MJD etc). Positional coordinates are supplied as t, y pairs. How to provide a data value representing a time is somewhat under-documented, but reading data from a time-sensitive format such as CDF will give column values that can be used as times. This surface type is somewhat experimental, and the `plot2time` command currently lacks some important features.

8.3 Layer Types

The different plot layers and how to configure them with parameters is given in the following subsections. The layers which may be plotted on a particular surface depend on the plot geometry, so not all of these are available for every plot command.

8.3.1 mark

Plots a marker of fixed size and shape at each position.

Usage Overview:

```
layerN=mark shapeN=filled_circle|open_circle|... sizeN=<pixels>
          shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN>
          <pos-coord-paramsN> inN=<table> ifmtN=<in-format>
          istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix N .

Positional Coordinate Parameters:

The positional coordinates `<pos-coord-paramsN>` give a position for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (`plot2plane`) the parameters would be x_N and y_N . The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer N input table as specified by parameter in_N . The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (Boolean)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

shadingN = auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN> (ShapeMode)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- `auto` (Section 8.4.1)
- `flat` (Section 8.4.2)
- `translucent` (Section 8.4.3)
- `transparent` (Section 8.4.4)
- `density` (Section 8.4.5)
- `aux` (Section 8.4.6)
- `weighted` (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: `auto`]

shapeN = filled_circle|open_circle|... (MarkShape)

Sets the shape of markers that are plotted at each position of the scatter plot.

The available options are:

- `filled_circle`
- `open_circle`
- `cross`
- `x`

- open_square
- open_diamond
- open_triangle_up
- open_triangle_down
- filled_square
- filled_diamond
- filled_triangle_up
- filled_triangle_down

[Default: filled_circle]

sizeN = <pixels> (Integer)

Size of the scatter plot markers. The unit is pixels, in most cases the marker is approximately twice the size of the supplied value.

[Default: 1]

8.3.2 size

Plots a marker of fixed shape but variable size at each position. The size is determined by an additional input data value.

The actual size of the markers depends on the setting of the `autoscale` parameter. If autoscaling is off, then the basic size of each marker is the input data value in units of pixels. If autoscaling is on, then the data values are gathered for all the currently visible points, and a scaling factor is applied so that the largest ones will be a sensible size (a few tens of pixels). This basic size can be further adjusted with the `scale` factor.

Currently data values of zero always correspond to marker size of zero, negative data values are not represented, and the mapping is linear. An absolute maximum of 100 pixels is also imposed on marker sizes. Other options may be introduced in future.

Note: for marker sizes that correspond to data values in data coordinates, you may find Error plotting more appropriate.

Usage Overview:

```
layerN=size shapeN=filled_circle|open_circle|... scaleN=<factor>
          autoscaleN=true|false
          shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN>
          <pos-coord-paramsN> sizeN=<num-expr> inN=<table>
          ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix `N`.

Positional Coordinate Parameters:

The positional coordinates `<pos-coord-paramsN>` give a position for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (`plot2plane`) the parameters would be `xN` and `yN`. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

autoscaleN = true|false (Boolean)

Determines whether the basic size of variable sized markers is automatically scaled to have a sensible size. If true, then the sizes of all the plotted markers are examined, and some

dynamically calculated factor is applied to them all to make them a sensible size (by default, the largest ones will be a few tens of pixels). If false, the sizes will be the actual input values in units of pixels.

If auto-scaling is off, then markers will keep exactly the same screen size during pan and zoom operations; if it's on, then the visible sizes will change according to what other points are currently plotted.

Marker size is also affected by the `scale` parameter.

[Default: `true`]

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (`auto`) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (`auto`)]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

scaleN = <factor> (*Double*)

Scales the size of variable-sized markers. The default is 1, smaller or larger values multiply the visible sizes accordingly.

[Default: 1]

shadingN = `auto|flat|translucent|transparent|density|aux|weighted` *<shade-paramsN>*
(ShapeMode)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- `auto` (Section 8.4.1)
- `flat` (Section 8.4.2)
- `translucent` (Section 8.4.3)
- `transparent` (Section 8.4.4)
- `density` (Section 8.4.5)
- `aux` (Section 8.4.6)
- `weighted` (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: `auto`]

shapeN = `filled_circle|open_circle|...` *(MarkShape)*

Sets the shape of markers that are plotted at each position of the scatter plot.

The available options are:

- `filled_circle`
- `open_circle`
- `cross`
- `x`
- `open_square`
- `open_diamond`
- `open_triangle_up`
- `open_triangle_down`
- `filled_square`
- `filled_diamond`
- `filled_triangle_up`
- `filled_triangle_down`

[Default: `filled_circle`]

sizeN = *<num-expr>* *(String)*

Size to draw each sized marker. Units are pixels unless auto-scaling is in effect, in which case units are arbitrary. The plotted size is also affected by the `scale` value.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.3 `sizexy`

Plots a shaped marker with variable horizontal and vertical extents at each position. The X and Y dimensions are determined by two additional input data values.

The actual size of the markers depends on the setting of the `autoscale` parameter. If autoscaling is off, the basic dimensions of each marker are given by the input data values in units of pixels. If autoscaling is on, the data values are gathered for all the currently visible points, and scaling factors are applied so that the largest ones will be a sensible size (a few tens of pixels). This autoscaling happens independently for the X and Y directions. The basic sizes can be further adjusted with the `scale` factor.

Currently data values of zero always correspond to marker dimension of zero, negative data values are not represented, and the mapping is linear. An absolute maximum of 100 pixels is also imposed on marker sizes. Other options may be introduced in future.

Note: for marker sizes that correspond to data values in data coordinates, you may find Error plotting more appropriate.

Usage Overview:

```
layerN=sizexy shapeN=open_rectangle|open_triangle|... scaleN=<factor>
autoscaleN=true|false
shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-params>
<pos-coord-paramsN> xsizeN=<num-expr> ysizeN=<num-expr>
inN=<table> ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

Positional Coordinate Parameters:

The positional coordinates <pos-coord-paramsN> give a position for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (plot2plane) the parameters would be xN and yN. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

autoscaleN = true|false (Boolean)

Determines whether the basic size of variable sized markers is automatically scaled to have a sensible size. If true, then the sizes of all the plotted markers are examined, and some dynamically calculated factor is applied to them all to make them a sensible size (by default, the largest ones will be a few tens of pixels). If false, the sizes will be the actual input values in units of pixels.

If auto-scaling is off, then markers will keep exactly the same screen size during pan and zoom operations; if it's on, then the visible sizes will change according to what other points are currently plotted.

Marker size is also affected by the `scale` parameter.

[Default: true]

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer N input table as specified by parameter inN. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file filename to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter inN. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (auto) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program

will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

scaleN = <factor> (*Double*)

Scales the size of variable-sized markers. The default is 1, smaller or larger values multiply the visible sizes accordingly.

[Default: 1]

shadingN = auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN> (*ShapeMode*)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- auto (Section 8.4.1)
- flat (Section 8.4.2)
- translucent (Section 8.4.3)
- transparent (Section 8.4.4)
- density (Section 8.4.5)
- aux (Section 8.4.6)
- weighted (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: auto]

shapeN = open_rectangle|open_triangle|... (*XYShape*)

The available options are:

- open_rectangle
- open_triangle
- open_triangle_down
- open_diamond
- open_ellipse
- filled_rectangle

- filled_triangle
- filled_triangle_down
- filled_diamond
- filled_ellipse

[Default: open_rectangle]

xsizeN = <num-expr> (*String*)

Horizontal extent of each marker. Units are pixels unless auto-scaling is in effect, in which case units are arbitrary.

The value is a numeric algebraic expression based on column names as described in Section 10.

ysizeN = <num-expr> (*String*)

Vertical extent of each marker. Units are pixels unless auto-scaling is in effect, in which case units are arbitrary.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.4 xyvector

Plots directed lines from the data position given delta values for the coordinates. The plotted markers are typically little arrows, but there are other options.

In some cases such delta values may be the actual magnitude required for the plot, but often the vector data represents a value which has a different magnitude or is in different units to the positional data. As a convenience for this case, the plotter can optionally scale the magnitudes of all the vectors to make them a sensible size, so by default the largest ones are a few tens of pixels long. This auto-scaling is in operation by default, but it can be turned off or adjusted with the scaling and auto-scaling options.

Usage Overview:

```
layerN=xyvector arrowN=small_arrow|medium_arrow|... scaleN=<factor>
               autoscaleN=true|false
               shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-param>
               xN=<num-expr> yN=<num-expr> xdeltaN=<num-expr>
               ydeltaN=<num-expr> inN=<table> ifmtN=<in-format>
               istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

arrowN = small_arrow|medium_arrow|... (*ErrorRenderer*)

How arrows are represented.

The available options are:

- small_arrow
- medium_arrow
- large_arrow
- small_open_dart
- medium_open_dart
- large_open_dart
- small_filled_dart
- medium_filled_dart
- large_filled_dart

- `lines`
- `capped_lines`

[Default: `small_arrow`]

`autoscaleN = true|false` (*Boolean*)

Determines whether the default size of variable-sized markers like vectors and ellipses are automatically scaled to have a sensible size. If true, then the sizes of all the plotted markers are examined, and some dynamically calculated factor is applied to them all to make them a sensible size (by default, the largest ones will be a few tens of pixels). If false, the sizes will be the actual input values interpreted in data coordinates.

If auto-scaling is on, then markers will keep approximately the same screen size during zoom operations; if it's off, they will keep the same size in data coordinates.

Marker size is also affected by the `scale` parameter.

[Default: `true`]

`icmdN = <cmds>` (*ProcessingStep[]*)

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

`ifmtN = <in-format>` (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (`auto`) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (`auto`)]

`inN = <table>` (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

`istreamN = true|false` (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less

resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

scaleN = <factor> (*Double*)

Affects the size of variable-sized markers like vectors and ellipses. The default value is 1, smaller or larger values multiply the visible sizes accordingly.

[Default: 1]

shadingN = `auto|flat|translucent|transparent|density|aux|weighted` <shade-paramsN> (*ShapeMode*)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- `auto` (Section 8.4.1)
- `flat` (Section 8.4.2)
- `translucent` (Section 8.4.3)
- `transparent` (Section 8.4.4)
- `density` (Section 8.4.5)
- `aux` (Section 8.4.6)
- `weighted` (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: `auto`]

xN = <num-expr> (*String*)

Horizontal coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

xdeltaN = <num-expr> (*String*)

Vector component in the X direction.

The value is a numeric algebraic expression based on column names as described in Section 10.

yN = <num-expr> (*String*)

Vertical coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

ydeltaN = <num-expr> (*String*)

Vector component in the Y direction.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.5 xyerror

Plots symmetric or asymmetric error bars in some or all of the plot dimensions. The shape of the error "bars" is quite configurable, including (for 2-d and 3-d errors) ellipses, rectangles etc aligned with the axes.

Usage Overview:

```
layerN=xyerror errorbarN=none|lines|capped_lines|...
                shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-param>
```



```

xN=<num-expr> yN=<num-expr> xerrhiN=<num-expr>
xerrloN=<num-expr> yerrhiN=<num-expr> yerrloN=<num-expr>
inN=<table> ifmtN=<in-format> istreamN=true|false
icmdN=<cmds>

```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

errorbarN = *none|lines|capped_lines|...* (*ErrorRenderer*)

How errorbars are represented.

The available options are:

- none
- lines
- capped_lines
- caps
- arrows
- ellipse
- crosshair_ellipse
- rectangle
- crosshair_rectangle
- filled_ellipse
- filled_rectangle

[Default: *lines*]

icmdN = *<cmds>* (*ProcessingStep[]*)

Specifies processing to be performed on the layer *N* input table as specified by parameter *inN*. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file *filename* to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = *<in-format>* (*String*)

Specifies the format of the input table as specified by parameter *inN*. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (*auto*) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (*auto*)]

inN = *<table>* (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the *ifmtN* parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("*<syscmd*" or "*syscmd|*"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

shadingN = auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN> (*ShapeMode*)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- auto (Section 8.4.1)
- flat (Section 8.4.2)
- translucent (Section 8.4.3)
- transparent (Section 8.4.4)
- density (Section 8.4.5)
- aux (Section 8.4.6)
- weighted (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: auto]

xN = <num-expr> (*String*)

Horizontal coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

xerrhiN = <num-expr> (*String*)

Error in the X coordinate in the positive direction. If no corresponding negative error value is supplied, then this value is also used in the negative direction, i.e. in that case errors are assumed to be symmetric.

The value is a numeric algebraic expression based on column names as described in Section 10.

xerrloN = <num-expr> (*String*)

Error in the X coordinate in the negative direction. If left blank, it is assumed to take the same value as the positive error.

The value is a numeric algebraic expression based on column names as described in Section 10.

yN = <num-expr> (*String*)

Vertical coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

yerrhiN = <num-expr> (*String*)

Error in the Y coordinate in the positive direction. If no corresponding negative error value is supplied, then this value is also used in the negative direction, i.e. in that case errors are assumed to be symmetric.

The value is a numeric algebraic expression based on column names as described in Section 10.

yerrloN = <num-expr> (String)

Error in the Y coordinate in the negative direction. If left blank, it is assumed to take the same value as the positive error.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.6 xyellipse

Plots an ellipse (or rectangle, triangle, or other similar figure) defined by two principal radii and an optional rotation angle.

Usage Overview:

```
layerN=xyellipse ellipseN=ellipse|crosshair_ellipse|... scaleN=<factor>
autoscaleN=true|false
shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-par>
xN=<num-expr> yN=<num-expr> raN=<num-expr> rbN=<num-expr>
posangN=<deg-expr> inN=<table> ifmtN=<in-format>
istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

autoscaleN = true|false (Boolean)

Determines whether the default size of variable-sized markers like vectors and ellipses are automatically scaled to have a sensible size. If true, then the sizes of all the plotted markers are examined, and some dynamically calculated factor is applied to them all to make them a sensible size (by default, the largest ones will be a few tens of pixels). If false, the sizes will be the actual input values interpreted in data coordinates.

If auto-scaling is on, then markers will keep approximately the same screen size during zoom operations; if it's off, they will keep the same size in data coordinates.

Marker size is also affected by the `scale` parameter.

[Default: true]

ellipseN = ellipse|crosshair_ellipse|... (ErrorRenderer)

How ellipses are represented.

The available options are:

- ellipse
- crosshair_ellipse
- filled_ellipse
- rectangle
- crosshair_rectangle
- filled_rectangle
- open_triangle
- filled_triangle
- lines
- capped_lines
- arrows

[Default: ellipse]

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (Boolean)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

posangN = <deg-expr> (String)

Orientation of the ellipse. The value is the angle in degrees from the X axis towards the Y axis of the first principal axis of the ellipse.

The value is a numeric algebraic expression based on column names as described in Section 10.

raN = <num-expr> (String)

Ellipse first principal radius.

The value is a numeric algebraic expression based on column names as described in Section 10.

rbN = <num-expr> (String)

Ellipse second principal radius. If this value is blank, the two radii will be assumed equal, i.e. the ellipses will be circles.

The value is a numeric algebraic expression based on column names as described in Section 10.

scaleN = <factor> (*Double*)

Affects the size of variable-sized markers like vectors and ellipses. The default value is 1, smaller or larger values multiply the visible sizes accordingly.

[Default: 1]

shadingN = auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN> (*ShapeMode*)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- auto (Section 8.4.1)
- flat (Section 8.4.2)
- translucent (Section 8.4.3)
- transparent (Section 8.4.4)
- density (Section 8.4.5)
- aux (Section 8.4.6)
- weighted (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: auto]

xN = <num-expr> (*String*)

Horizontal coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

yN = <num-expr> (*String*)

Vertical coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.7 link2

Plots a line linking two positions from the same input table row.

Usage Overview:

```
layerN=link2 shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN>
<pos-coord-params1N> <pos-coord-params2N> inN=<table>
ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

Positional Coordinate Parameters:

The positional coordinates <pos-coord-params1N> , <pos-coord-params2N> give 2 positions for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (plot2plane) the parameters would be x1N, y1N, x2N and y2N. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

shadingN = auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN> (*ShapeMode*)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- `auto` (Section 8.4.1)
- `flat` (Section 8.4.2)
- `translucent` (Section 8.4.3)
- `transparent` (Section 8.4.4)
- `density` (Section 8.4.5)
- `aux` (Section 8.4.6)

- `weighted` (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: `auto`]

8.3.8 `mark2`

Plots 2 similar markers of fixed size and shape representing 2 separate positions from the same input table row.

Usage Overview:

```
layerN=mark2 shapeN=filled_circle|open_circle|... sizeN=<pixels>
              shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-params>
              <pos-coord-params1N> <pos-coord-params2N> inN=<table>
              ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix `N`.

Positional Coordinate Parameters:

The positional coordinates `<pos-coord-params1N>` , `<pos-coord-params2N>` give 2 positions for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (`plot2plane`) the parameters would be `x1N`, `y1N`, `x2N` and `y2N`. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

`icmdN = <cmds> (ProcessingStep[])`

Specifies processing to be performed on the layer `N` input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters ("`;`"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "`@filename`" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

`ifmtN = <in-format> (String)`

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (`auto`) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (`auto`)]

`inN = <table> (StarTable)`

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "`-`", meaning standard input. In this case the input format must be given

explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.

- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

shadingN = auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN> (*ShapeMode*)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- auto (Section 8.4.1)
- flat (Section 8.4.2)
- translucent (Section 8.4.3)
- transparent (Section 8.4.4)
- density (Section 8.4.5)
- aux (Section 8.4.6)
- weighted (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: auto]

shapeN = filled_circle|open_circle|... (*MarkShape*)

Sets the shape of markers that are plotted at each position of the scatter plot.

The available options are:

- filled_circle
- open_circle
- cross
- x
- open_square
- open_diamond
- open_triangle_up
- open_triangle_down
- filled_square
- filled_diamond
- filled_triangle_up
- filled_triangle_down

[Default: filled_circle]

sizeN = <pixels> (*Integer*)

Size of the scatter plot markers. The unit is pixels, in most cases the marker is approximately twice the size of the supplied value.

[Default: 1]

8.3.9 line

Plots a point-to-point line joining up the positions of data points. Note that for a large and unordered data set this can lead to a big scribble on the screen.

Usage Overview:

```
layerN=line colorN=<rrggb>|red|blue|... thickN=<pixels>
dashN=dot|dash|...|<a,b,...> antialiasN=true|false
<pos-coord-paramsN> inN=<table> ifmtN=<in-format>
istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

Positional Coordinate Parameters:

The positional coordinates *<pos-coord-paramsN>* give a position for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (*plot2plane*) the parameters would be *xN* and *yN*. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

antialiasN = true|false (Boolean)

If true, plotted lines are drawn with antialiasing. Antialiased lines look smoother, but may take perceptibly longer to draw. Only has any effect for bitmapped output formats.

[Default: false]

colorN = <rrggb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

dashN = dot|dash|...|<a,b,...> (float[])

Determines the dash pattern of the line drawn. If null (the default), the line is solid.

Possible values for dashed lines are dot, dash, longdash, dotdash. You can alternatively supply a comma-separated list of on/off length values such as "4,2,8,2".

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer *N* input table as specified by parameter *inN*. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file *filename* to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (Boolean)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

thickN = <pixels> (Integer)

Thickness of plotted line in pixels.

[Default: `1`]

8.3.10 linearfit

Plots a line of best fit for the data points.

Usage Overview:

```
layerN=linearfit colorN=<rrggb>|red|blue|... thickN=<pixels>
dashN=dot|dash|...|<a,b,...> antialiasN=true|false
<pos-coord-paramsN> weightN=<num-expr> inN=<table>
ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix `N`.

Positional Coordinate Parameters:

The positional coordinates `<pos-coord-paramsN>` give a position for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (`plot2plane`) the parameters would be `xN` and `yN`. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

antialiasN = true|false (*Boolean*)

If true, plotted lines are drawn with antialiasing. Antialiased lines look smoother, but may take perceptibly longer to draw. Only has any effect for bitmapped output formats.

[Default: false]

colorN = <rrggbb>|red|blue|... (*Color*)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

dashN = dot|dash|...|<a,b,...> (*float[]*)

Determines the dash pattern of the line drawn. If null (the default), the line is solid.

Possible values for dashed lines are dot, dash, longdash, dotdash. You can alternatively supply a comma-separated list of on/off length values such as "4,2,8,2".

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the layer N input table as specified by parameter inN. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file filename to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of the input table as specified by parameter inN. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (auto) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the ifmtN parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("<syscmd" or "syscmd|"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the inN parameter will be read as a stream. It is

necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

thickN = <pixels> (*Integer*)

Thickness of plotted line in pixels.

[Default: 1]

weightN = <num-expr> (*String*)

The weight associated with each data point for fitting purposes. This is used for calculating the coefficients of the line of best fit, and the correlation coefficient. If no coordinate is supplied, all points are assumed to have equal weight (1). Otherwise, any point with a null weight value is assigned a weight of zero, i.e. ignored.

Given certain assumptions about independence of samples, a suitable value for the weight may be $1/(\text{err} * \text{err})$, if `err` is the measurement error for each Y value.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.11 label

Draws a text label at each position. You can select the font, where the labels appear in relation to the point positions, and how crowded the points have to get before they are suppressed.

Usage Overview:

```
layerN=label texttypeN=plain|antialias|latex fontsizeN=<int-value>
fontstyleN=standard|serif|mono
fontweightN=plain|bold|italic|bold_italic
anchorN=west|east|north|south colorN=<rrggb>|red|blue|...
spacingN=<pixels> crowdlimitN=<n> <pos-coord-paramsN>
labelN=<expr> inN=<table> ifmtN=<in-format> istreamN=true|false
icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix `N`.

Positional Coordinate Parameters:

The positional coordinates `<pos-coord-paramsN>` give a position for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (`plot2plane`) the parameters would be `xN` and `yN`. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

anchorN = west|east|north|south (*Anchor*)

Determines where the text appears in relation to the plotted points. Values are points of the compass.

The available options are:

- west
- east
- north

- south

[Default: west]

colorN = <rrggbb>|red|blue|... (*Color*)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

crowdlimitN = <n> (*Integer*)

Sets the maximum number of labels in a label group. This many labels can appear closely spaced without being affected by the label spacing parameter.

It is useful for instance if you are looking at pairs of points, which will always be close together; if you set this value to 2, an isolated pair of labels can be seen, but if it's 1 then they will only be plotted when they are distant from each other, which may only happen at very high magnifications.

[Default: 2]

fontsizeN = <int-value> (*Integer*)

Size of the text font in points.

[Default: 12]

fontstyleN = standard|serif|mono (*FontType*)

Font style for text.

The available options are:

- standard
- serif
- mono

[Default: standard]

fontweightN = plain|bold|italic|bold_italic (*FontWeight*)

Font weight for text.

The available options are:

- plain
- bold
- italic
- bold_italic

[Default: plain]

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the layer N input table as specified by parameter inN. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file filename to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters

and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

labelN = <expr> (*String*)

Column or expression giving the text of the label to be written near the position being labelled. Label values may be of any type (string or numeric)

The value is a `Object` algebraic expression based on column names as described in Section 10.

spacingN = <pixels> (*Integer*)

Determines the closest that labels can be spaced. If a group of labels is closer to another group than the value of this parameter, they will not be drawn, to avoid the display becoming too cluttered. The effect is that you can see individual labels when you zoom in, but not when there are many labelled points plotted close together on the screen. Set the value higher for less cluttered labelling.

[Default: `12`]

texttypeN = plain|antialias|latex (*TextSyntax*)

Determines how to turn label text into characters on the plot. `Plain` and `Antialias` both take the text at face value, but `Antialias` smooths the characters. `LaTeX` interprets the text as LaTeX source code and typesets it accordingly.

When not using LaTeX, antialiased text usually looks nicer, but can be perceptibly slower to plot. At time of writing, on MacOS antialiased text seems to be required to stop the writing coming out upside-down for non-horizontal text (MacOS java bug).

[Default: `plain`]

8.3.12 contour

Plots position density contours. This provides another way (alongside the auto and density shading modes) to visualise the characteristics of overdense regions in a crowded plot. It's not very useful if you just have a few points.

The contours are currently drawn as pixels rather than lines so they don't look very beautiful in exported vector output formats (PDF, PostScript). This may be improved in the future.

Usage Overview:

```
layerN=contour colorN=<rrggb>|red|blue|... nlevelN=<int-value>
              smoothN=<pixels> scalingN=linear|log|equal zeroN=<number>
              <pos-coord-paramsN> inN=<table> ifmtN=<in-format>
              istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

Positional Coordinate Parameters:

The positional coordinates *<pos-coord-paramsN>* give a position for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (*plot2plane*) the parameters would be *xN* and *yN*. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

colorN = <rrggb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer *N* input table as specified by parameter *inN*. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file *filename* to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter *inN*. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (auto) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

nlevelN = <int-value> (*Integer*)

Number of countour lines drawn. In fact, this is an upper limit; if there is not enough variation in the plot's density, then fewer conrour lines will be drawn.

[Default: 5]

scalingN = linear|log|equal (*LevelMode*)

How the smoothed density is treated before contour levels are determined.

The available options are:

- `linear`: levels are equally spaced
- `log`: level logarithms are equally spaced
- `equal`: levels are spaced to provide equal-area inter-contour regions

[Default: linear]

smoothN = <pixels> (*Integer*)

The size of the smoothing kernel applied to the density before performing the contour determination. If set too low the contours will be too crinkly, and if too high they will lose definition.

[Default: 4]

zeroN = <number> (*Double*)

Determines the level at which the first contour (and hence all the others, which are separated from it by a fixed amount) are drawn.

[Default: 0]

8.3.13 density

Plots a density map on the pixel grid of the plot surface, coarsened by a configurable factor. You can optionally use a weighting for the points, and you can configure how the points are combined to produce the output pixel values.

The way that data values are mapped to colours is usually controlled by options at the level of the plot itself, rather than by per-layer configuration.

Usage Overview:

```
layerN=density binpixN=<int-value> combineN=<value> transparencyN=0..1
      <pos-coord-paramsN> weightN=<num-expr> inN=<table>
      ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

Positional Coordinate Parameters:

The positional coordinates *<pos-coord-paramsN>* give a position for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (*plot2plane*) the parameters would be *xN* and *yN*. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

binpixN = <int-value> *(Integer)*

Determines the dimension of grid bins in pixels. Bins are square in pixel dimensions, and this parameter gives the extent in pixels along each side. Currently, only integer values are allowed.

[Default: 2]

combineN = <value> *(Combiner)*

Defines how values contributing to the same density map bin are combined together to produce the value assigned to that bin (and hence its colour).

For unweighted values (a pure density map), it usually makes sense to use *count*. However, if the input is weighted by an additional data coordinate, one of the other values such as *mean* may be more revealing.

[Default: *sum*]

icmdN = <cmds> *(ProcessingStep[])*

Specifies processing to be performed on the layer *N* input table as specified by parameter *inN*. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file *filename* to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> *(String)*

Specifies the format of the input table as specified by parameter *inN*. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (*auto*) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (*auto*)]

inN = <table> *(StarTable)*

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

transparencyN = 0..1 (*Double*)

Transparency with which components are plotted, in the range 0 (opaque) to 1 (invisible). The value is 1-alpha.

[Default: 0]

weightN = <num-expr> (*String*)

Weighting of data points. If supplied, each point contributes a value to the histogram equal to the data value multiplied by this coordinate. If not supplied, the effect is the same as supplying a fixed value of one.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.14 fill

If a two-dimensional dataset represents a single-valued function, this plotter will fill the area underneath the function's curve with a solid colour. Parts of the surface which would only be partially covered (because of rapid function variation within the width of a single pixel) are represented using appropriate alpha-blending. The filled area may alternatively be that above the curve or to its left or right.

One example of its use is to reconstruct the appearance of a histogram plot from a set of histogram bins. For X,Y data which is not single-valued, the result may not be very useful.

Usage Overview:

```
layerN=fill colorN=<rrgbb>|red|blue|... transparencyN=0..1
          horizontalN=true|false positiveN=true|false <pos-coord-paramsN>
          inN=<table> ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix `N`.

Positional Coordinate Parameters:

The positional coordinates `<pos-coord-paramsN>` give a position for each row of the input table. Their form depends on the plot geometry, i.e. which plotting command is used. For a plane plot (`plot2plane`) the parameters would be `xN` and `yN`. The coordinate parameter values are in all cases strings interpreted as numeric expressions based on column names. These can be column names, fixed values or algebraic expressions as described in Section 10.

colorN = <rrggb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

horizontalN = true|false (Boolean)

Determines whether the filling is vertical (suitable for functions of the horizontal variable), or horizontal (suitable for functions of the vertical variable). If false, the fill is vertical (to the X axis), and if true, the fill is horizontal (to the Y axis).

[Default: false]

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (`auto`) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (`auto`)]

inN = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

positiveN = true|false (*Boolean*)

Determines the directional sense of the filling. If false, the fill is between the data points and negative infinity along the relevant axis (e.g. down from the data points to the bottom of the plot). If true, the fill is in the other direction.

[Default: false]

transparencyN = 0..1 (*Double*)

Transparency with which components are plotted, in the range 0 (opaque) to 1 (invisible). The value is 1-alpha.

[Default: 0]

8.3.15 histogram

Plots a histogram.

Usage Overview:

```
layerN=histogram colorN=<rrggb>|red|blue|... transparencyN=0..1
               binsizeN=+<width>|-<count> phaseN=<number>
               cumulativeN=true|false normaliseN=none|area|maximum|height
               barformN=open|filled|semi_filled|steps|semi_steps|spikes
               thickN=<pixels> dashN=dot|dash|...|<a,b,...> xN=<num-expr>
               weightN=<num-expr> inN=<table> ifmtN=<in-format>
               istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix `N`.

barformN = open|filled|semi_filled|steps|semi_steps|spikes (*Form*)

How histogram bars are represented. Note that options using transparent colours may not render very faithfully to some vector formats like PDF and EPS.

The available options are:

- open
- filled
- semi_filled
- steps
- semi_steps
- spikes

[Default: semi_filled]

binsizeN = +<width>|-<count> (*BinSizer*)

Configures the width of histogram bins. If the supplied string is a positive number, it is interpreted as a fixed width in the data coordinates of the X axis (if the X axis is logarithmic, the value is a fixed factor). If it is a negative number, then it will be interpreted as the approximate number of bins to display across the width of the plot (though an attempt is made to use only round numbers for bin widths).

When setting this value graphically, you can use either the slider to adjust the bin count or the numeric entry field to fix the bin width.

[Default: -30]

colorN = <rrggbb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

cumulativeN = true|false (Boolean)

If true, the histogram bars plotted are calculated cumulatively; each bin includes the counts from all previous bins.

[Default: false]

dashN = dot|dash|...|<a,b,...> (float[])

Determines the dash pattern of the line drawn. If null (the default), the line is solid.

Possible values for dashed lines are dot, dash, longdash, dotdash. You can alternatively supply a comma-separated list of on/off length values such as "4,2,8,2".

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer N input table as specified by parameter inN. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file filename to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter inN. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (auto) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the ifmtN parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("<syscmd" or "syscmd|"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

normaliseN = none|area|maximum|height (*Normalisation*)

Defines how, if at all, the bars of histogram-like plots are normalised.

The available options are:

- `none`: No normalisation is performed.
- `area`: The total area of histogram bars is normalised to unity. For cumulative plots, this behaves like `height`.
- `maximum`: The height of the tallest histogram bar is normalised to unity. For cumulative plots, this behaves like `height`.
- `height`: The total height of histogram bars is normalised to unity.

[Default: none]

phaseN = <number> (*Double*)

Controls where the horizontal zero point for binning is set. For instance if your bin size is 1, this value controls whether bin boundaries are at 0, 1, 2, .. or 0.5, 1.5, 2.5, ... etc.

A value of 0 (or any integer) will result in a bin boundary at $X=0$ (linear X axis) or $X=1$ (logarithmic X axis). A fractional value will give a bin boundary at that value multiplied by the bin width.

[Default: 0]

thickN = <pixels> (*Integer*)

Thickness of plotted line in pixels.

[Default: 2]

transparencyN = 0..1 (*Double*)

Transparency with which components are plotted, in the range 0 (opaque) to 1 (invisible). The value is 1-alpha.

[Default: 0]

weightN = <num-expr> (*String*)

Weighting of data points. If supplied, each point contributes a value to the histogram equal to the data value multiplied by this coordinate. If not supplied, the effect is the same as supplying a fixed value of one.

The value is a numeric algebraic expression based on column names as described in Section 10.

xN = <num-expr> (*String*)

Horizontal coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.16 kde

Plots a Discrete Kernel Density Estimate giving a smoothed frequency of data values along the

horizontal axis, using a fixed-width smoothing kernel. This is a generalisation of a histogram in which the bins are always 1 pixel wide, and a smoothing kernel is applied to each bin. The width and shape of the kernel may be varied.

This is suitable for cases where the division into discrete bins done by a normal histogram is unnecessary or troublesome.

Note this is not a true Kernel Density Estimate, since, for performance reasons, the smoothing is applied to the (pixel-width) bins rather than to each data sample. The deviation from a true KDE caused by this quantisation will be at the pixel level, hence in most cases not visually apparent.

Usage Overview:

```
layerN=kde colorN=<rrgbb>|red|blue|... transparencyN=0..1
smoothN=+<width>|-<count>
kernelN=square|linear|epanechnikov|cos|cos2|gauss3|gauss6
cumulativeN=true|false normaliseN=none|area|maximum|height
fillN=solid|line|semi thickN=<pixels> xN=<num-expr>
weightN=<num-expr> inN=<table> ifmtN=<in-format>
istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

colorN = <rrgbb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

cumulativeN = true|false (Boolean)

If true, the histogram bars plotted are calculated cumulatively; each bin includes the counts from all previous bins.

[Default: false]

fillN = solid|line|semi (FillMode)

How the density function is represented.

The available options are:

- **solid**: area between level and axis is filled with solid colour
- **line**: level is marked by a wiggly line
- **semi**: level is marked by a wiggly line, and area below it is filled with a transparent colour

[Default: semi]

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer *N* input table as specified by parameter *inN*. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file *filename* to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters

and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (Boolean)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

kernelN = square|linear|epanechnikov|cos|cos2|gauss3|gauss6 (KernelIdShape)

The functional form of the smoothing kernel. The functions listed refer to the unscaled shape; all kernels are normalised to give a total area of unity.

The available options are:

- `square`: Uniform value: $f(x)=1$, $|x|=0..1$
- `linear`: Triangle: $f(x)=1-|x|$, $|x|=0..1$
- `epanechnikov`: Parabola: $f(x)=1-x^2$, $|x|=0..1$
- `cos`: Cosine: $f(x)=\cos(x\pi/2)$, $|x|=0..1$
- `cos2`: Cosine squared: $f(x)=\cos^2(x\pi/2)$, $|x|=0..1$
- `gauss3`: Gaussian truncated at 3.0 sigma: $f(x)=\exp(-x^2/2)$, $|x|=0..3$
- `gauss6`: Gaussian truncated at 6.0 sigma: $f(x)=\exp(-x^2/2)$, $|x|=0..6$

[Default: `epanechnikov`]

normaliseN = none|area|maximum|height (Normalisation)

Defines how, if at all, the bars of histogram-like plots are normalised.

The available options are:

- `none`: No normalisation is performed.
- `area`: The total area of histogram bars is normalised to unity. For cumulative plots, this behaves like `height`.
- `maximum`: The height of the tallest histogram bar is normalised to unity. For cumulative plots, this behaves like `height`.

- **height**: The total height of histogram bars is normalised to unity.

[Default: none]

smoothN = +<width>|-<count> *(BinSizer)*

Configures the smoothing width for kernel density estimation. This is the characteristic width of the kernel function to be convolved with the density to produce the visible plot.

If the supplied value is a positive number it is interpreted as a fixed width in the data coordinates of the X axis (if the X axis is logarithmic, the value is a fixed factor). If it is a negative number, then it will be interpreted as the approximate number of smoothing widths that fit in the width of the visible plot (i.e. plot width / smoothing width). If the value is zero, no smoothing is applied.

When setting this value graphically, you can use either the slider to adjust the bin count or the numeric entry field to fix the bin width.

[Default: -100]

thickN = <pixels> *(Integer)*

Thickness of plotted line in pixels.

[Default: 2]

transparencyN = 0..1 *(Double)*

Transparency with which components are plotted, in the range 0 (opaque) to 1 (invisible). The value is 1-alpha.

[Default: 0]

weightN = <num-expr> *(String)*

Weighting of data points. If supplied, each point contributes a value to the histogram equal to the data value multiplied by this coordinate. If not supplied, the effect is the same as supplying a fixed value of one.

The value is a numeric algebraic expression based on column names as described in Section 10.

xN = <num-expr> *(String)*

Horizontal coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.17 knn

Plots a Discrete Kernel Density Estimate giving a smoothed frequency of data values along the horizontal axis, using an adaptive (K-Nearest-Neighbours) smoothing kernel. This is a generalisation of a histogram in which the bins are always 1 pixel wide, and a smoothing kernel is applied to each bin. The width and shape of the kernel may be varied.

The K-Nearest-Neighbour figure gives the number of points in each direction to determine the width of the smoothing kernel for smoothing each bin. Upper and lower limits for the kernel width are also supplied; if the upper and lower limits are equal, this is equivalent to a fixed-width kernel.

Note this is not a true Kernel Density Estimate, since, for performance reasons, the smoothing is applied to the (pixel-width) bins rather than to each data sample. The deviation from a true KDE caused by this quantisation will be at the pixel level, hence in most cases not visually apparent.

Usage Overview:

```

layerN=knn colorN=<rrgbb>|red|blue|... transparencyN=0..1 knnN=<number>
symmetricN=true|false minsmoothN=+<width>|-<count>
maxsmoothN=+<width>|-<count>
kernelN=square|linear|epanechnikov|cos|cos2|gauss3|gauss6
cumulativeN=true|false normaliseN=none|area|maximum|height
fillN=solid|line|semi thickN=<pixels> xN=<num-expr>
weightN=<num-expr> inN=<table> ifmtN=<in-format>
istreamN=true|false icmdN=<cmds>

```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

colorN = <rrgbb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

cumulativeN = true|false (Boolean)

If true, the histogram bars plotted are calculated cumulatively; each bin includes the counts from all previous bins.

[Default: false]

fillN = solid|line|semi (FillMode)

How the density function is represented.

The available options are:

- **solid**: area between level and axis is filled with solid colour
- **line**: level is marked by a wiggly line
- **semi**: level is marked by a wiggly line, and area below it is filled with a transparent colour

[Default: semi]

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer N input table as specified by parameter inN. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file filename to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter inN. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (auto) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

kernelN = square|linear|epanechnikov|cos|cos2|gauss3|gauss6 (*KernelIdShape*)

The functional form of the smoothing kernel. The functions listed refer to the unscaled shape; all kernels are normalised to give a total area of unity.

The available options are:

- square: Uniform value: $f(x)=1$, $|x|=0..1$
- linear: Triangle: $f(x)=1-|x|$, $|x|=0..1$
- epanechnikov: Parabola: $f(x)=1-x^2$, $|x|=0..1$
- cos: Cosine: $f(x)=\cos(x\pi/2)$, $|x|=0..1$
- cos2: Cosine squared: $f(x)=\cos^2(x\pi/2)$, $|x|=0..1$
- gauss3: Gaussian truncated at 3.0 sigma: $f(x)=\exp(-x^2/2)$, $|x|=0..3$
- gauss6: Gaussian truncated at 6.0 sigma: $f(x)=\exp(-x^2/2)$, $|x|=0..6$

[Default: epanechnikov]

knnN = <number> (*Double*)

Sets the number of nearest neighbours to count away from a sample point to determine the width of the smoothing kernel at that point. For the symmetric case this is the number of nearest neighbours summed over both directions, and for the asymmetric case it is the number in a single direction.

The threshold is actually the weighted total of samples; for unweighted (`weight=1`) bins that is equivalent to the number of samples.

[Default: 100]

maxsmoothN = +<width>|-<count> (*BinSizer*)

Fixes the maximum size of the smoothing kernel. This functions as an upper limit on the distance that is otherwise determined by searching for the K nearest neighbours at each sample point.

If the supplied value is a positive number it is interpreted as a fixed width in the data coordinates of the X axis (if the X axis is logarithmic, the value is a fixed factor). If it is a negative number, then it will be interpreted as the approximate number of smoothing widths that fit in the width of the visible plot (i.e. plot width / smoothing width). If the value is zero, no smoothing is applied.

When setting this value graphically, you can use either the slider to adjust the bin count or the numeric entry field to fix the bin width.

[Default: -100]

minsmoothN = **+<width>|-<count>** (*BinSizer*)

Fixes the minimum size of the smoothing kernel. This functions as a lower limit on the distance that is otherwise determined by searching for the K nearest neighbours at each sample point.

If the supplied value is a positive number it is interpreted as a fixed width in the data coordinates of the X axis (if the X axis is logarithmic, the value is a fixed factor). If it is a negative number, then it will be interpreted as the approximate number of smoothing widths that fit in the width of the visible plot (i.e. plot width / smoothing width). If the value is zero, no smoothing is applied.

When setting this value graphically, you can use either the slider to adjust the bin count or the numeric entry field to fix the bin width.

[Default: 0]

normaliseN = **none|area|maximum|height** (*Normalisation*)

Defines how, if at all, the bars of histogram-like plots are normalised.

The available options are:

- **none**: No normalisation is performed.
- **area**: The total area of histogram bars is normalised to unity. For cumulative plots, this behaves like **height**.
- **maximum**: The height of the tallest histogram bar is normalised to unity. For cumulative plots, this behaves like **height**.
- **height**: The total height of histogram bars is normalised to unity.

[Default: none]

symmetricN = **true|false** (*Boolean*)

If true, the nearest neighbour search is carried out in both directions, and the kernel is symmetric. If false, the nearest neighbour search is carried out separately in the positive and negative directions, and the kernel width is accordingly different in the positive and negative directions.

[Default: true]

thickN = **<pixels>** (*Integer*)

Thickness of plotted line in pixels.

[Default: 2]

transparencyN = **0..1** (*Double*)

Transparency with which components are plotted, in the range 0 (opaque) to 1 (invisible). The value is 1-alpha.

[Default: 0]

weightN = **<num-expr>** (*String*)

Weighting of data points. If supplied, each point contributes a value to the histogram equal to the data value multiplied by this coordinate. If not supplied, the effect is the same as supplying a fixed value of one.

The value is a numeric algebraic expression based on column names as described in Section 10.

xN = **<num-expr>** (*String*)

Horizontal coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.18 densogram

Represents smoothed density of data values along the horizontal axis using a colourmap. This is like a Kernel Density Estimate (smoothed histogram with bins 1 pixel wide), but instead of representing the data extent vertically as bars or a line, values are represented by a fixed-size pixel-width column of a colour from a colour map. A smoothing kernel, whose width and shape may be varied, is applied to each data point.

This is a rather unconventional way to represent density data, and this plotting mode is probably not very useful. But hey, nobody's forcing you to use it.

Usage Overview:

```
layerN=densogram colorN=<rrggb>|red|blue|... smoothN=+<width>|-<count>
kernelN=square|linear|epanechnikov|cos|cos2|gauss3|gauss6
densemapN=inferno|magma|plasma|... denseclipN=<lo>,<hi>
denseflipN=true|false densequantN=<number>
densefuncN=log|linear|sqrt|square densesubN=<lo>,<hi>
cumulativeN=true|false sizeN=<pixels> posN=<fraction>
xN=<num-expr> weightN=<num-expr> inN=<table>
ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

colorN = <rrggb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

cumulativeN = true|false (Boolean)

If true, the histogram bars plotted are calculated cumulatively; each bin includes the counts from all previous bins.

[Default: false]

denseclipN = <lo>,<hi> (Subrange)

Defines a subrange of the colour ramp to be used for Density shading. The is specified as a (low,high) comma-separated pair of two numbers between 0 and 1.

If the full range 0,1 (the default) is used, the whole range of colours specified by the selected shader will be used. But if, for instance a value of 0,0.5 is given, only those colours at the left hand end of the ramp will be seen.

[Default: 0,1]

denseflipN = true|false (Boolean)

If true, the colour map on the Density axis will be reversed.

[Default: false]

densefuncN = log|linear|sqrt|square (Scaling)

Defines the way that values in the Density range are mapped to the selected colour ramp.

The available options are:

- log: Logarithmic scaling
- linear: Linear scaling

- `sqrt`: Square root scaling
- `square`: Square scaling

[Default: `linear`]

densemapN = `inferno|magma|plasma|...` (*Shader*)

Color map used for Density axis shading.

A mixed bag of colour ramps are available: `inferno`, `magma`, `plasma`, `viridis`, `cubehelix`, `sron`, `rainbow`, `rainbow2`, `rainbow3`, `pastel`, `accent`, `gnuplot`, `gnuplot2`, `specxby`, `set1`, `paired`, `hotcold`, `rdbu`, `piyg`, `brbg`, `cyan-magenta`, `red-blue`, `brg`, `heat`, `cold`, `light`, `greyscale`, `colour`, `standard`, `bugn`, `bupu`, `orrd`, `pubu`, `purd`, `huecl`, `hue`, `intensity`, `rgb_red`, `rgb_green`, `rgb_blue`, `hsv_h`, `hsv_s`, `hsv_v`, `yuv_y`, `yuv_u`, `yuv_v`, `scale_hsv_s`, `scale_hsv_v`, `scale_yuv_y`, `mask`, `blackier`, `whiter`, `transparency`. *Note*: many of these, including rainbow-like ones, are frowned upon by the visualisation community.

[Default: `inferno`]

densequantN = `<number>` (*Double*)

Allows the colour map used for the Density axis to be quantised. If an integer value N is chosen then the colour map will be viewed as N discrete evenly-spaced levels, so that only N different colours will appear in the plot. This can be used to generate a contour-like effect, and may make it easier to trace the boundaries of regions of interest by eye.

If left blank, the colour map is nominally continuous (though in practice it may be quantised to a medium-sized number like 256).

densesubN = `<lo>,<hi>` (*Subrange*)

Defines a normalised adjustment to the data range of the Density axis. The value may be specified as a comma-separated pair of two numbers, giving the lower and upper bounds of the range of of interest respectively. This sub-range is applied to the data range that would otherwise be used, either automatically calculated or explicitly supplied; zero corresponds to the lower bound and one to the upper.

The default value "0,1" therefore has no effect. The range could be restricted to its lower half with the value 0,0.5.

[Default: 0,1]

icmdN = `<cmds>` (*ProcessingStep[]*)

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = `<in-format>` (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (`auto`) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (`auto`)]

inN = `<table>` (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

kernelN = square|linear|epanechnikov|cos|cos2|gauss3|gauss6 (*KernelIdShape*)

The functional form of the smoothing kernel. The functions listed refer to the unscaled shape; all kernels are normalised to give a total area of unity.

The available options are:

- `square`: Uniform value: $f(x)=1$, $|x|=0..1$
- `linear`: Triangle: $f(x)=1-|x|$, $|x|=0..1$
- `epanechnikov`: Parabola: $f(x)=1-x^2$, $|x|=0..1$
- `cos`: Cosine: $f(x)=\cos(x\pi/2)$, $|x|=0..1$
- `cos2`: Cosine squared: $f(x)=\cos^2(x\pi/2)$, $|x|=0..1$
- `gauss3`: Gaussian truncated at 3.0 sigma: $f(x)=\exp(-x^2/2)$, $|x|=0..3$
- `gauss6`: Gaussian truncated at 6.0 sigma: $f(x)=\exp(-x^2/2)$, $|x|=0..6$

[Default: epanechnikov]

posN = <fraction> (*Double*)

Determines where on the plot region the density bar appears. The value should be in the range 0..1; zero corresponds to the bottom of the plot and one to the top.

[Default: 0.05]

sizeN = <pixels> (*Integer*)

Height of the density bar in pixels.

[Default: 12]

smoothN = +<width>|-<count> (*BinSizer*)

Configures the smoothing width for kernel density estimation. This is the characteristic width of the kernel function to be convolved with the density to produce the visible plot.

If the supplied value is a positive number it is interpreted as a fixed width in the data coordinates of the X axis (if the X axis is logarithmic, the value is a fixed factor). If it is a negative number, then it will be interpreted as the approximate number of smoothing widths that fit in the width of the visible plot (i.e. plot width / smoothing width). If the value is zero, no smoothing is applied.

When setting this value graphically, you can use either the slider to adjust the bin count or the numeric entry field to fix the bin width.

[Default: -100]

weightN = <num-expr> (String)

Weighting of data points. If supplied, each point contributes a value to the histogram equal to the data value multiplied by this coordinate. If not supplied, the effect is the same as supplying a fixed value of one.

The value is a numeric algebraic expression based on column names as described in Section 10.

xN = <num-expr> (String)

Horizontal coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.19 function

Plots an analytic function. This layer is currently only available for the Plane plots (including histogram).

Usage Overview:

```
layerN=function axisN=Horizontal|Vertical xnameN=<name> fexprN=<expr>
              colorN=<rrggbb>|red|blue|... thickN=<pixels>
              dashN=dot|dash|...|<a,b,...> antialiasN=true|false
```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

antialiasN = true|false (Boolean)

If true, plotted lines are drawn with antialiasing. Antialiased lines look smoother, but may take perceptibly longer to draw. Only has any effect for bitmapped output formats.

[Default: false]

axisN = Horizontal|Vertical (FuncAxis)

Which axis the independent variable varies along. Options are currently Horizontal and Vertical.

[Default: Horizontal]

colorN = <rrggbb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

dashN = dot|dash|...|<a,b,...> (float[])

Determines the dash pattern of the line drawn. If null (the default), the line is solid.

Possible values for dashed lines are dot, dash, longdash, dotdash. You can alternatively supply a comma-separated list of on/off length values such as "4,2,8,2".

fexprN = <expr> (String)

An expression using TOPCAT's expression language in terms of the independent variable to define the function. This expression must be standalone - it cannot reference any tables.

thickN = <pixels> (Integer)

Thickness of plotted line in pixels.

[Default: 1]

xnameN = <name> (*String*)

Name of the independent variable for use in the function expression. This is typically *x* for a horizontal independent variable and *y* for a vertical independent variable, but any string that is a legal expression language identifier (starts with a letter, continues with letters, numbers, underscores) can be used.

[Default: *x*]

8.3.20 skyvector

Plots directed lines from the data position given delta values for the coordinates. The plotted markers are typically little arrows, but there are other options.

In some cases such delta values may be the actual magnitude required for the plot, but often the vector data represents a value which has a different magnitude or is in different units to the positional data. As a convenience for this case, the plotter can optionally scale the magnitudes of all the vectors to make them a sensible size, so by default the largest ones are a few tens of pixels long. This auto-scaling is in operation by default, but it can be turned off or adjusted with the scaling and auto-scaling options.

Usage Overview:

```
layerN=skyvector arrowN=small_arrow|medium_arrow|... scaleN=<factor>
               autoscaleN=true|false
               shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-par
               lonN=<deg-expr> latN=<deg-expr> dlonN=<deg-expr>
               dlatN=<deg-expr> inN=<table> ifmtN=<in-format>
               istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

arrowN = small_arrow|medium_arrow|... (*ErrorRenderer*)

How arrows are represented.

The available options are:

- small_arrow
- medium_arrow
- large_arrow
- small_open_dart
- medium_open_dart
- large_open_dart
- small_filled_dart
- medium_filled_dart
- large_filled_dart
- lines
- capped_lines

[Default: small_arrow]

autoscaleN = true|false (*Boolean*)

Determines whether the default size of variable-sized markers like vectors and ellipses are automatically scaled to have a sensible size. If true, then the sizes of all the plotted markers are examined, and some dynamically calculated factor is applied to them all to make them a

sensible size (by default, the largest ones will be a few tens of pixels). If false, the sizes will be the actual input values interpreted in data coordinates.

If auto-scaling is on, then markers will keep approximately the same screen size during zoom operations; if it's off, they will keep the same size in data coordinates.

Marker size is also affected by the `scale` parameter.

[Default: `true`]

dlatN = `<deg-expr>` (*String*)

Change in the latitude coordinate represented by the plotted vector. The supplied value is an angle in degrees.

The value is a numeric algebraic expression based on column names as described in Section 10.

dlonN = `<deg-expr>` (*String*)

Change in the longitude coordinate represented by the plotted vector. The supplied value is an angle in degrees, and is considered to be premultiplied by $\cos(\text{Latitude})$.

The value is a numeric algebraic expression based on column names as described in Section 10.

icmdN = `<cmds>` (*ProcessingStep[]*)

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = `<in-format>` (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = `<table>` (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = `true|false` (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is

necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

latN = <deg-expr> (String)

Latitude in decimal degrees.

The value is a numeric algebraic expression based on column names as described in Section 10.

lonN = <deg-expr> (String)

Longitude in decimal degrees.

The value is a numeric algebraic expression based on column names as described in Section 10.

scaleN = <factor> (Double)

Affects the size of variable-sized markers like vectors and ellipses. The default value is 1, smaller or larger values multiply the visible sizes accordingly.

[Default: 1]

shadingN = auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN> (ShapeMode)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- auto (Section 8.4.1)
- flat (Section 8.4.2)
- translucent (Section 8.4.3)
- transparent (Section 8.4.4)
- density (Section 8.4.5)
- aux (Section 8.4.6)
- weighted (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: `auto`]

8.3.21 skyellipse

Plots an ellipse (or rectangle, triangle, or other similar figure) defined by two principal radii and an optional rotation angle.

Usage Overview:

```
layerN=skyellipse ellipseN=ellipse|crosshair_ellipse|... scaleN=<factor>
autoscaleN=true|false
shadingN=auto|flat|translucent|transparent|density|aux|weighted <shade-paramsN>
lonN=<deg-expr> latN=<deg-expr> raN=<deg-expr>
rbN=<deg-expr> posangN=<deg-expr> inN=<table>
ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix `N`.

autoscaleN = true|false (*Boolean*)

Determines whether the default size of variable-sized markers like vectors and ellipses are automatically scaled to have a sensible size. If true, then the sizes of all the plotted markers are examined, and some dynamically calculated factor is applied to them all to make them a sensible size (by default, the largest ones will be a few tens of pixels). If false, the sizes will be the actual input values interpreted in data coordinates.

If auto-scaling is on, then markers will keep approximately the same screen size during zoom operations; if it's off, they will keep the same size in data coordinates.

Marker size is also affected by the `scale` parameter.

[Default: true]

ellipseN = ellipse|crosshair_ellipse|... (*ErrorRenderer*)

How ellipses are represented.

The available options are:

- ellipse
- crosshair_ellipse
- filled_ellipse
- rectangle
- crosshair_rectangle
- filled_rectangle
- open_triangle
- filled_triangle
- lines
- capped_lines
- arrows

[Default: ellipse]

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given

explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.

- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = `true|false` (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

latN = `<deg-expr>` (*String*)

Latitude in decimal degrees.

The value is a numeric algebraic expression based on column names as described in Section 10.

lonN = `<deg-expr>` (*String*)

Longitude in decimal degrees.

The value is a numeric algebraic expression based on column names as described in Section 10.

posangN = `<deg-expr>` (*String*)

Orientation of the ellipse. The value is the angle in degrees from the North pole to the primary axis of the ellipse in the direction of increasing longitude.

The value is a numeric algebraic expression based on column names as described in Section 10.

raN = `<deg-expr>` (*String*)

Ellipse first principal radius in degrees.

The value is a numeric algebraic expression based on column names as described in Section 10.

rbN = `<deg-expr>` (*String*)

Ellipse second principal radius in degrees. If this value is blank, the two radii will be assumed equal, i.e. the ellipses will be circles.

The value is a numeric algebraic expression based on column names as described in Section 10.

scaleN = `<factor>` (*Double*)

Affects the size of variable-sized markers like vectors and ellipses. The default value is 1, smaller or larger values multiply the visible sizes accordingly.

[Default: 1]

shadingN = `auto|flat|translucent|transparent|density|aux|weighted` `<shade-paramsN>` (*ShapeMode*)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- `auto` (Section 8.4.1)

- flat (Section 8.4.2)
- translucent (Section 8.4.3)
- transparent (Section 8.4.4)
- density (Section 8.4.5)
- aux (Section 8.4.6)
- weighted (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: auto]

8.3.22 skydensity

Plots a density map on the sky. The grid on which the values are drawn uses the HEALPix tessellation, with a configurable resolution. You can optionally use a weighting for the points, and you can configure how the points are combined to produce the output pixel values.

The way that data values are mapped to colours is usually controlled by options at the level of the plot itself, rather than by per-layer configuration.

Usage Overview:

```
layerN=skydensity levelN=<-rel-level|+abs-level> combineN=<value>
transparencyN=0..1 lonN=<deg-expr> latN=<deg-expr>
weightN=<num-expr> inN=<table> ifmtN=<in-format>
istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

combineN = <value> (Combiner)

Defines how values contributing to the same density map bin are combined together to produce the value assigned to that bin (and hence its colour).

For unweighted values (a pure density map), it usually makes sense to use `count`. However, if the input is weighted by an additional data coordinate, one of the other values such as `mean` may be more revealing.

[Default: sum]

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer *N* input table as specified by parameter *inN*. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter *inN*. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program

will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

latN = <deg-expr> (*String*)

Latitude in decimal degrees.

The value is a numeric algebraic expression based on column names as described in Section 10.

levelN = <-rel-level|+abs-level> (*Integer*)

Determines the HEALPix level of pixels which are averaged over to calculate density.

If the supplied value is a non-negative integer, it gives the absolute level to use; at level 0 there are 12 pixels on the sky, and the count multiplies by 4 for each increment.

If the value is negative, it represents a relative level; it is approximately the (negative) number of screen pixels along one side of a HEALPix sky pixel. In this case the actual HEALPix level will depend on the current zoom.

[Default: -3]

lonN = <deg-expr> (*String*)

Longitude in decimal degrees.

The value is a numeric algebraic expression based on column names as described in Section 10.

transparencyN = 0..1 (*Double*)

Transparency with which components are plotted, in the range 0 (opaque) to 1 (invisible). The value is 1-alpha.

[Default: 0]

weightN = <num-expr> (*String*)

Weighting of data points. If supplied, each point contributes a value to the histogram equal to the data value multiplied by this coordinate. If not supplied, the effect is the same as supplying a fixed value of one.

The value is a numeric algebraic expression based on column names as described in Section

10.

8.3.23 xyzvector

Plots directed lines from the data position given delta values for the coordinates. The plotted markers are typically little arrows, but there are other options.

In some cases such delta values may be the actual magnitude required for the plot, but often the vector data represents a value which has a different magnitude or is in different units to the positional data. As a convenience for this case, the plotter can optionally scale the magnitudes of all the vectors to make them a sensible size, so by default the largest ones are a few tens of pixels long. This auto-scaling is in operation by default, but it can be turned off or adjusted with the scaling and auto-scaling options.

Usage Overview:

```
layerN=xyzvector arrowN=small_arrow|medium_arrow|... scaleN=<factor>
                autoscaleN=true|false
                shadingN=flat|translucent|transparent|density|aux|weighted <shade-paramsN>
                xN=<num-expr> yN=<num-expr> zN=<num-expr>
                xdeltaN=<num-expr> ydeltaN=<num-expr> zdeltaN=<num-expr>
                inN=<table> ifmtN=<in-format> istreamN=true|false
                icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

arrowN = small_arrow|medium_arrow|... (ErrorRenderer)

How arrows are represented.

The available options are:

- small_arrow
- medium_arrow
- large_arrow
- small_open_dart
- medium_open_dart
- large_open_dart
- small_filled_dart
- medium_filled_dart
- large_filled_dart
- lines
- capped_lines

[Default: small_arrow]

autoscaleN = true|false (Boolean)

Determines whether the default size of variable-sized markers like vectors and ellipses are automatically scaled to have a sensible size. If true, then the sizes of all the plotted markers are examined, and some dynamically calculated factor is applied to them all to make them a sensible size (by default, the largest ones will be a few tens of pixels). If false, the sizes will be the actual input values interpreted in data coordinates.

If auto-scaling is on, then markers will keep approximately the same screen size during zoom operations; if it's off, they will keep the same size in data coordinates.

Marker size is also affected by the `scale` parameter.

[Default: true]

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

scaleN = <factor> (*Double*)

Affects the size of variable-sized markers like vectors and ellipses. The default value is 1, smaller or larger values multiply the visible sizes accordingly.

[Default: 1]

shadingN = flat|translucent|transparent|density|aux|weighted <shade-paramsN> (*ShapeMode*)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- `flat` (Section 8.4.2)
- `translucent` (Section 8.4.3)
- `transparent` (Section 8.4.4)

- density (Section 8.4.5)
- aux (Section 8.4.6)
- weighted (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: flat]

xN = <num-expr> (*String*)
X coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

xdeltaN = <num-expr> (*String*)
Vector component in the X direction.

The value is a numeric algebraic expression based on column names as described in Section 10.

yN = <num-expr> (*String*)
Y coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

ydeltaN = <num-expr> (*String*)
Vector component in the Y direction.

The value is a numeric algebraic expression based on column names as described in Section 10.

zN = <num-expr> (*String*)
Z coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

zdeltaN = <num-expr> (*String*)
Vector component in the Z direction.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.24 xyzerror

Plots symmetric or asymmetric error bars in some or all of the plot dimensions. The shape of the error "bars" is quite configurable, including (for 2-d and 3-d errors) ellipses, rectangles etc aligned with the axes.

Usage Overview:

```
layerN=xyzerror errorbarN=none|lines|capped_lines|...
                shadingN=flat|translucent|transparent|density|aux|weighted <shade-paramsN>
                xN=<num-expr> yN=<num-expr> zN=<num-expr> xerrhiN=<num-expr>
                xerrloN=<num-expr> yerrhiN=<num-expr> yerrloN=<num-expr>
                zerrhiN=<num-expr> zerrloN=<num-expr> inN=<table>
                ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

errorbarN = none|lines|capped_lines|... (*ErrorRenderer*)

How errorbars are represented.

The available options are:

- none
- lines
- capped_lines
- caps
- arrows
- cuboid
- ellipse
- crosshair_ellipse
- rectangle
- crosshair_rectangle
- filled_ellipse
- filled_rectangle

[Default: lines]

icmdN = <cmds> (*ProcessingStep*[])

Specifies processing to be performed on the layer N input table as specified by parameter *inN*. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file *filename* to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of the input table as specified by parameter *inN*. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (auto) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the *ifmtN* parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("<syscmd" or "syscmd|"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the *inN* parameter will be read as a stream. It is necessary to give the *ifmtN* parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be

streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

shadingN = flat|translucent|transparent|density|aux|weighted <shade-paramsN>
(*ShapeMode*)

Determines how plotted objects in layer N are coloured. This may be influenced by how many objects are plotted over each other as well as the values of other parameters. Available options (Section 8.4) are:

- flat (Section 8.4.2)
- translucent (Section 8.4.3)
- transparent (Section 8.4.4)
- density (Section 8.4.5)
- aux (Section 8.4.6)
- weighted (Section 8.4.7)

Each of these options comes with its own set of parameters to specify the details of how colouring is done.

[Default: flat]

xN = <num-expr> (*String*)
X coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

xerrhiN = <num-expr> (*String*)

Error in the X coordinate in the positive direction. If no corresponding negative error value is supplied, then this value is also used in the negative direction, i.e. in that case errors are assumed to be symmetric.

The value is a numeric algebraic expression based on column names as described in Section 10.

xerrloN = <num-expr> (*String*)

Error in the X coordinate in the negative direction. If left blank, it is assumed to take the same value as the positive error.

The value is a numeric algebraic expression based on column names as described in Section 10.

yN = <num-expr> (*String*)
Y coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

yerrhiN = <num-expr> (*String*)

Error in the Y coordinate in the positive direction. If no corresponding negative error value is supplied, then this value is also used in the negative direction, i.e. in that case errors are assumed to be symmetric.

The value is a numeric algebraic expression based on column names as described in Section 10.

yerrloN = <num-expr> (*String*)

Error in the Y coordinate in the negative direction. If left blank, it is assumed to take the same value as the positive error.

The value is a numeric algebraic expression based on column names as described in Section 10.

zN = <num-expr> (*String*)
Z coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

zerrhiN = <num-expr> (*String*)

Error in the Z coordinate in the positive direction. If no corresponding negative error value is supplied, then this value is also used in the negative direction, i.e. in that case errors are assumed to be symmetric.

The value is a numeric algebraic expression based on column names as described in Section 10.

zerrloN = <num-expr> (*String*)

Error in the Z coordinate in the negative direction. If left blank, it is assumed to take the same value as the positive error.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.25 yerror

Shape

Plots symmetric or asymmetric error bars in the Y direction.

Shading

Paints markers in a single fixed colour.

Usage Overview:

```
layerN=yerror errorbarN=none|lines|capped_lines|caps|arrows
colorN=<rrggb>|red|blue|... tN=<time-expr> yN=<num-expr>
yerrhiN=<num-expr> yerrloN=<num-expr> inN=<table>
ifmtN=<in-format> istreamN=true|false icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

colorN = <rrggb>|red|blue|... (*Color*)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

errorbarN = none|lines|capped_lines|caps|arrows (*ErrorRenderer*)

How errorbars are represented.

The available options are:

- none
- lines
- capped_lines
- caps
- arrows

[Default: lines]

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the layer N input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (Boolean)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

tN = <time-expr> (String)

Time coordinate.

The value is a `Object` algebraic expression based on column names as described in Section 10.

yN = <num-expr> (String)

Vertical coordinate.

The value is a numeric algebraic expression based on column names as described in Section 10.

yerrhiN = <num-expr> (String)

Error in the Y coordinate in the positive direction. If no corresponding negative error value is supplied, then this value is also used in the negative direction, i.e. in that case errors are

assumed to be symmetric.

The value is a numeric algebraic expression based on column names as described in Section 10.

yerrloN = <num-expr> (*String*)

Error in the Y coordinate in the negative direction. If left blank, it is assumed to take the same value as the positive error.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.3.26 spectrogram

Plots spectrograms. A spectrogram is a sequence of spectra plotted as vertical 1-d images, each one plotted at a different horizontal coordinate.

This specialised layer is only available for `time` plots.

The way that data values are mapped to colours is usually controlled by options at the level of the plot itself, rather than by per-layer configuration.

Usage Overview:

```
layerN=spectrogram tN=<time-expr> spectrumN=<array-expr> twidthN=<num-expr>
                    inN=<table> ifmtN=<in-format> istreamN=true|false
                    icmdN=<cmds>
```

All the parameters listed here affect only the relevant layer, identified by the suffix `N`.

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the layer `N` input table as specified by parameter `inN`. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this

way.

- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = `true|false` (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

spectrumN = `<array-expr>` (*String*)

Provides an array of spectral samples at each data point. The value must be a numeric array (e.g. the value of an array-valued column).

The value is a `Object` algebraic expression based on column names as described in Section 10.

tN = `<time-expr>` (*String*)

Time coordinate.

The value is a `Object` algebraic expression based on column names as described in Section 10.

twidthN = `<num-expr>` (*String*)

Range on the Time axis over which the spectrum is plotted. If no value is supplied, an attempt will be made to determine it automatically by looking at the spacing of the Time coordinates plotted in the spectrogram.

The value is a numeric algebraic expression based on column names as described in Section 10.

8.4 Shading Modes

Some plot layer types have an associated `shading` parameter which determines how plotted markers are coloured. This is independent of the marker shapes (which may be points, vectors, ellipses, ...) but may be affected by how many markers are plotted on top of each other, additional input table values, selected colour maps etc. For the simplest shading types (e.g. `flat`) it's just a case of choosing a colour, but the more complex ones have several associated parameters.

The various shading types and their usages are described in the following subsections.

8.4.1 `auto`

Paints isolated points in their selected colour but where multiple points *in the same layer* overlap it adjusts the colour by darkening it. This means that for isolated points (most or all points in a non-crowded plot, or outliers in a crowded plot) it behaves just like `flat` mode, but it's easy to see where overdense regions lie.

This is like density mode, but with no user-configurable options.

Usage:

```
shadingN=auto colorN=<rrggb>|red|blue|...
```


All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

Associated parameters are as follows:

colorN = <rrggbb>|red|blue|... (*Color*)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

8.4.2 flat

Paints markers in a single fixed colour.

Usage:

```
shadingN=flat colorN=<rrggbb>|red|blue|...
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

Associated parameters are as follows:

colorN = <rrggbb>|red|blue|... (*Color*)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

8.4.3 translucent

Paints markers in a transparent version of their selected colour. The degree of transparency is determined by how many points are plotted on top of each other and by the transparency level. Unlike transparent mode, the transparency varies according to the average point density in the plot, so leaving the setting the same as you zoom in and out usually has a sensible effect.

Usage:

```
shadingN=translucent colorN=<rrggbb>|red|blue|... translevelN=<number>
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

Associated parameters are as follows:

colorN = <rrggbb>|red|blue|... (*Color*)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g.

`ff00ff` for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

translevelN = <number> (Double)

Sets the level of automatically controlled transparency. The higher this value the more transparent points are. Exactly how transparent points are depends on how many are currently being plotted on top of each other and the value of this parameter. The idea is that you can set it to some fixed value, and then get something which looks similarly transparent while you zoom in and out.

[Default: 0.1]

8.4.4 transparent

Paints markers in a transparent version of their selected colour. The degree of transparency is determined by how many points are plotted on top of each other and by the opaque limit. The opaque limit fixes how many points must be plotted on top of each other to completely obscure the background. This is set to a fixed value, so a transparent level that works well for a crowded region (or low magnification) may not work so well for a sparse region (or when zoomed in).

Usage:

```
shadingN=transparent colorN=<rrggbb>|red|blue|... opaqueN=<number>
```

All the parameters listed here affect only the relevant layer, identified by the suffix `N`.

Associated parameters are as follows:

colorN = <rrggbb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. `"ff00ff"` for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

opaqueN = <number> (Double)

The opacity of plotted points. The value is the number of points which have to be overplotted before the background is fully obscured.

[Default: 4]

8.4.5 density

Paints markers using a configurable colour map to indicate how many points are plotted over each other. Specifically, it colours each pixel according to how many times that pixel has been covered by one of the markers plotted by the layer in question. To put it another way, it generates a false-colour density map with pixel granularity using a smoothing kernel of the form of the markers plotted by the layer. The upshot is that you can see the plot density of points or other markers plotted.

This is like auto mode, but with more user-configurable options.

Usage:

```

shadingN=density colorN=<rrggb>|red|blue|...
                densemapN=blacker|whiter|inferno|... denseclipN=<lo>,<hi>
                denseflipN=true|false densequantN=<number>
                densefuncN=log|linear|sqrt|square densesubN=<lo>,<hi>

```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

Associated parameters are as follows:

colorN = <rrggb>|red|blue|... (Color)

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

denseclipN = <lo>,<hi> (Subrange)

Defines a subrange of the colour ramp to be used for Density shading. The is specified as a (low,high) comma-separated pair of two numbers between 0 and 1.

If the full range 0,1 (the default) is used, the whole range of colours specified by the selected shader will be used. But if, for instance a value of 0,0.5 is given, only those colours at the left hand end of the ramp will be seen.

[Default: 0,1]

denseflipN = true|false (Boolean)

If true, the colour map on the Density axis will be reversed.

[Default: false]

densefuncN = log|linear|sqrt|square (Scaling)

Defines the way that values in the Density range are mapped to the selected colour ramp.

The available options are:

- log: Logarithmic scaling
- linear: Linear scaling
- sqrt: Square root scaling
- square: Square scaling

[Default: log]

densemapN = blacker|whiter|inferno|... (Shader)

Color map used for Density axis shading.

A mixed bag of colour ramps are available: blacker, whiter, inferno, magma, plasma, viridis, cubehelix, sron, rainbow, rainbow2, rainbow3, pastel, accent, gnuplot, gnuplot2, specxby, set1, paired, hotcold, rdbu, piyg, brbg, cyan-magenta, red-blue, brg, heat, cold, light, greyscale, colour, standard, bugn, bupu, orrd, pubu, purd, huecl, hue, intensity, rgb_red, rgb_green, rgb_blue, hsv_h, hsv_s, hsv_v, yuv_y, yuv_u, yuv_v, scale_hsv_s, scale_hsv_v, scale_yuv_y. *Note:* many of these, including rainbow-like ones, are frowned upon by the visualisation community.

[Default: blacker]

densequantN = <number> (Double)

Allows the colour map used for the Density axis to be quantised. If an integer value *N* is chosen then the colour map will be viewed as *N* discrete evenly-spaced levels, so that only *N*

different colours will appear in the plot. This can be used to generate a contour-like effect, and may make it easier to trace the boundaries of regions of interest by eye.

If left blank, the colour map is nominally continuous (though in practice it may be quantised to a medium-sized number like 256).

densesubN = <lo>,<hi> (Subrange)

Defines a normalised adjustment to the data range of the Density axis. The value may be specified as a comma-separated pair of two numbers, giving the lower and upper bounds of the range of of interest respectively. This sub-range is applied to the data range that would otherwise be used, either automatically calculated or explicitly supplied; zero corresponds to the lower bound and one to the upper.

The default value "0,1" therefore has no effect. The range could be restricted to its lower half with the value 0,0.5.

[Default: 0,1]

8.4.6 aux

Paints markers in a colour determined by the value of an additional data coordinate. The marker colours then represent an additional dimension of the plot. You can also adjust the transparency of the colours used. The way that data values are mapped to colours is usually controlled by options at the level of the plot itself, rather than by per-layer configuration.

Usage:

```
shadingN=aux auxN=<num-expr> auxnullcolorN=<rrgbb>|red|blue|...
opaqueN=<number>
```

All the parameters listed here affect only the relevant layer, identified by the suffix N.

Associated parameters are as follows:

auxN = <num-expr> (String)

Colour coordinate for Aux shading.

This parameter gives a column name, fixed value, or algebraic expression for the aux coordinate for layer N. The value is a numeric algebraic expression based on column names as described in Section 10.

auxnullcolorN = <rrgbb>|red|blue|... (Color)

The color of points with a null value of the Aux coordinate.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

If the value is null, then points with a null Aux value will not be plotted at all.

[Default: grey]

opaqueN = <number> (Double)

The opacity of points plotted in the Aux colour. The value is the number of points which have to be overplotted before the background is fully obscured.

[Default: 1]

8.4.7 weighted

Paints markers like the Density mode, but with optional weighting by an additional coordinate. You can configure how the weighted coordinates are combined to give the final weighted result. The way that data values are mapped to colours is usually controlled by options at the level of the plot itself, rather than by per-layer configuration.

Usage:

```
shadingN=weighted weightN=<num-expr> colorN=<rrggb>|red|blue|...
                    combineN=sum|mean|median|min|max|variance|count|hit
```

All the parameters listed here affect only the relevant layer, identified by the suffix *N*.

Associated parameters are as follows:

colorN = <rrggb>|red|blue|... *(Color)*

The color of plotted data.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: red]

combineN = sum|mean|median|min|max|variance|count|hit *(Combiner)*

Defines how values contributing to the same pixel are combined together to produce the value assigned to that pixel (and hence its colour).

When a weight is in use, mean or sum are typically sensible choices. If there is no weight (a pure density map) then count is usually better, but in that case it may make more sense (it is more efficient) to use one of the other shading modes instead.

The available options are:

- sum: the sum of all the combined values
- mean: the mean of the combined values
- median: the median of the combined values (may be slow)
- min: the minimum of all the combined values
- max: the maximum of all the combined values
- variance: the sample variance of the combined values
- count: the number of non-blank values (weight is ignored)
- hit: 1 if any values present, NaN otherwise (weight is ignored)

[Default: mean]

weightN = <num-expr> *(String)*

Weight coordinate for weighted density shading.

This parameter gives a column name, fixed value, or algebraic expression for the weight coordinate for layer *N*. The value is a numeric algebraic expression based on column names as described in Section 10.

8.5 Output Modes

The plots generated by the plotting commands can be used in various different ways. One thing you might want to do is to write the output to a file in a given graphics format (*out*); another is to

preview it directly on the screen (*swing*). By default one or other of these will happen depending on whether you specify an output file. However there are other possibilities; these are listed in the following subsections.

Except for display to the screen, these modes should work happily on a headless machine (one with no graphics display, as may be the case for a web server). When running headless, you may find it necessary to set the java system property "java.awt.headless" to true - see Section 3.3.

The default output mode is *auto*, which means that output is to a file if an output file is specified, or to the screen if it is not. So in most cases you don't need to specify the *omode* parameter explicitly.

8.5.1 *swing*

Usage:

`omode=swing`

Plot will be displayed in a window on the screen. This plot is "live"; it can be resized and (except for old-style plots) navigated around with mouse actions in the same way as plots in TOPCAT.

8.5.2 *out*

Usage:

`omode=out out=<out-file> ofmt=png|png-transp|gif|jpeg|pdf|eps|eps-gzip`

Plot will be written to a file given by *out* using the graphics format given by *ofmt*.

8.5.3 *cgi*

Usage:

`omode=cgi ofmt=png|png-transp|gif|jpeg|pdf|eps|eps-gzip`

Plot will be written in a way suitable for CGI use direct from a web server. The output is in the graphics format given by *ofmt*, preceded by a suitable "Content-type" declaration.

8.5.4 *discard*

Usage:

`omode=discard`

Plot is drawn, but discarded. There is no output.

8.5.5 *auto*

Usage:

`omode=auto [out=<out-file>]`

Behaves as *swing* or *out* mode depending on presence of *out* parameter

8.6 Export Formats

Several of the plot output modes write the plot in some graphics format or other. When selecting an

output format it is important to understand the distinction between *bitmapped* and *vector* formats; basically bitmapped formats represent the image as a grid of finite-sized pixels while vector formats notionally draw smooth lines. Bitmapped formats are fine for a computer screen, but for high quality paper printouts you will want a vector format. You can convert from vector to bitmapped but not (usefully) in the other direction. There are a couple of subtleties to this distinction specific to STILTS graphical output as discussed below.

The following formats are the available values for the `ofmt` parameter of the various plot commands:

png

PNG format. This is a flexible bitmapped format providing transparency and an unlimited number of colours with good lossless compression. It is widely supported by non-ancient browsers and other image viewers, and is generally recommended for bitmapped output.

gif

GIF format. This is a bitmapped format providing transparency and lossless compression. The number of colours is limited to 255 however, so if you are using auxiliary axes (colour variation to represent higher dimensionality) or other plot features which use a wide range of colours you may see image degradation. It has long been widely supported by browsers and other image viewers.

jpeg

JPEG format. This is a bitmapped format with lossy compression intended primarily for photographs. Transparency is not supported, and although there is no limit on the maximum number of colours, its lossiness means that plots generated using it generally look a bit smudged.

pdf

Portable Document Format. This is the format which can be read by Adobe's Acrobat Reader. It is a widely portable vector format, and is suitable for printing at high resolution, either standalone or imported into some other presentation format. However, there are a couple of caveats when it comes to using it with STILTS plots.

1. If used to plot a very large number of points, the output PDF file can get quite large, though it's much better than for `eps` output (see below).
2. For certain colour shading options (auto, density, and in some circumstances transparency), the body of the plot will be drawn as a bitmap rather than vector graphics. This is sometimes a blessing in disguise since with very large numbers of points a vector PDF file could get unmanageably large in any case. In this case the interior of the plot will be pixellated. The axes and annotations outside of the plot will still be drawn in vector format however.

eps

Encapsulated Postscript. This is a vector format which is suitable for printing at high resolution either standalone or imported into some other presentation format (you may need to convert it via PDF depending on the intended destination). However, there are a couple of caveats when it comes to using it with STILTS plots.

1. Unfortunately the postscript driver used by STILTS is not very efficient and can result in large, sometimes very large, postscript output files. This is likely to be a problem for plots with a large number of non-transparent points. For this reason `eps-gzip` or `pdf` may be a better choice.
2. Postscript has no support for partial transparency, so if plots are drawn with partially transparent points (common for very large data sets) the only way they can be rendered is by drawing the body of the plot as a bitmap rather than as vector graphics. This is sometimes a blessing in disguise since with very large numbers of points a vector postscript file would likely be unmanageably large in any case. So if there is any

transparency in the plot, the interior of the plot will be pixellated. The axes and annotations outside of the plot will still be drawn in vector format however.

eps-gzip

Just like the `eps` format above except that the output is automatically compressed using the GZIP format as it is written. Postscript compresses well (typically a factor of 5-10).

9 Old-Style Plotting

This section describes deprecated commands. For recommended plotting commands, see Section 8.

From version 2.0 (October 2008), STILTS incorporated three table plotting commands:

- `plot2d`: Old-style 2D Scatter Plot
- `plot3d`: Old-style 3D Scatter Plot
- `plthist`: Old-style Histogram

These provided command-line access to some, though not all, of the plotting capabilities offered by TOPCAT.

Since version 3.0 (October 2014), these commands are deprecated in favour of the more powerful ones described in Section 8. The rest of this section describes some aspects of the deprecated commands for the benefit of legacy code. The output modes and formats are the same in old- and new-style plots, and are discussed in Section 8.5 and Section 8.6. The handling of parameters and suffixes for these commands is not quite the same as for new-style plots, and is documented in the next subsection.

As a simple example, if a file "cat.fits" contains the columns RMAG and BMAG for red and blue magnitudes, you can draw a two-dimensional colour-magnitude scatter plot with the command:

```
stilts plot2d in=cat.fits xdata=BMAG-RMAG ydata=BMAG
```

Since an output file is not specified, the plot is shown on the screen for convenience. To send the output to a PNG file, do instead:

```
stilts plot2d in=cat.fits xdata=BMAG-RMAG ydata=BMAG out=plot.png ofmt=png
```

in some cases (including the above), the `ofmt` parameter is not required since STILTS may be able to guess the format from the output file name. Various other options for output and graphics formats are described in Section 8.5 and Section 8.6

Some of the parameters use suffixes to define data sets and therefore behave a bit differently from the parameters elsewhere in STILTS - a discussion of these is given in the following subsection. Some other plotting-specific topics are also discussed below.

9.1 Parameter Suffixes

This section describes deprecated commands. For recommended plotting commands, see Section 8.

Some of the parameters for the plotting tasks behave a little bit differently to other parameters in STILTS, in order to accommodate related sets of values. If you look at the usage of one of the plotting commands, for instance in Appendix B.12.1, you will see that a number of the parameters have the suffixes "N" or "NS". These suffixes can be substituted with any convenient string to identify parameters which relate to the same input datasets or subsets. Specifically:

Suffix "N":

Denotes an input dataset. At least the `inN` parameter must be given to identify the source of the data; any other parameters with the same value of the N suffix relate to that dataset. A *dataset* here refers to a particular set of plot data from a table; in most cases each input table corresponds to a different dataset, though two datasets may correspond to different sets of columns from the same table.

Suffix "NS":

Denotes a particular subset of the rows in dataset N. At least the `subsetNS` parameter must be given to identify the expression by which the subset is defined; any other parameters with the

same value of the `NS` suffix relate to that subset.

Some examples will help to illustrate. The following will generate a Cartesian plot of catalogue position from a single dataset:

```
stilts plot2d in=gals.fits xdata=RA ydata=DEC
```

In this case the `N` suffix is present on each of the parameters `in`, `xdata` and `ydata`, but is equal to the empty string, hence invisible. This is perfectly legal, and convenient when only a single table is in use. If we wish to overplot two datasets however, the dataset suffixes (or one of them at least) have to be made explicit so that different ones can be used, for instance:

```
stilts plot2d in1=gals.fits xdata1=RA ydata1=DEC
               in2=stars.fits xdata2=RAJ2000 ydata2=DEJ2000
```

The suffix values "1" and "2" are quite arbitrary and can be chosen as convenient, so the following would do exactly the same as the previous example:

```
stilts plot2d in_GAL=gals.fits xdata_GAL=RA ydata_GAL=DEC
               in_STAR=stars.fits xdata_STAR=RAJ2000 ydata_STAR=DEJ2000
```

The other parameters which have the `N` suffix apply only to the matching dataset, so for instance the following:

```
stilts plot2d in1=gals.fits xdata1=RA ydata1=DEC txtlabell1=NGC_ID
               in2=stars.fits xdata2=RAJ2000 ydata2=DEJ2000
```

would draw text labels adjacent to the points from only the `gals.fits` file giving the contents of its `NGC_ID` column.

The `NS` suffix identifies distinct *row subsets* within the same or different datasets. A subset is defined by supplying a boolean inclusion expression (each row is included only if the expression evaluates true for that row) as the value of a `subsetNS` parameter. If, as in all the examples we have seen so far, no `subsetNS` parameter is supplied for a given dataset, then it is treated as a special case, as if a single subset with a name equal to the empty string (`s=""`) containing all rows has been specified. So our earlier simple example:

```
stilts plot2d in=gals.fits xdata=RA ydata=DEC
```

is equivalent to

```
stilts plot2d in=gals.fits xdata=RA ydata=DEC subset=true
```

If we wish to split the plotted points into two sets based on their R-B colours, we can write something like:

```
stilts plot2d in=gals.fits xdata=RA ydata=DEC
               subsetX='RMAG-BMAG>0' subsetY='RMAG-BMAG<=0'
```

This will generate a plot with two subsets shown using different colours and/or plotting symbols. These colours and symbols are selected automatically. More control over the appearance can be exercised by setting values for some of the other parameters with `NS` suffixes, for instance

```
stilts plot2d in=gals.fits xdata=RA ydata=DEC
               subset_A='RMAG-BMAG>0' colour_A=blue
               subset_B='RMAG-BMAG<=0' colour_B=red
```

Again, the suffix strings can be chosen to have any value as convenient.

The dataset- and subset-specific parameters must be put together if there are multiple datasets with multiple subsets to plot simultaneously, for instance:

```
stilts plot2d in_1=gals.fits xdata_1=RA ydata_1=DEC
                        subset_1_A='RMAG-BMAG>0' colour_1_A=blue
                        subset_1_B='RMAG-BMAG<=0' colour_1_B=red
in_2=stars.fits xdata_2=RAJ2000 ydata_2=DEJ2000
                colour_2=green
```

Finally, it's not quite true that the suffixes chosen have no effect on the plot; they may influence the order in which sets are plotted. Markers drawn for sets plotted earlier may be obscured by the markers drawn for sets plotted later, so this can affect the appearance of the plot. If you want to control this, use the `sequence` parameter. For instance, to ensure that star data appears on top of galaxy data in the plot, do the following:

```
stilts plot2d in_GAL=gals.fits xdata_GAL=RA ydata_GAL=DEC
              in_STAR=stars.fits xdata_STAR=RAJ2000 ydata_STAR=DEJ2000
              sequence=_GAL,_STAR
```

More examples can be found in the **Examples** subsections of the individual plotting command descriptions in Appendix B.

10 Algebraic Expression Syntax

Many of the STILTS commands allow you to use algebraic expressions based on table columns when doing things like making row selections, defining new columns, selecting values to plot or match, and so on. In these cases you are defining an expression which has a value in each row as a function of the values in the existing columns in that row. This is a powerful feature which permits you to manipulate and select table data in very flexible ways. The syntax for entering these expressions is explained in this section.

What you write are actually expressions in the Java language, which are compiled into Java bytecode before evaluation. However, this does not mean that you need to be a Java programmer to write them. The syntax is pretty similar to C, but even if you've never programmed in C most simple things, and many complicated ones, are quite intuitive.

The following explanation gives some guidance and examples for writing these expressions. Unfortunately a complete tutorial on writing Java is beyond the scope of this document, but it should provide enough information for even a novice to write useful expressions.

The expressions that you can write are basically any function of all the column values which apply to a given row; the function result can then be used where STILTS needs a per-row value, for instance to define a new column. If the built-in operators and functions are not sufficient, or it's unwieldy to express your function in one line of code, it is possible to add new functions by writing your own classes - see Section 10.7.3.

Note that since these algebraic expressions often contain spaces, you may need to enclose them in single or double quotes so that they don't get confused with other parts of the command string.

Note: if Java is running in an environment with certain security restrictions (a security manager which does not permit creation of custom class loaders) then algebraic expressions won't work at all. It's not particularly likely that security restrictions will be in place if you are running from the command line though.

10.1 Referencing Column Values

To create a useful expression which can be evaluated for each row in a table, you will have to refer to cells in different columns of that row. You can do this in three ways:

By Name

The Name of the column may be used if it is unique (no other column in the table has the same name) and if it has a suitable form. This means that it must have the form of a Java variable - basically starting with a letter and continuing with letters, numbers, underscores and currency symbols. In particular it cannot contain spaces, commas, parentheses etc.

As a special case, if an expression contains just a single column name, rather than some more complicated expression, then any column name may be used, even one containing non-alphanumeric characters.

Column names are treated case-insensitively.

By \$ID

The "\$ID" identifier of the column may always be used to refer to it; this is a useful fallback if the column name isn't suitable for some reason (for instance it contains spaces or is not unique). This is just a "\$" sign followed by the column index - the first column is \$1.

By ucd\$ specifier

If the column has a Unified Content Descriptor (this will usually only be the case for VOTable or possibly FITS format tables) you can refer to it using an identifier of the form

`ucd$<ucd-spec>`". Depending on the version of UCD scheme used, UCDs can contain various punctuation marks such as underscores, semicolons and dots; for the purpose of this syntax these should all be represented as underscores ("_"). So to identify a column which has the UCD "phot.mag;em.opt.R", you should use the identifier "ucd\$phot_mag_em_opt_r". Matching is not case-sensitive. Furthermore, a trailing underscore acts as a wildcard, so that the above column could also be referenced using the identifier "ucd\$phot_mag_". If multiple columns have UCDs which match the given identifier, the first one will be used.

Note that the same syntax can be used for referencing table parameters (see the next section); columns take preference so if a column and a parameter both match the requested UCD, the column value will be used.

By `utype$` specifier

If the column has a **Utype** (this will usually only be the case for VOTable or possibly FITS format tables) you can refer to it using an identifier of the form "utype\$<utype-spec>". Utypes can contain various punctuation marks such as colons and dots; for the purpose of this syntax these should all be represented as underscores ("_"). So to identify a column which has the Utype "ssa:Access.Format", you should use the identifier "utype\$ssa_Access_Format". Matching is not case-sensitive. If multiple columns have Utypes which match the given identifier, the first one will be used.

Note that the same syntax can be used for referencing table parameters (see the next section); columns take preference so if a column and a parameter both match the requested Utype, the column value will be used.

With the `Object$` prefix

If a column is referenced with the prefix "Object\$" before its identifier (e.g. "Object\$BMAG" for a column named BMAG) the result will be the column value as a java Object. Without that prefix, numeric columns are evaluated as java primitives. In most cases, you *don't want to do this*, since it means that you can't use the value in arithmetic expressions. However, if you need the value to be passed to a (possibly user-defined) method, and you need that method to be invoked even when the value is null, you have to do it like this. Null-valued primitives otherwise cause expression evaluation to abort.

There is also a special column:

`$index`

The value of this is the current row number (the first row is 1). You can alternatively use the form `$0`. (The form `index` is also permitted, but deprecated). Note that this value is a `long` (8-byte integer); when using it in certain expressions you may find it necessary to convert it to an `int` (4-byte integer) using the `toInteger()` function.

The value of the variables so referenced will be a primitive (boolean, byte, short, char, int, long, float, double) if the column contains one of the corresponding types. Otherwise it will be an Object of the type held by the column, for instance a String. In practice this means: you can write the name of a column, and it will evaluate to the numeric (or string) value that that column contains in each row. You can then use this in normal algebraic expressions such as "B_MAG-U_MAG" as you'd expect.

10.2 Referencing Parameter Values

Some tables have constant values associated with them; these may represent such things as the epoch at which observations were taken, the name of the catalogue, an angular resolution associated with all observations, or any number of other things. Such constants are known as *table parameters* (not to be confused with parameters passed to STILTS commands) and can be thought of as extra columns which have the same value for every row. The values of such parameters can be referenced in STILTS algebraic expressions as follows:

param\$name

If the parameter name has a suitable form (starting with a letter and continuing with letters or numbers) it can be referenced by prefixing that name with the string `param$`.

ucd\$ucd-spec

If the parameter has a Unified Content Descriptor it can be referenced by prefixing the UCD specifier with the string `ucd$`. Any punctuation marks in the UCD should be replaced by underscores, and a trailing underscore is interpreted as a wildcard. See Section 10.1 for more discussion.

utype\$utype-spec

If the parameter has a Utype, it can be referenced by prefixing the Utype specifier with the string `utype$`. Any punctuation marks in the Utype should be replaced by underscores. See Section 10.1 for more discussion.

Note that if a parameter has a name in an unsuitable form (e.g. containing spaces) and has no UCD then it cannot be referenced in an expression.

There are also a couple of special values:

\$ncol

The number of columns in the table.

\$nrow

The number of rows in the table. Note in some cases this is not known (e.g. if the table is being streamed), in which case the value of this variable is null. Note also that this value is a `long` (8-byte integer); when using it in certain expressions you may find it necessary to convert it to an `int` (4-byte integer) using the `toInteger()` function.

10.3 Null Values

When no special steps are taken, if a null value (blank cell) is encountered in evaluating an expression (usually because one of the columns it relies on has a null value in the row in question) then the result of the expression is also null.

It is possible to exercise more control than this, but it requires a little bit of care, because the expressions work in terms of primitive values (numeric or boolean ones) which don't in general have a defined null value. The name `"null"` in expressions gives you the java `null` reference, but this cannot be matched against a primitive value or used as the return value of a primitive expression.

For most purposes, the following two tips should enable you to work with null values:

Testing for null

To test whether a column contains a null value, prepend the string `"NULL_"` (use upper case) to the column name or `$ID`. This will yield a boolean value which is true if the column contains a blank, and false otherwise.

Returning null

To return a null value from a numeric expression, use the name `"NULL"` (upper case). To return a null value from a non-numeric expression (e.g. a String column) use the name `"null"` (lower case).

Null values are often used in conjunction with the conditional operator, `"? :"`; the expression

```
test ? tval : fval
```

returns the value `tval` if the boolean expression `test` evaluates true, or `fval` if `test` evaluates false. So for instance the following expression:

```
Vmag == -99 ? NULL : Vmag
```

can be used to define a new column which has the same value as the `Vmag` column for most values, but if `Vmag` has the "magic" value -99 the new column will contain a blank. The opposite trick (substituting a blank value with a magic one) can be done like this:

```
NULL_Vmag ? -99 : Vmag
```

Some more examples are given in Section 10.6.

10.4 Operators

The operators are pretty much the same as in the C language. The common ones are:

Arithmetic

- +** (add)
- (subtract)
- *** (multiply)
- /** (divide)
- %** (modulus)

Boolean

- !** (not)
- &&** (and)
- ||** (or)
- ^** (exclusive-or)
- ==** (numeric identity)
- !=** (numeric non-identity)
- <** (less than)
- >** (greater than)
- <=** (less than or equal)
- >=** (greater than or equal)

Bitwise

- &** (and)
- |** (or)
- ^** (exclusive-or)
- <<** (left shift)
- >>** (right shift)
- >>>** (logical right shift)

Numeric Typecasts

- (byte)** (numeric -> signed byte)
- (short)** (numeric -> 2-byte integer)
- (int)** (numeric -> 4-byte integer)
- (long)** (numeric -> 8-byte integer)
- (float)** (numeric -> 4-type floating point)
- (double)** (numeric -> 8-byte floating point)

Note you may find the Maths (Section 10.5.8) conversion functions more convenient for numeric conversions than these.

Other

+ (string concatenation)
[] (array dereferencing - first element is zero)
?: (conditional switch)
instanceof (class membership)

10.5 Functions

Many functions are available for use within your expressions, covering standard mathematical and trigonometric functions, arithmetic utility functions, type conversions, and some more specialised astronomical ones. You can use them in just the way you'd expect, by using the function name (unlike column names, this is case-sensitive) followed by comma-separated arguments in brackets, so

```
max( IMAG, J MAG )
```

will give you the larger of the values in the columns IMAG and J MAG, and so on.

The functions available for use by default are listed by class in the following subsections with their arguments and short descriptions. The `funcs` command provides another way to browse these function descriptions online.

10.5.1 Tilings

Pixel tiling functions for the celestial sphere.

```
htmIndex( level, ra, dec )
```

Gives the HTM (Hierarchical Triangular Mesh) pixel index for a given sky position.

- `level` (*integer*): HTM level
- `ra` (*floating point*): right ascension in degrees
- `dec` (*floating point*): declination in degrees
- return value (*long integer*): pixel index

```
healpixNestIndex( k, ra, dec )
```

Gives the pixel index for a given sky position in the HEALPix NEST scheme.

- `k` (*integer*): resolution parameter - log to base 2 of nside
- `ra` (*floating point*): right ascension in degrees
- `dec` (*floating point*): declination in degrees
- return value (*long integer*): pixel index

```
healpixRingIndex( k, ra, dec )
```

Gives the pixel index for a given sky position in the HEALPix RING scheme.

- `k` (*integer*): resolution parameter - log to base 2 of nside
- `ra` (*floating point*): right ascension in degrees
- `dec` (*floating point*): declination in degrees
- return value (*long integer*): pixel index

```
healpixK( pixelsize )
```

Gives the HEALPix resolution parameter suitable for a given pixel size. This `k` value is the logarithm to base 2 of the Nside parameter.

- `pixelsize` (*floating point*): pixel size in degrees
- return value (*integer*): HEALPix resolution parameter `k`

healpixResolution(k)

Gives the approximate resolution in degrees for a given HEALPix resolution parameter *k*. This *k* value is the logarithm to base 2 of the Nside parameter.

- *k* (*integer*): HEALPix resolution parameter *k*
- return value (*floating point*): approximate angular resolution in degrees

healpixSteradians(k)

Returns the solid angle in steradians of each HEALPix pixel at a given order.

- *k* (*integer*): HEALPix resolution parameter *k*
- return value (*floating point*): pixel size in steradians

healpixSqdeg(k)

Returns the solid angle in square degrees of each HEALPix pixel at a given order.

- *k* (*integer*): HEALPix resolution parameter *k*
- return value (*floating point*): pixel size in steradians

steradiansToSqdeg(sr)

Converts a solid angle from steradians to square degrees.

The unit sphere is 4π steradians = $360 \times 360 / \pi$ square degrees.

- *sr* (*floating point*): quantity in steradians
- return value (*floating point*): quantity in square degrees

sqdegToSteradians(sqdeg)

Converts a solid angle from square degrees to steradians.

The unit sphere is 4π steradians = $360 \times 360 / \pi$ square degrees.

- *sqdeg* (*floating point*): quantity in square degrees
- return value (*floating point*): quantity in steradians

htmLevel(pixelsize)

Gives the HTM *level* parameter suitable for a given pixel size.

- *pixelsize* (*floating point*): required resolution in degrees
- return value (*integer*): HTM level parameter

htmResolution(level)

Gives the approximate resolution in degrees for a given HTM depth level.

- *level* (*integer*): HTM depth
- return value (*floating point*): approximate angular resolution in degrees

SQDEG

Solid angle in steradians corresponding to 1 square degree.

10.5.2 Arithmetic

Standard arithmetic functions including things like rounding, sign manipulation, and maximum/minimum functions.

roundUp(x)

Rounds a value up to an integer value. Formally, returns the smallest (closest to negative infinity) integer value that is not less than the argument.

- *x (floating point)*: a value.
- return value (*integer*): *x* rounded up

roundDown(*x*)

Rounds a value down to an integer value. Formally, returns the largest (closest to positive infinity) integer value that is not greater than the argument.

- *x (floating point)*: a value
- return value (*integer*): *x* rounded down

round(*x*)

Rounds a value to the nearest integer. Formally, returns the integer that is closest in value to the argument. If two integers are equally close, the result is the even one.

- *x (floating point)*: a floating point value.
- return value (*integer*): *x* rounded to the nearest integer

roundDecimal(*x*, *dp*)

Rounds a value to a given number of decimal places. The result is a `float` (32-bit floating point value), so this is only suitable for relatively low-precision values. It's intended for truncating the number of apparent significant figures represented by a value which you know has been obtained by combining other values of limited precision. For more control, see the functions in the `Formats` class.

- *x (floating point)*: a floating point value
- *dp (integer)*: number of decimal places (digits after the decimal point) to retain
- return value (*floating point*): floating point value close to *x* but with a limited apparent precision

abs(*x*)

Returns the absolute value of an integer value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

- *x (integer)*: the argument whose absolute value is to be determined
- return value (*integer*): the absolute value of the argument.

abs(*x*)

Returns the absolute value of a floating point value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

- *x (floating point)*: the argument whose absolute value is to be determined
- return value (*floating point*): the absolute value of the argument.

max(*a*, *b*)

Returns the greater of two integer values. If the arguments have the same value, the result is that same value.

Multiple-argument maximum functions are also provided in the `Arrays` and `Lists` packages.

- *a (integer)*: an argument.
- *b (integer)*: another argument.
- return value (*integer*): the larger of *a* and *b*.

maxNaN(*a*, *b*)

Returns the greater of two floating point values. If the arguments have the same value, the result is that same value. If either value is blank, then the result is blank.

- *a (floating point)*: an argument.
- *b (floating point)*: another argument.
- return value (*floating point*): the larger of *a* and *b*.

maxReal(a, b)

Returns the greater of two floating point values, ignoring blanks. If the arguments have the same value, the result is that same value. If one argument is blank, the result is the other one. If both arguments are blank, the result is blank.

Multiple-argument maximum functions are also provided in the `Arrays` and `Lists` packages.

- *a (floating point)*: an argument
- *b (floating point)*: another argument
- return value (*floating point*): the larger non-blank value of *a* and *b*

min(a, b)

Returns the smaller of two integer values. If the arguments have the same value, the result is that same value.

Multiple-argument minimum functions are also provided in the `Arrays` and `Lists` packages.

- *a (integer)*: an argument.
- *b (integer)*: another argument.
- return value (*integer*): the smaller of *a* and *b*.

minNaN(a, b)

Returns the smaller of two floating point values. If the arguments have the same value, the result is that same value. If either value is blank, then the result is blank.

- *a (floating point)*: an argument.
- *b (floating point)*: another argument.
- return value (*floating point*): the smaller of *a* and *b*.

minReal(a, b)

Returns the smaller of two floating point values, ignoring blanks. If the arguments have the same value, the result is that same value. If one argument is blank, the result is the other one. If both arguments are blank, the result is blank.

Multiple-argument minimum functions are also provided in the `Arrays` and `Lists` packages.

- *a (floating point)*: an argument
- *b (floating point)*: another argument
- return value (*floating point*): the larger non-blank value of *a* and *b*

10.5.3 Conversions

Functions for converting between strings and numeric values.

toString(fpVal)

Turns a numeric value into a string.

- *fpVal (floating point)*: floating point numeric value
- return value (*String*): a string representation of *fpVal*

toString(intVal)

Turns an integer numeric value into a string.

- *intVal (long integer)*: integer numeric value
- return value (*String*): a string representation of *intVal*

toString(charVal)

Turns a single character value into a string.

- `charVal (char)`: character numeric value
- return value (*String*): a string representation of `charVal`

toString(byteVal)

Turns a byte value into a string.

- `byteVal (byte)`: byte numeric value
- return value (*String*): a string representation of `byteVal`

toString(booleanVal)

Turns a boolean value into a string.

- `booleanVal (boolean)`: boolean value (true or false)
- return value (*String*): a string representation of `booleanVal` ("true" or "false")

toString(objVal)

Turns any object value into a string. As applied to existing string values this isn't really useful, but it means that you can apply `toString` to any object value without knowing its type and get a useful return from it.

- `objVal (Object)`: non-primitive value
- return value (*String*): a string representation of `objVal`

parseByte(str)

Attempts to interpret a string as a byte (8-bit signed integer) value. If the input string can't be interpreted in this way, a blank value will result.

- `str (String)`: string containing numeric representation
- return value (*byte*): byte value of `str`

parseShort(str)

Attempts to interpret a string as a short (16-bit signed integer) value. If the input string can't be interpreted in this way, a blank value will result.

- `str (String)`: string containing numeric representation
- return value (*short integer*): byte value of `str`

parseInt(str)

Attempts to interpret a string as an int (32-bit signed integer) value. If the input string can't be interpreted in this way, a blank value will result.

- `str (String)`: string containing numeric representation
- return value (*integer*): byte value of `str`

parseLong(str)

Attempts to interpret a string as a long (64-bit signed integer) value. If the input string can't be interpreted in this way, a blank value will result.

- `str (String)`: string containing numeric representation
- return value (*long integer*): byte value of `str`

parseFloat(str)

Attempts to interpret a string as a float (32-bit floating point) value. If the input string can't be interpreted in this way, a blank value will result.

- `str (String)`: string containing numeric representation
- return value (*floating point*): byte value of `str`

parseDouble(str)

Attempts to interpret a string as a double (64-bit signed integer) value. If the input string can't

be interpreted in this way, a blank value will result.

- `str` (*String*): string containing numeric representation
- return value (*floating point*): byte value of `str`

toByte(value)

Attempts to convert the numeric argument to a byte (8-bit signed integer) result. If it is out of range, a blank value will result.

- `value` (*floating point*): numeric value for conversion
- return value (*byte*): `value` converted to type byte

toShort(value)

Attempts to convert the numeric argument to a short (16-bit signed integer) result. If it is out of range, a blank value will result.

- `value` (*floating point*): numeric value for conversion
- return value (*short integer*): `value` converted to type short

toInteger(value)

Attempts to convert the numeric argument to an int (32-bit signed integer) result. If it is out of range, a blank value will result.

- `value` (*floating point*): numeric value for conversion
- return value (*integer*): `value` converted to type int

toLong(value)

Attempts to convert the numeric argument to a long (64-bit signed integer) result. If it is out of range, a blank value will result.

- `value` (*floating point*): numeric value for conversion
- return value (*long integer*): `value` converted to type long

toFloat(value)

Attempts to convert the numeric argument to a float (32-bit floating point) result. If it is out of range, a blank value will result.

- `value` (*floating point*): numeric value for conversion
- return value (*floating point*): `value` converted to type float

toDouble(value)

Converts the numeric argument to a double (64-bit signed integer) result.

- `value` (*floating point*): numeric value for conversion
- return value (*floating point*): `value` converted to type double

toHex(value)

Converts the integer argument to hexadecimal form.

- `value` (*long integer*): integer value
- return value (*String*): hexadecimal representation of `value`

fromHex(hexVal)

Converts a string representing a hexadecimal number to its integer value.

- `hexVal` (*String*): hexadecimal representation of value
- return value (*integer*): integer value represented by `hexVal`

10.5.4 Distances

Functions for converting between different measures of cosmological distance.

The following parameters are used:

- **z**: redshift
- **H0**: Hubble constant in km/sec/Mpc (example value ~70)
- **omegaM**: Density ratio of the universe (example value 0.3)
- **omegaLambda**: Normalised cosmological constant (example value 0.7)

For a flat universe, $\text{omegaM} + \text{omegaLambda} = 1$

The terms and formulae used here are taken from the paper by D.W.Hogg, *Distance measures in cosmology*, astro-ph/9905116 (<http://arxiv.org/abs/astro-ph/9905116>) v4 (2000).

MpcToM(distMpc)

Converts from MegaParsecs to metres.

- **distMpc** (*floating point*): distance in Mpc
- **return value** (*floating point*): distance in m

mToMpc(distM)

Converts from metres to MegaParsecs.

- **distM** (*floating point*): distance in m
- **return value** (*floating point*): distance in Mpc

zToDist(z)

Quick and dirty function for converting from redshift to distance.

Warning: this makes some reasonable assumptions about the cosmology and returns the luminosity distance. It is only intended for approximate use. If you care about the details, use one of the more specific functions here.

- **z** (*floating point*): redshift
- **return value** (*floating point*): some distance measure in Mpc

zToAge(z)

Quick and dirty function for converting from redshift to time.

Warning: this makes some reasonable assumptions about the cosmology. It is only intended for approximate use. If you care about the details use one of the more specific functions here.

- **z** (*floating point*): redshift
- **return value** (*floating point*): 'age' of photons from redshift z in Gyr

comovingDistanceL(z, H0, omegaM, omegaLambda)

Line-of-sight comoving distance.

- **z** (*floating point*): redshift
- **H0** (*floating point*): Hubble constant in km/sec/Mpc
- **omegaM** (*floating point*): density ratio of the universe
- **omegaLambda** (*floating point*): normalised cosmological constant
- **return value** (*floating point*): line-of-sight comoving distance in Mpc

comovingDistanceT(z, H0, omegaM, omegaLambda)

Transverse comoving distance.

- **z** (*floating point*): redshift
- **H0** (*floating point*): Hubble constant in km/sec/Mpc

- `omegaM` (*floating point*): density ratio of the universe
- `omegaLambda` (*floating point*): normalised cosmological constant
- return value (*floating point*): transverse comoving distance in Mpc

angularDiameterDistance(z, H0, omegaM, omegaLambda)

Angular diameter distance.

- `z` (*floating point*): redshift
- `H0` (*floating point*): Hubble constant in km/sec/Mpc
- `omegaM` (*floating point*): density ratio of the universe
- `omegaLambda` (*floating point*): normalised cosmological constant
- return value (*floating point*): angular diameter distance in Mpc

luminosityDistance(z, H0, omegaM, omegaLambda)

Luminosity distance.

- `z` (*floating point*): redshift
- `H0` (*floating point*): Hubble constant in km/sec/Mpc
- `omegaM` (*floating point*): density ratio of the universe
- `omegaLambda` (*floating point*): normalised cosmological constant
- return value (*floating point*): luminosity distance in Mpc

lookbackTime(z, H0, omegaM, omegaLambda)

Lookback time. This returns the difference between the age of the universe at time of observation (now) and the age of the universe at the time when photons of redshift `z` were emitted.

- `z` (*floating point*): redshift
- `H0` (*floating point*): Hubble constant in km/sec/Mpc
- `omegaM` (*floating point*): density ratio of the universe
- `omegaLambda` (*floating point*): normalised cosmological constant
- return value (*floating point*): lookback time in Gyr

comovingVolume(z, H0, omegaM, omegaLambda)

Comoving volume. This returns the all-sky total comoving volume out to a given redshift `z`.

- `z` (*floating point*): redshift
- `H0` (*floating point*): Hubble constant in km/sec/Mpc
- `omegaM` (*floating point*): density ratio of the universe
- `omegaLambda` (*floating point*): normalised cosmological constant
- return value (*floating point*): comoving volume in Gpc³

SPEED_OF_LIGHT

Speed of light in m/s.

METRE_PER_PARSEC

Number of metres in a parsec.

SEC_PER_YEAR

Number of seconds in a year.

10.5.5 KCorrections

Functions for calculating K-corrections.

kCorr(filter, redshift, colorType, colorValue)

Calculates K-corrections. This allows you to determine K-corrections for a galaxy, given its redshift and a colour. Filters for GALEX, SDSS, UKIDSS, Johnson, Cousins and 2MASS are covered.

To define the calculation you must choose both a filter, specified as a `KCF_*` constant, and a colour (filter pair) specified as a `KCC_*` constant. For each available filter, only certain colours are available, as described in the documentation of the relevant `KCF_*` constant.

The algorithm used is described at <http://kcor.sai.msu.ru/> (<http://kcor.sai.msu.ru/>). This is based on the paper "*Analytical Approximations of K-corrections in Optical and Near-Infrared Bands*" by I.Chilingarian, A.-L.Melchior and I.Zolotukhin (2010MNRAS.405.1409C (<http://adsabs.harvard.edu/abs/2010MNRAS.405.1409C>)), but extended to include GALEX UV bands and with redshift coverage up to 0.5 as described in "*Universal UV-optical Colour-Colour-Magnitude Relation of Galaxies*" by I.Chilingarian and I.Zolotukhin (2012MNRAS.419.1727C (<http://adsabs.harvard.edu/abs/2012MNRAS.419.1727C>))).

- `filter` (*KCorrections.KFilter*): `KCF_*` constant defining the filter for which you want to calculate the K-correction
- `redshift` (*floating point*): galaxy redshift; this should be in the range 0-0.5
- `colorType` (*KCorrections.KColor*): `KCC_*` constant defining the filter pair for the calculation; check the `KCF_*` constant documentation to see which ones are permitted for a given filter
- `colorValue` (*floating point*): the value of the colour
- `return value` (*floating point*): K correction

KCF_FUV

GALEX FUV filter (AB). Use with `KCC_FUVNUV` or `KCC_FUVu`.

KCF_NUV

GALEX NUV filter (AB). Use with `KCC_NUVg` or `KCC_NUVr`.

KCF_u

SDSS u filter (AB). Use with `KCC_ur`, `KCC_ui` or `KCC_uz`.

KCF_g

SDSS g filter (AB). Use with `KCC_gr`, `KCC_gi` or `KCC_gz`.

KCF_r

SDSS r filter (AB). Use with `KCC_gr` or `KCC_ur`.

KCF_i

SDSS i filter (AB). Use with `KCC_gi` or `KCC_ui`.

KCF_z

SDSS z filter (AB). Use with `KCC_rz`, `KCC_gz` or `KCC_uz`.

KCF_Y

UKIDSS Y filter (AB). Use with `KCC_YH` or `KCC_YK`.

KCF_J

UKIDSS J filter (AB). Use with `KCC_JK` or `KCC_JH`.

KCF_H

UKIDSS H filter (AB). Use with KCC_HK or KCC_JH.

KCF_K

UKIDSS K filter (AB). Use with KCC_JK or KCC_HK.

KCF_U

Johnson U filter (Vega). Use with KCC_URc.

KCF_B

Johnson B filter (Vega). Use with KCC_BRc or KCC_BIc.

KCF_V

Johnson V filter (Vega). Use with KCC_VIc or KCC_VRc.

KCF_Rc

Cousins Rc filter (Vega). Use with KCC_BRc or KCC_VRc.

KCF_Ic

Cousins Ic filter (Vega). Use with KCC_VIc.

KCF_J2

2MASS J filter (Vega). Use with KCC_J2Ks2 or KCC_J2H2.

KCF_H2

2MASS H filter (Vega). Use with KCC_H2Ks2 or KCC_J2H2.

KCF_Ks2

2MASS Ks filter (Vega). Use with KCC_J2Ks2 or KCC_H2Ks2.

KCC_BIc

Johnson B - Cousins Ic colour.

KCC_BRc

Johnson B - Cousins Rc colour.

KCC_FUVNUV

GALEX FUV - NUV colour.

KCC_FUVu

GALEX FUV - SDSS u colour.

KCC_gi

SDSS g - i colour.

KCC_gr

SDSS g - r colour.

KCC_gz

SDSS g - z colour.

KCC_H2Ks2

2MASS H - Ks colour.

KCC_HK

UKIDSS H - K colour.

KCC_J2H2

2MASS J - H colour.

KCC_J2Ks2

2MASS J - Ks colour.

KCC_JH

UKIDSS J - H colour.

KCC_JK

UKIDSS J - K colour.

KCC_NUVg

GALEX NUV - SDSS g colour.

KCC_NUVr

GALEX NUV - SDSS r colour.

KCC_rz

SDSS r - SDSS z colour.

KCC_ui

SDSS u - SDSS i colour.

KCC_URc

Johnson U - Cousins Rc colour.

KCC_ur

SDSS u - r colour.

KCC_uz

SDSS u - z colour.

KCC_VIc

Johnson V - Cousins Ic colour.

KCC_VRc

Johnson V - Cousins Rc colour.

KCC_YH

UKIDSS Y - H colour.

KCC_YK

UKIDSS Y - K colour.

10.5.6 Times

Functions for conversion of time values between various forms. The forms used are

Modified Julian Date (MJD)

A continuous measure in days since midnight at the start of 17 November 1858. Based on UTC.

ISO 8601

A string representation of the form `yyyy-mm-ddThh:mm:ss.s`, where the `T` is a literal character (a space character may be used instead). Based on UTC.

Julian Epoch

A continuous measure based on a Julian year of exactly 365.25 days. For approximate purposes this resembles the fractional number of years AD represented by the date. Sometimes (but not here) represented by prefixing a 'J'; J2000.0 is defined as 2000 January 1.5 in the TT timescale.

Besselian Epoch

A continuous measure based on a tropical year of about 365.2422 days. For approximate purposes this resembles the fractional number of years AD represented by the date. Sometimes (but not here) represented by prefixing a 'B'.

Decimal Year

Fractional number of years AD represented by the date. 2000.0, or equivalently 1999.99 recurring, is midnight at the start of the first of January 2000. Because of leap years, the size of a unit depends on what year it is in.

Therefore midday on the 25th of October 2004 is `2004-10-25T12:00:00` in ISO 8601 format, 53303.5 as an MJD value, 2004.81588 as a Julian Epoch and 2004.81726 as a Besselian Epoch.

Currently this implementation cannot be relied upon to better than a millisecond.

`isoToMjd(isoDate)`

Converts an ISO8601 date string to Modified Julian Date. The basic format of the `isoDate` argument is `yyyy-mm-ddThh:mm:ss.s`, though some deviations from this form are permitted:

- The 'T' which separates date from time can be replaced by a space
- The seconds, minutes and/or hours can be omitted
- The decimal part of the seconds can be any length, and is optional
- A 'z' (which indicates UTC) may be appended to the time

Some legal examples are therefore: `"1994-12-21T14:18:23.2"`, `"1968-01-14"`, and `"2112-05-25 16:45Z"`.

- `isoDate (String)`: date in ISO 8601 format
- `return value (floating point)`: modified Julian date corresponding to `isoDate`

`dateToMjd(year, month, day, hour, min, sec)`

Converts a calendar date and time to Modified Julian Date.

- `year (integer)`: year AD
- `month (integer)`: index of month; January is 1, December is 12
- `day (integer)`: day of month (the first day is 1)

- `hour (integer)`: hour (0-23)
- `min (integer)`: minute (0-59)
- `sec (floating point)`: second (0<=sec<60)
- return value (*floating point*): modified Julian date corresponding to arguments

dateToMjd(year, month, day)

Converts a calendar date to Modified Julian Date.

- `year (integer)`: year AD
- `month (integer)`: index of month; January is 1, December is 12
- `day (integer)`: day of month (the first day is 1)
- return value (*floating point*): modified Julian date corresponding to 00:00:00 of the date specified by the arguments

decYearToMjd(decYear)

Converts a Decimal Year to a Modified Julian Date.

- `decYear (floating point)`: decimal year
- return value (*floating point*): modified Julian Date

mjdToIso(mjd)

Converts a Modified Julian Date value to an ISO 8601-format date-time string. The output format is `yyyy-mm-ddThh:mm:ss`.

- `mjd (floating point)`: modified Julian date
- return value (*String*): ISO 8601 format date corresponding to `mjd`

mjdToDate(mjd)

Converts a Modified Julian Date value to an ISO 8601-format date string. The output format is `yyyy-mm-dd`.

- `mjd (floating point)`: modified Julian date
- return value (*String*): ISO 8601 format date corresponding to `mjd`

mjdToTime(mjd)

Converts a Modified Julian Date value to an ISO 8601-format time-only string. The output format is `hh:mm:ss`.

- `mjd (floating point)`: modified Julian date
- return value (*String*): ISO 8601 format time corresponding to `mjd`

mjdToDecYear(mjd)

Converts a Modified Julian Date to Decimal Year.

- `mjd (floating point)`: modified Julian Date
- return value (*floating point*): decimal year

formatMjd(mjd, format)

Converts a Modified Julian Date value to a date using a customisable date format. The format is as defined by the `java.text.SimpleDateFormat` (<http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html>) class. The default output corresponds to the string `"yyyy-MM-dd 'T' HH:mm:ss"`

- `mjd (floating point)`: modified Julian date
- `format (String)`: formatting pattern
- return value (*String*): custom formatted time corresponding to `mjd`

mjdToJulian(mjd)

Converts a Modified Julian Date to Julian Epoch. For approximate purposes, the result of this routine consists of an integral part which gives the year AD and a fractional part which

represents the distance through that year, so that for instance 2000.5 is approximately 1 July 2000.

- `mjd` (*floating point*): modified Julian date
- return value (*floating point*): Julian epoch

julianToMjd(julianEpoch)

Converts a Julian Epoch to Modified Julian Date. For approximate purposes, the argument of this routine consists of an integral part which gives the year AD and a fractional part which represents the distance through that year, so that for instance 2000.5 is approximately 1 July 2000.

- `julianEpoch` (*floating point*): Julian epoch
- return value (*floating point*): modified Julian date

mjdToBesselian(mjd)

Converts Modified Julian Date to Besselian Epoch. For approximate purposes, the result of this routine consists of an integral part which gives the year AD and a fractional part which represents the distance through that year, so that for instance 1950.5 is approximately 1 July 1950.

- `mjd` (*floating point*): modified Julian date
- return value (*floating point*): Besselian epoch

besselianToMjd(besselianEpoch)

Converts Besselian Epoch to Modified Julian Date. For approximate purposes, the argument of this routine consists of an integral part which gives the year AD and a fractional part which represents the distance through that year, so that for instance 1950.5 is approximately 1 July 1950.

- `besselianEpoch` (*floating point*): Besselian epoch
- return value (*floating point*): modified Julian date

unixMillisToMjd(unixMillis)

Converts from milliseconds since the Unix epoch (1970-01-01T00:00:00) to a modified Julian date value

- `unixMillis` (*long integer*): milliseconds since the Unix epoch
- return value (*floating point*): modified Julian date

mjdToUnixMillis(mjd)

Converts from modified Julian date to milliseconds since the Unix epoch (1970-01-01T00:00:00).

- `mjd` (*floating point*): modified Julian date
- return value (*long integer*): milliseconds since the Unix epoch

10.5.7 TrigDegrees

Standard trigonometric functions with angles in degrees.

sinDeg(theta)

Sine of an angle.

- `theta` (*floating point*): an angle, in degrees
- return value (*floating point*): the sine of the argument

cosDeg(theta)

Cosine of an angle.

- `theta` (*floating point*): an angle, in degrees
- return value (*floating point*): the cosine of the argument

tanDeg(`theta`)

Tangent of an angle.

- `theta` (*floating point*): an angle, in degrees
- return value (*floating point*): the tangent of the argument.

asinDeg(`x`)

Arc sine. The result is in the range of -90 through 90.

- `x` (*floating point*): the value whose arc sine is to be returned.
- return value (*floating point*): the arc sine of the argument in degrees

acosDeg(`x`)

Arc cosine. The result is in the range of 0.0 through 180.

- `x` (*floating point*): the value whose arc cosine is to be returned.
- return value (*floating point*): the arc cosine of the argument in degrees

atanDeg(`x`)

Arc tangent. The result is in the range of -90 through 90.

- `x` (*floating point*): the value whose arc tangent is to be returned.
- return value (*floating point*): the arc tangent of the argument in degrees

atan2Deg(`y`, `x`)

Converts rectangular coordinates (`x`,`y`) to polar (`r`,`theta`). This method computes the phase `theta` by computing an arc tangent of `y/x` in the range of -180 to 180.

- `y` (*floating point*): the ordinate coordinate
- `x` (*floating point*): the abscissa coordinate
- return value (*floating point*): the `theta` component in degrees of the point (`r`,`theta`) in polar coordinates that corresponds to the point (`x`,`y`) in Cartesian coordinates.

10.5.8 Maths

Standard mathematical and trigonometric functions. Trigonometric functions work with angles in radians.

sin(`theta`)

Sine of an angle.

- `theta` (*floating point*): an angle, in radians.
- return value (*floating point*): the sine of the argument.

cos(`theta`)

Cosine of an angle.

- `theta` (*floating point*): an angle, in radians.
- return value (*floating point*): the cosine of the argument.

tan(`theta`)

Tangent of an angle.

- `theta` (*floating point*): an angle, in radians.

- return value (*floating point*): the tangent of the argument.

asin(x)

Arc sine of an angle. The result is in the range of $-pi/2$ through $pi/2$.

- *x* (*floating point*): the value whose arc sine is to be returned.
- return value (*floating point*): the arc sine of the argument (radians)

acos(x)

Arc cosine of an angle. The result is in the range of 0.0 through pi .

- *x* (*floating point*): the value whose arc cosine is to be returned.
- return value (*floating point*): the arc cosine of the argument (radians)

atan(x)

Arc tangent of an angle. The result is in the range of $-pi/2$ through $pi/2$.

- *x* (*floating point*): the value whose arc tangent is to be returned.
- return value (*floating point*): the arc tangent of the argument (radians)

exp(x)

Euler's number e raised to a power.

- *x* (*floating point*): the exponent to raise e to.
- return value (*floating point*): the value e^x , where e is the base of the natural logarithms.

log10(x)

Logarithm to base 10.

- *x* (*floating point*): argument
- return value (*floating point*): $\log_{10}(x)$

ln(x)

Natural logarithm.

- *x* (*floating point*): argument
- return value (*floating point*): $\log_e(x)$

sqrt(x)

Square root. The result is correctly rounded and positive.

- *x* (*floating point*): a value.
- return value (*floating point*): the positive square root of *x*. If the argument is NaN or less than zero, the result is NaN.

hypot(x, y)

Returns the square root of the sum of squares of its two arguments. Doing it like this may avoid intermediate overflow or underflow.

- *x* (*floating point*): a value
- *y* (*floating point*): a value
- return value (*floating point*): $\sqrt{x^2 + y^2}$

atan2(y, x)

Converts rectangular coordinates (*x*,*y*) to polar (*r*,*theta*). This method computes the phase *theta* by computing an arc tangent of *y*/*x* in the range of $-pi$ to pi .

- *y* (*floating point*): the ordinate coordinate
- *x* (*floating point*): the abscissa coordinate
- return value (*floating point*): the *theta* component (radians) of the point (*r*,*theta*) in polar coordinates that corresponds to the point (*x*,*y*) in Cartesian coordinates.

pow(a, b)

Exponentiation. The result is the value of the first argument raised to the power of the second argument.

- *a (floating point)*: the base.
- *b (floating point)*: the exponent.
- return value (*floating point*): the value a^b .

sinh(x)

Hyperbolic sine.

- *x (floating point)*: parameter
- return value (*floating point*): result

cosh(x)

Hyperbolic cosine.

- *x (floating point)*: parameter
- return value (*floating point*): result

tanh(x)

Hyperbolic tangent.

- *x (floating point)*: parameter
- return value (*floating point*): result

asinh(x)

Inverse hyperbolic sine.

- *x (floating point)*: parameter
- return value (*floating point*): result

acosh(x)

Inverse hyperbolic cosine.

- *x (floating point)*: parameter
- return value (*floating point*): result

atanh(x)

Inverse hyperbolic tangent.

- *x (floating point)*: parameter
- return value (*floating point*): result

E

Euler's number e , the base of natural logarithms.

PI

Pi , the ratio of the circumference of a circle to its diameter.

Infinity

Positive infinite floating point value.

NaN

Not-a-Number floating point value. Use with care; arithmetic and logical operations behave in strange ways near NaN (for instance, $\text{NaN} \neq \text{NaN}$). For most purposes this is equivalent to the blank value.

RANDOM

Evaluates to a random number in the range $0 \leq x < 1$. This is different for each cell of the table. The quality of the randomness may not be particularly good.

10.5.9 Arrays

Functions which operate on array-valued cells. The array parameters of these functions can only be used on values which are already arrays (usually, numeric arrays). In most cases that means on values in table columns which are declared as array-valued. FITS and VOTable tables can have columns which contain array values, but other formats such as CSV cannot.

If you want to calculate aggregating functions like sum, min, max etc on multiple values which are not part of an array, it's easier to use the functions from the `Lists` class.

Note that none of these functions will calculate statistical functions over a whole column of a table.

The functions fall into a number of categories:

- Aggregating operations, which map an array value to a scalar, including `size`, `count`, `countTrue`, `maximum`, `minimum`, `sum`, `mean`, `median`, `quantile`, `stdev`, `variance`, `join`.
- Operations on one or more arrays which produce array results, including `add`, `subtract`, `multiply`, `divide`, `reciprocal`, `condition`.
- The function `array`, which lets you assemble an array value from a list of scalar numbers. This can be used with the aggregating functions here, but it's generally easier to use the corresponding functions from the `Lists` class.

sum(array)

Returns the sum of all the non-blank elements in the array. If `array` is not a numeric array, `null` is returned.

- `array (Object)`: array of numbers
- return value (*floating point*): sum of all the numeric values in `array`

mean(array)

Returns the mean of all the non-blank elements in the array. If `array` is not a numeric array, `null` is returned.

- `array (Object)`: array of numbers
- return value (*floating point*): mean of all the numeric values in `array`

variance(array)

Returns the population variance of all the non-blank elements in the array. If `array` is not a numeric array, `null` is returned.

- `array (Object)`: array of numbers
- return value (*floating point*): variance of the numeric values in `array`

stdev(array)

Returns the population standard deviation of all the non-blank elements in the array. If `array` is not a numeric array, `null` is returned.

- `array (Object)`: array of numbers
- return value (*floating point*): standard deviation of the numeric values in `array`

minimum(array)

Returns the smallest of the non-blank elements in the array. If `array` is not a numeric array, `null` is returned.

- `array (Object)`: array of numbers
- return value (*floating point*): minimum of the numeric values in `array`

maximum(array)

Returns the largest of the non-blank elements in the array. If `array` is not a numeric array, `null` is returned.

- `array (Object)`: array of numbers
- return value (*floating point*): maximum of the numeric values in `array`

median(array)

Returns the median of the non-blank elements in the array. If `array` is not a numeric array, `null` is returned.

- `array (Object)`: array of numbers
- return value (*floating point*): median of the numeric values in `array`

quantile(array, quant)

Returns a quantile value of the non-blank elements in the array. Which quantile is determined by the `quant` value; values of 0, 0.5 and 1 give the minimum, median and maximum respectively. A value of 0.99 would give the 99th percentile.

- `array (Object)`: array of numbers
- `quant (floating point)`: number in the range 0-1 determining which quantile to calculate
- return value (*floating point*): quantile corresponding to `quant`

size(array)

Returns the number of elements in the array. If `array` is not an array, zero is returned.

- `array (Object)`: array
- return value (*integer*): size of `array`

count(array)

Returns the number of non-blank elements in the array. If `array` is not an array, zero is returned.

- `array (Object)`: array (may or may not be numeric)
- return value (*integer*): number of non-blank elements in `array`

countTrue(array)

Returns the number of true elements in an array of boolean values.

- `array (array of boolean)`: array of true/false values
- return value (*integer*): number of true values in `array`

join(array, joiner)

Returns a string composed of concatenating all the elements of an array, separated by a joiner string. If `array` is not an array, `null` is returned.

- `array (Object)`: array of numbers or strings
- `joiner (String)`: text string to interpose between adjacent elements
- return value (*String*): string composed of `array` elements separated by `joiner` strings

add(array1, array2)

Returns the result of adding two numeric arrays element by element. Both arrays must be numeric, and the arrays must have the same length. If either of those conditions is not true, `null` is returned. The types of the arrays do not need to be the same, so for example it is

permitted to add an integer array to a floating point array.

- `array1 (Object)`: first array of numeric values
- `array2 (Object)`: second array of numeric values
- `return value (array of floating point)`: element-by-element sum of `array1` and `array2`, the same length as the input arrays

add(array, constant)

Returns the result of adding a constant value to every element of a numeric array. If the supplied `array` argument is not a numeric array, `null` is returned.

- `array (Object)`: array input
- `constant (floating point)`: value to add to each array element
- `return value (array of floating point)`: array output, the same length as the `array` parameter

subtract(array1, array2)

Returns the result of subtracting one numeric array from the other element by element. Both arrays must be numeric, and the arrays must have the same length. If either of those conditions is not true, `null` is returned. The types of the arrays do not need to be the same, so for example it is permitted to subtract an integer array from a floating point array.

- `array1 (Object)`: first array of numeric values
- `array2 (Object)`: second array of numeric values
- `return value (array of floating point)`: element-by-element difference of `array1` and `array2`, the same length as the input arrays

multiply(array1, array2)

Returns the result of multiplying two numeric arrays element by element. Both arrays must be numeric, and the arrays must have the same length. If either of those conditions is not true, `null` is returned. The types of the arrays do not need to be the same, so for example it is permitted to multiply an integer array by a floating point array.

- `array1 (Object)`: first array of numeric values
- `array2 (Object)`: second array of numeric values
- `return value (array of floating point)`: element-by-element product of `array1` and `array2`, the same length as the input arrays

multiply(array, constant)

Returns the result of multiplying every element of a numeric array by a constant value. If the supplied `array` argument is not a numeric array, `null` is returned.

- `array (Object)`: array input
- `constant (floating point)`: value by which to multiply each array element
- `return value (array of floating point)`: array output, the same length as the `array` parameter

divide(array1, array2)

Returns the result of dividing two numeric arrays element by element. Both arrays must be numeric, and the arrays must have the same length. If either of those conditions is not true, `null` is returned. The types of the arrays do not need to be the same, so for example it is permitted to divide an integer array by a floating point array.

- `array1 (Object)`: array of numerator values (numeric)
- `array2 (Object)`: array of denominator values (numeric)
- `return value (array of floating point)`: element-by-element result of `array1[i]/array2[i]` the same length as the input arrays

reciprocal(array)

Returns the result of taking the reciprocal of every element of a numeric array. If the supplied `array` argument is not a numeric array, `null` is returned.

- `array (Object)`: array input
- `return value (array of floating point)`: array output, the same length as the `array` parameter

condition(flagArray, trueValue, falseValue)

Maps a boolean array to a numeric array by using supplied numeric values to represent true and false values from the input array.

This has the same effect as applying the expression `outArray[i] = flagArray[i] ? trueValue : falseValue`.

- `flagArray (array of boolean)`: array of boolean values
- `trueValue (floating point)`: output value corresponding to an input true value
- `falseValue (floating point)`: output value corresponding to an input false value
- `return value (array of floating point)`: output numeric array, same length as `flagArray`

array(values, ...)

Returns a floating point numeric array built from the given arguments.

- `values (floating point, one or more)`: one or more array elements
- `return value (array of floating point)`: array

intArray(values, ...)

Returns an integer numeric array built from the given arguments.

- `values (integer, one or more)`: one or more array elements
- `return value (array of integer)`: array

stringArray(values, ...)

Returns a String array built from the given arguments.

- `values (String, one or more)`: one or more array elements
- `return value (array of String)`: array

10.5.10 Fluxes

Functions for conversion between flux and magnitude values. Functions are provided for conversion between flux in Janskys and AB magnitudes.

Some constants for approximate conversions between different magnitude scales are also provided:

- Constants `JOHNSON_AB_*`, for Johnson <-> AB magnitude conversions, from Frei and Gunn, *Astronomical Journal* 108, 1476 (1994), Table 2 (1994AJ....108.1476F (<http://adsabs.harvard.edu/abs/1994AJ....108.1476F>)).
- Constants `VEGA_AB_*`, for Vega <-> AB magnitude conversions, from Blanton et al., *Astronomical Journal* 129, 2562 (2005), Eqs. (5) (2005AJ....129.2562B (<http://adsabs.harvard.edu/abs/2005AJ....129.2562B>)).

abToJansky(magAB)

Converts AB magnitude to flux in Jansky.

$$F/Jy = 10^{(23 - (AB + 48.6)) / 2.5}$$

- `magAB (floating point)`: AB magnitude value
- `return value (floating point)`: equivalent flux in Jansky

janskyToAb(fluxJansky)

Converts flux in Jansky to AB magnitude.

$$AB = 2.5 * (23 - \log_{10}(F/Jy)) - 48.6$$

- *fluxJansky (floating point)*: flux in Jansky
- *return value (floating point)*: equivalent AB magnitude

luminosityToFlux(lumin, dist)

Converts luminosity to flux given a luminosity distance.

$$F = \text{lumin} / (4 \times \text{Pi} \times \text{dist}^2)$$

- *lumin (floating point)*: luminosity
- *dist (floating point)*: luminosity distance
- *return value (floating point)*: equivalent flux

fluxToLuminosity(flux, dist)

Converts flux to luminosity given a luminosity distance.

$$\text{lumin} = (4 \times \text{Pi} \times \text{dist}^2) F$$

- *flux (floating point)*: flux
- *dist (floating point)*: luminosity distance
- *return value (floating point)*: equivalent luminosity

JOHNSON_AB_V

Approximate offset between Johnson and AB magnitudes in V band.

$$V_J \sim V_{AB} + \text{JOHNSON_AB_V.}$$

JOHNSON_AB_B

Approximate offset between Johnson and AB magnitudes in B band.

$$B_J \sim B_{AB} + \text{JOHNSON_AB_B.}$$

JOHNSON_AB_Bj

Approximate offset between Johnson and AB magnitudes in Bj band.

$$B_{jJ} \sim B_{jAB} + \text{JOHNSON_AB_Bj.}$$

JOHNSON_AB_R

Approximate offset between Johnson and AB magnitudes in R band.

$$R_J \sim R_{AB} + \text{JOHNSON_AB_R.}$$

JOHNSON_AB_I

Approximate offset between Johnson and AB magnitudes in I band. $I_J \sim I_{AB} + \text{JOHNSON_AB_I.}$

JOHNSON_AB_g

Approximate offset between Johnson and AB magnitudes in g band. $g_J \sim g_{AB} + \text{JOHNSON_AB_g.}$

JOHNSON_AB_r

Approximate offset between Johnson and AB magnitudes in r band. $r_J \sim r_{AB} + \text{JOHNSON_AB_r.}$

JOHNSON_AB_i

Approximate offset between Johnson and AB magnitudes in i band. $i_J \sim i_{AB} + \text{JOHNSON_AB_i.}$

JOHNSON_AB_Rc

Approximate offset between Johnson and AB magnitudes in Rc band.
 $R_{cJ} \sim R_{cAB} + \text{JOHNSON_AB_Rc}.$

JOHNSON_AB_Ic

Approximate offset between Johnson and AB magnitudes in Ic band.
 $I_{cJ} \sim I_{cAB} + \text{JOHNSON_AB_Ic}.$

JOHNSON_AB_uPrime

Offset between Johnson and AB magnitudes in u' band (zero).
 $u'_J = u'_{AB} + \text{JOHNSON_AB_uPrime} = u'_{AB}.$

JOHNSON_AB_gPrime

Offset between Johnson and AB magnitudes in g' band (zero).
 $g'_J = g'_{AB} + \text{JOHNSON_AB_gPrime} = g'_{AB}.$

JOHNSON_AB_rPrime

Offset between Johnson and AB magnitudes in r' band (zero).
 $r'_J = r'_{AB} + \text{JOHNSON_AB_rPrime} = r'_{AB}.$

JOHNSON_AB_iPrime

Offset between Johnson and AB magnitudes in i' band (zero).
 $i'_J = i'_{AB} + \text{JOHNSON_AB_iPrime} = i'_{AB}.$

JOHNSON_AB_zPrime

Offset between Johnson and AB magnitudes in z' band (zero).
 $z'_J = z'_{AB} + \text{JOHNSON_AB_zPrime} = z'_{AB}.$

VEGA_AB_J

Approximate offset between Vega (as in 2MASS) and AB magnitudes in J band.
 $J_{\text{Vega}} \sim J_{AB} + \text{VEGA_AB_J}.$

VEGA_AB_H

Approximate offset between Vega (as in 2MASS) and AB magnitudes in H band.
 $H_{\text{Vega}} \sim H_{AB} + \text{VEGA_AB_H}.$

VEGA_AB_K

Approximate offset between Vega (as in 2MASS) and AB magnitudes in K band.
 $K_{\text{Vega}} \sim K_{AB} + \text{VEGA_AB_K}.$

10.5.11 Strings

String manipulation and query functions.

concat(strings, ...)

Concatenates multiple values into a string. In some cases the same effect can be achieved by writing $s1+s2+\dots$, but this method makes sure that values are converted to strings, with the blank value invisible.

- *strings (Object, one or more)*: one or more strings
- *return value (String)*: concatenation of input strings, without separators

join(separator, words, ...)

Joins multiple values into a string, with a given separator between each pair.

- *separator (String)*: string to insert between adjacent words
- *words (Object, one or more)*: one or more values to join
- *return value (String)*: input values joined together with *separator*

equals(s1, s2)

Determines whether two strings are equal. Note you should use this function instead of `s1==s2`, which can (for technical reasons) return false even if the strings are the same.

- *s1 (String)*: first string
- *s2 (String)*: second string
- *return value (boolean)*: true if *s1* and *s2* are both blank, or have the same content

equalsIgnoreCase(s1, s2)

Determines whether two strings are equal apart from possible upper/lower case distinctions.

- *s1 (String)*: first string
- *s2 (String)*: second string
- *return value (boolean)*: true if *s1* and *s2* are both blank, or have the same content apart from case folding

startsWith(whole, start)

Determines whether a string starts with a certain substring.

- *whole (String)*: the string to test
- *start (String)*: the sequence that may appear at the start of *whole*
- *return value (boolean)*: true if the first few characters of *whole* are the same as *start*

endsWith(whole, end)

Determines whether a string ends with a certain substring.

- *whole (String)*: the string to test
- *end (String)*: the sequence that may appear at the end of *whole*
- *return value (boolean)*: true if the last few characters of *whole* are the same as *end*

contains(whole, sub)

Determines whether a string contains a given substring.

- *whole (String)*: the string to test
- *sub (String)*: the sequence that may appear within *whole*
- *return value (boolean)*: true if the sequence *sub* appears within *whole*

length(str)

Returns the length of a string in characters.

- *str (String)*: string
- *return value (integer)*: number of characters in *str*

split(words)

Splits a string into an array of space-separated words. One or more spaces separates each word from the next. Leading and trailing spaces are ignored.

The result is an array of strings, and if you want to use the individual elements you need to use square-bracket indexing, with `[0]` representing the first object

- *words (String)*: string with embedded spaces delimiting the words
- *return value (array of String)*: array of the separate words; you can extract the individual words from the result using square bracket indexing

split(words, regex)

Splits a string into an array of words separated by a given regular expression.

The result is an array of strings, and if you want to use the individual elements you need to use square-bracket indexing, with `[0]` representing the first object

- `words (String)`: string with multiple parts
- `regex (String)`: regular expression delimiting the different words in the `words` parameter
- return value (*array of String*): array of the separate words; you can extract the individual words from the result using square bracket indexing

matches(str, regex)

Tests whether a string matches a given regular expression.

- `str (String)`: string to test
- `regex (String)`: regular expression string
- return value (*boolean*): true if `regex` matches `str` anywhere

matchGroup(str, regex)

Returns the first grouped expression matched in a string defined by a regular expression. A grouped expression is one enclosed in parentheses.

- `str (String)`: string to match against
- `regex (String)`: regular expression containing a grouped section
- return value (*String*): contents of the matched group (or null, if `regex` didn't match `str`)

replaceFirst(str, regex, replacement)

Replaces the first occurrence of a regular expression in a string with a different substring value.

- `str (String)`: string to manipulate
- `regex (String)`: regular expression to match in `str`
- `replacement (String)`: replacement string
- return value (*String*): same as `str`, but with the first match (if any) of `regex` replaced by `replacement`

replaceAll(str, regex, replacement)

Replaces all occurrences of a regular expression in a string with a different substring value.

- `str (String)`: string to manipulate
- `regex (String)`: regular expression to match in `str`
- `replacement (String)`: replacement string
- return value (*String*): same as `str`, but with all matches of `regex` replaced by `replacement`

substring(str, startIndex)

Returns the last part of a given string. The substring begins with the character at the specified index and extends to the end of this string.

- `str (String)`: the input string
- `startIndex (integer)`: the beginning index, inclusive
- return value (*String*): last part of `str`, omitting the first `startIndex` characters

substring(str, startIndex, endIndex)

Returns a substring of a given string. The substring begins with the character at `startIndex` and continues to the character at index `endIndex-1`. Thus the length of the substring is `endIndex-startIndex`.

- `str (String)`: the input string
- `startIndex (integer)`: the beginning index, inclusive

- `endIndex (integer)`: the end index, inclusive
- return value (*String*): substring of `str`

toUpperCase(str)

Returns an uppercased version of a string.

- `str (String)`: input string
- return value (*String*): uppercased version of `str`

toLowerCase(str)

Returns a lowercased version of a string.

- `str (String)`: input string
- return value (*String*): lowercased version of `str`

trim(str)

Trims whitespace from both ends of a string.

- `str (String)`: input string
- return value (*String*): `str` with any spaces trimmed from start and finish

padWithZeros(value, ndigit)

Takes an integer argument and returns a string representing the same numeric value but padded with leading zeros to a specified length.

- `value (long integer)`: numeric value to pad
- `ndigit (integer)`: the number of digits in the resulting string
- return value (*String*): a string evaluating to the same as `value` with at least `ndigit` characters

10.5.12 Formats

Functions for formatting numeric values.

formatDecimal(value, dp)

Turns a floating point value into a string with a given number of decimal places using standard settings.

- `value (floating point)`: value to format
- `dp (integer)`: number of decimal places (digits after the decimal point)
- return value (*String*): formatted string

formatDecimalLocal(value, dp)

Turns a floating point value into a string using current locale settings. For instance if language is set to French, decimal points will be represented as a comma "," instead of a full stop ".". Otherwise behaves the same as the corresponding `formatDecimal` function.

- `value (floating point)`: value to format
- `dp (integer)`: number of decimal places (digits after the decimal point)
- return value (*String*): formatted string

formatDecimal(value, format)

Turns a floating point value into a formatted string using standard settings. The `format` string is as defined by Java's `java.text.DecimalFormat` (<http://docs.oracle.com/javase/6/docs/api/java/text/DecimalFormat.html>) class.

- `value (floating point)`: value to format
- `format (String)`: format specifier

- return value (*String*): formatted string

formatDecimalLocal(value, format)

Turns a floating point value into a formatted string using current locale settings. For instance if language is set to French, decimal points will be represented as a comma "," instead of a full stop ".". Otherwise behaves the same as the corresponding `formatDecimal` function.

- *value (floating point)*: value to format
- *format (String)*: format specifier
- return value (*String*): formatted string

10.5.13 CoordsRadians

Functions for angle transformations and manipulations, based on radians rather than degrees. In particular, methods for translating between radians and HH:MM:SS.S or DDD:MM:SS.S type sexagesimal representations are provided.

radiansToDms(rad)

Converts an angle in radians to a formatted degrees:minutes:seconds string. No fractional part of the seconds field is given.

- *rad (floating point)*: angle in radians
- return value (*String*): DMS-format string representing *rad*

radiansToDms(rad, secFig)

Converts an angle in radians to a formatted degrees:minutes:seconds string with a given number of decimal places in the seconds field.

- *rad (floating point)*: angle in radians
- *secFig (integer)*: number of decimal places in the seconds field
- return value (*String*): DMS-format string representing *rad*

radiansToHms(rad)

Converts an angle in radians to a formatted hours:minutes:seconds string. No fractional part of the seconds field is given.

- *rad (floating point)*: angle in radians
- return value (*String*): HMS-format string representing *rad*

radiansToHms(rad, secFig)

Converts an angle in radians to a formatted hours:minutes:seconds string with a given number of decimal places in the seconds field.

- *rad (floating point)*: angle in radians
- *secFig (integer)*: number of decimal places in the seconds field
- return value (*String*): HMS-format string representing *rad*

dmsToRadians(dms)

Converts a formatted degrees:minutes:seconds string to an angle in radians. Delimiters may be colon, space, characters `dm[s]`, or some others. Additional spaces and leading +/- are permitted. The :seconds part is optional.

- *dms (String)*: formatted DMS string
- return value (*floating point*): angle in radians specified by *dms*

hmsToRadians(hms)

Converts a formatted hours:minutes:seconds string to an angle in radians. Delimiters may be

colon, space, characters `hm[s]`, or some others. Additional spaces and leading `+/-` are permitted. The `:seconds` part is optional.

- `hms` (*String*): formatted HMS string
- return value (*floating point*): angle in radians specified by `hms`

dmsToRadians(deg, min, sec)

Converts degrees, minutes, seconds to an angle in radians.

In conversions of this type, one has to be careful to get the sign right in converting angles which are between 0 and -1 degrees. This routine uses the sign bit of the `deg` argument, taking care to distinguish between +0 and -0 (their internal representations are different for floating point values). It is illegal for the `min` or `sec` arguments to be negative.

- `deg` (*floating point*): degrees part of angle
- `min` (*floating point*): minutes part of angle
- `sec` (*floating point*): seconds part of angle
- return value (*floating point*): specified angle in radians

hmsToRadians(hour, min, sec)

Converts hours, minutes, seconds to an angle in radians.

In conversions of this type, one has to be careful to get the sign right in converting angles which are between 0 and -1 hours. This routine uses the sign bit of the `hour` argument, taking care to distinguish between +0 and -0 (their internal representations are different for floating point values).

- `hour` (*floating point*): degrees part of angle
- `min` (*floating point*): minutes part of angle
- `sec` (*floating point*): seconds part of angle
- return value (*floating point*): specified angle in radians

skyDistanceRadians(ra1, dec1, ra2, dec2)

Calculates the separation (distance around a great circle) of two points on the sky in radians.

- `ra1` (*floating point*): right ascension of point 1 in radians
- `dec1` (*floating point*): declination of point 1 in radians
- `ra2` (*floating point*): right ascension of point 2 in radians
- `dec2` (*floating point*): declination of point 2 in radians
- return value (*floating point*): angular distance between point 1 and point 2 in radians

hoursToRadians(hours)

Converts hours to radians.

- `hours` (*floating point*): angle in hours
- return value (*floating point*): angle in radians

degreesToRadians(deg)

Converts degrees to radians.

- `deg` (*floating point*): angle in degrees
- return value (*floating point*): angle in radians

radiansToDegrees(rad)

Converts radians to degrees.

- `rad` (*floating point*): angle in radians
- return value (*floating point*): angle in degrees

raFK4toFK5radians(raFK4, decFK4)

Converts a B1950.0 FK4 position to J2000.0 FK5 at an epoch of B1950.0 yielding Right

Ascension. This assumes zero proper motion in the FK5 frame.

- `raFK4` (*floating point*): right ascension in B1950.0 FK4 system (radians)
- `decFK4` (*floating point*): declination in B1950.0 FK4 system (radians)
- return value (*floating point*): right ascension in J2000.0 FK5 system (radians)

decFK4toFK5radians(raFK4, decFK4)

Converts a B1950.0 FK4 position to J2000.0 FK5 at an epoch of B1950.0 yielding Declination. This assumes zero proper motion in the FK5 frame.

- `raFK4` (*floating point*): right ascension in B1950.0 FK4 system (radians)
- `decFK4` (*floating point*): declination in B1950.0 FK4 system (radians)
- return value (*floating point*): declination in J2000.0 FK5 system (radians)

raFK5toFK4radians(raFK5, decFK5)

Converts a J2000.0 FK5 position to B1950.0 FK4 at an epoch of B1950.0 yielding Declination. This assumes zero proper motion, parallax and radial velocity in the FK5 frame.

- `raFK5` (*floating point*): right ascension in J2000.0 FK5 system (radians)
- `decFK5` (*floating point*): declination in J2000.0 FK5 system (radians)
- return value (*floating point*): right ascension in the FK4 system (radians)

decFK5toFK4radians(raFK5, decFK5)

Converts a J2000.0 FK5 position to B1950.0 FK4 at an epoch of B1950.0 yielding Declination. This assumes zero proper motion, parallax and radial velocity in the FK5 frame.

- `raFK5` (*floating point*): right ascension in J2000.0 FK5 system (radians)
- `decFK5` (*floating point*): declination in J2000.0 FK5 system (radians)
- return value (*floating point*): right ascension in the FK4 system (radians)

raFK4toFK5Radians(raFK4, decFK4, beepoch)

Converts a B1950.0 FK4 position to J2000.0 FK5 yielding Right Ascension. This assumes zero proper motion in the FK5 frame. The `beepoch` parameter is the epoch at which the position in the FK4 frame was determined.

- `raFK4` (*floating point*): right ascension in B1950.0 FK4 system (radians)
- `decFK4` (*floating point*): declination in B1950.0 FK4 system (radians)
- `beepoch` (*floating point*): Besselian epoch
- return value (*floating point*): right ascension in J2000.0 FK5 system (radians)

decFK4toFK5Radians(raFK4, decFK4, beepoch)

Converts a B1950.0 FK4 position to J2000.0 FK5 yielding Declination. This assumes zero proper motion in the FK5 frame. The `beepoch` parameter is the epoch at which the position in the FK4 frame was determined.

- `raFK4` (*floating point*): right ascension in B1950.0 FK4 system (radians)
- `decFK4` (*floating point*): declination in B1950.0 FK4 system (radians)
- `beepoch` (*floating point*): Besselian epoch
- return value (*floating point*): declination in J2000.0 FK5 system (radians)

raFK5toFK4Radians(raFK5, decFK5, beepoch)

Converts a J2000.0 FK5 position to B1950.0 FK4 yielding Declination. This assumes zero proper motion, parallax and radial velocity in the FK5 frame.

- `raFK5` (*floating point*): right ascension in J2000.0 FK5 system (radians)
- `decFK5` (*floating point*): declination in J2000.0 FK5 system (radians)
- `beepoch` (*floating point*): Besselian epoch
- return value (*floating point*): right ascension in the FK4 system (radians)

decFK5toFK4Radians(raFK5, decFK5, beepoch)

Converts a J2000.0 FK5 position to B1950.0 FK4 yielding Declination. This assumes zero proper motion, parallax and radial velocity in the FK5 frame.

- `raFK5` (*floating point*): right ascension in J2000.0 FK5 system (radians)
- `decFK5` (*floating point*): declination in J2000.0 FK5 system (radians)
- `bepoch` (*floating point*): Besselian epoch
- return value (*floating point*): right ascension in the FK4 system (radians)

DEGREE_RADIANS

The size of one degree in radians.

HOURL_RADIANS

The size of one hour of right ascension in radians.

ARC_MINUTE_RADIANS

The size of one arcminute in radians.

ARC_SECOND_RADIANS

The size of one arcsecond in radians.

10.5.14 Coverage

Functions related to coverage and footprints.

One coverage standard is Multi-Order Coverage maps, described at <http://www.ivoa.net/Documents/MOC/> (<http://www.ivoa.net/Documents/MOC/>). MOC positions are always defined in ICRS equatorial coordinates.

MOC locations may be given as either the filename or the URL of a MOC FITS file. Alternatively, they may be the identifier of a VizieR table, for instance "`V/139/sdss9`" (SDSS DR9). A list of all the MOCs available from VizieR can currently be found at <http://alasky.u-strasbg.fr/footprints/tables/vizier/> (<http://alasky.u-strasbg.fr/footprints/tables/vizier/>). You can search for VizieR table identifiers from the VizieR web page (<http://vizier.u-strasbg.fr/>) (<http://vizier.u-strasbg.fr/>); note you must use the *table* identifier (like "`V/139/sdss9`") and not the *catalogue* identifier (like "`V/139`").

`inMoc(mocLocation, ra, dec)`

Indicates whether a given sky position falls strictly within a given MOC (Multi-Order Coverage map). If the given `mocLocation` value does not represent a MOC (for instance no file exists or the file is not in MOC format) a warning will be issued the first time it's referenced, and the result will be false.

- `mocLocation` (*String*): location of a FITS MOC file: a filename, a URL, or a VizieR table name
- `ra` (*floating point*): ICRS right ascension in degrees
- `dec` (*floating point*): ICRS declination in degrees
- return value (*boolean*): true iff the given position falls within the given MOC

`nearMoc(mocLocation, ra, dec, distanceDeg)`

Indicates whether a given sky position either falls within, or is within a certain distance of the edge of, a given MOC (Multi-Order Coverage map). If the given `mocLocation` value does not represent a MOC (for instance no file exists or the file is not in MOC format) a warning will be issued the first time it's referenced, and the result will be false.

- `mocLocation (String)`: location of a FITS MOC file: a filename, a URL, or a Vizier table name
- `ra (floating point)`: ICRS right ascension in degrees
- `dec (floating point)`: ICRS declination in degrees
- `distanceDeg (floating point)`: permitted distance from MOC boundary in degrees
- return value (*boolean*): true iff the given position is within `distance` degrees of the given MOC

10.5.15 Lists

Functions which operate on lists of values.

Some of these resemble similar functions in the `Arrays` class, and in some cases are interchangeable, but these are easier to use on non-array values because you don't have to explicitly wrap up lists of arguments as an array. However, for implementation reasons, most of the functions defined here can be used on values which are already `double[]` arrays (for instance array-valued columns) rather than as comma-separated lists of floating point values.

sum(values, ...)

Returns the sum of all the non-blank supplied arguments.

- `values (floating point, one or more)`: one or more numeric values
- return value (*floating point*): sum of `values`

mean(values, ...)

Returns the mean of all the non-blank supplied arguments.

- `values (floating point, one or more)`: one or more numeric values
- return value (*floating point*): mean of `values`

variance(values, ...)

Returns the population variance of the non-blank supplied arguments.

- `values (floating point, one or more)`: one or more numeric values
- return value (*floating point*): population variance of `values`

stdev(values, ...)

Returns the population standard deviation of the non-blank supplied arguments.

- `values (floating point, one or more)`: one or more numeric values
- return value (*floating point*): population standard deviation of `values`

min(values, ...)

Returns the minimum of all the non-blank supplied arguments.

- `values (floating point, one or more)`: one or more numeric values
- return value (*floating point*): minimum of `values`

max(values, ...)

Returns the maximum of all the non-blank supplied arguments.

- `values (floating point, one or more)`: one or more numeric values
- return value (*floating point*): maximum of `values`

median(values, ...)

Returns the median of all the non-blank supplied arguments.

- `values (floating point, one or more)`: one or more numeric values

- return value (*floating point*): median of `values`

countTrue(values, ...)

Returns the number of true values in a list of boolean arguments. Note if any of the values are blank, the result may be blank as well.

- `values` (*boolean, one or more*): one or more true/false values
- return value (*integer*): number of elements of `values` that are true

10.5.16 CoordsDegrees

Functions for angle transformations and manipulations, with angles generally in degrees. In particular, methods for translating between degrees and HH:MM:SS.S or DDD:MM:SS.S type sexagesimal representations are provided.

degreesToDms(deg)

Converts an angle in degrees to a formatted degrees:minutes:seconds string. No fractional part of the seconds field is given.

- `deg` (*floating point*): angle in degrees
- return value (*String*): DMS-format string representing `deg`

degreesToDms(deg, secFig)

Converts an angle in degrees to a formatted degrees:minutes:seconds string with a given number of decimal places in the seconds field.

- `deg` (*floating point*): angle in degrees
- `secFig` (*integer*): number of decimal places in the seconds field
- return value (*String*): DMS-format string representing `deg`

degreesToHms(deg)

Converts an angle in degrees to a formatted hours:minutes:seconds string. No fractional part of the seconds field is given.

- `deg` (*floating point*): angle in degrees
- return value (*String*): HMS-format string representing `deg`

degreesToHms(deg, secFig)

Converts an angle in degrees to a formatted hours:minutes:seconds string with a given number of decimal places in the seconds field.

- `deg` (*floating point*): angle in degrees
- `secFig` (*integer*): number of decimal places in the seconds field
- return value (*String*): HMS-format string representing `deg`

dmsToDegrees(dms)

Converts a formatted degrees:minutes:seconds string to an angle in degrees. Delimiters may be colon, space, characters `dm[s]`, or some others. Additional spaces and leading +/- are permitted. The :seconds part is optional.

- `dms` (*String*): formatted DMS string
- return value (*floating point*): angle in degrees specified by `dms`

hmsToDegrees(hms)

Converts a formatted hours:minutes:seconds string to an angle in degrees. Delimiters may be colon, space, characters `hm[s]`, or some others. Additional spaces and leading +/- are permitted. The :seconds part is optional.

- `hms` (*String*): formatted HMS string
- return value (*floating point*): angle in degrees specified by `hms`

dmsToDegrees(deg, min, sec)

Converts degrees, minutes, seconds to an angle in degrees.

In conversions of this type, one has to be careful to get the sign right in converting angles which are between 0 and -1 degrees. This routine uses the sign bit of the `deg` argument, taking care to distinguish between +0 and -0 (their internal representations are different for floating point values). It is illegal for the `min` or `sec` arguments to be negative.

- `deg` (*floating point*): degrees part of angle
- `min` (*floating point*): minutes part of angle
- `sec` (*floating point*): seconds part of angle
- return value (*floating point*): specified angle in degrees

hmsToDegrees(hour, min, sec)

Converts hours, minutes, seconds to an angle in degrees.

In conversions of this type, one has to be careful to get the sign right in converting angles which are between 0 and -1 hours. This routine uses the sign bit of the `hour` argument, taking care to distinguish between +0 and -0 (their internal representations are different for floating point values).

- `hour` (*floating point*): degrees part of angle
- `min` (*floating point*): minutes part of angle
- `sec` (*floating point*): seconds part of angle
- return value (*floating point*): specified angle in degrees

skyDistanceDegrees(ra1, dec1, ra2, dec2)

Calculates the separation (distance around a great circle) of two points on the sky in degrees.

- `ra1` (*floating point*): right ascension of point 1 in degrees
- `dec1` (*floating point*): declination of point 1 in degrees
- `ra2` (*floating point*): right ascension of point 2 in degrees
- `dec2` (*floating point*): declination of point 2 in degrees
- return value (*floating point*): angular distance between point 1 and point 2 in degrees

10.6 Examples

Here are some examples for defining new columns; the expressions below could appear as the `<expr>` in a `tpipe addcol` or `sortexpr` command).

Average

```
(first + second) * 0.5
```

Square root

```
sqrt(variance)
```

Angle conversion

```
radiansToDegrees(DEC_radians)
degreesToRadians(RA_degrees)
```

Conversion from string to number


```
parseInt($12)
parseDouble(ident)
```

Conversion from number to string

```
toString(index)
```

Conversion between numeric types

```
toShort(obs_type)
toDouble(range)
```

or

```
(short) obs_type
(double) range
```

Conversion from sexagesimal to degrees

```
hmsToDegrees(RA1950)
dmsToDegrees(decDeg,decMin,decSec)
```

Conversion from degrees to sexagesimal

```
degreesToDms($3)
degreesToHms(RA,2)
```

Outlier clipping

```
min(1000, max(value, 0))
```

Converting a magic value to null

```
jmag == 9999 ? NULL : jmag
```

Converting a null value to a magic one

```
NULL_jmag ? 9999 : jmag
```

Taking the third scalar element from an array-valued column

```
psfCounts[2]
```

and here are some examples of boolean expressions that could be used for row selection (appearing in a `tpipe select` command)

Within a numeric range

```
RA > 100 && RA < 120 && Dec > 75 && Dec < 85
```

Within a circle

```
$2*$2 + $3*$3 < 1
skyDistanceDegrees(ra0,dec0,hmsToDegrees(RA),dmsToDegrees(DEC))<15./3600.
```

First 100 rows

```
index <= 100
```

(though you could use `tpipe cmd='head 100'` instead)

Every tenth row

```
index % 10 == 0
```

(though you could use `tpipe cmd='every 10'` instead)

String equality/matching

```
equals(SECTOR, "ZZ9 Plural Z Alpha")
equalsIgnoreCase(SECTOR, "zz9 plural z alpha")
startsWith(SECTOR, "ZZ")
contains(ph_qual, "U")
```

String regular expression matching

```
matches(SECTOR, "[XYZ] Alpha")
```

Test for non-blank value

```
! NULL_ellipticity
```

10.7 Advanced Topics

This section contains some notes on getting the most out of the algebraic expressions facility. If you're not a Java programmer, some of the following may be a bit daunting - read on at your own risk!

10.7.1 Expression evaluation

This note provides a bit more detail for Java programmers on what is going on here; it describes how the use of functions in STILTS algebraic expressions relates to normal Java code.

The expressions which you write are compiled to Java bytecode when you enter them (if there is a 'compilation error' it will be reported straight away). The functions listed in the previous subsections are all the `public static` methods of the classes which are made available by default. The classes listed are all in the package `uk.ac.starlink.ttools.func`. However, the `public static` methods are all imported into an anonymous namespace for bytecode compilation, so that you write `(sqrt(x,y))` and not `Maths.sqrt(x,y)`. The same happens to other classes that are imported (which can be in any package or none) - their `public static` methods all go into the anonymous namespace. Thus, method name clashes are a possibility.

This cleverness is all made possible by the rather wonderful JEL (<http://www.gnu.org/software/jel/>).

10.7.2 Instance Methods

There is another category of functions which can be used apart from those listed in Section 10.5. These are called, in Java/object-oriented parlance, "instance methods" and represent functions that can be executed on an object.

It is possible to invoke any of its public instance methods on any object (though not on primitive values - numeric and boolean ones). The syntax is that you place a `"."` followed by the method invocation after the object you want to invoke the method on, hence `NAME.substring(3)` instead of `substring(NAME,3)`. If you know what you're doing, feel free to go ahead and do this. However, most of the instance methods you're likely to want to use have equivalents in the normal functions listed in the previous section, so unless you're a Java programmer or feeling adventurous, you may

be best off ignoring this feature.

10.7.3 Adding User-Defined Functions

The functions provided by default for use with algebraic expressions, while powerful, may not provide all the operations you need. For this reason, it is possible to write your own extensions to the expression language. In this way you can specify arbitrarily complicated functions. Note however that this will only allow you to define new columns or subsets where each cell is a function only of the other cells in the same row - it will not allow values in one row to be functions of values in another.

In order to do this, you have to write and compile a (probably short) program in the Java language. A full discussion of how to go about this is beyond the scope of this document, so if you are new to Java and/or programming you may need to find a friendly local programmer to assist (or mail the author). The following explanation is aimed at Java programmers, but may not be incomprehensible to non-specialists.

The steps you need to follow are:

1. Write and compile a class containing one or more static public methods representing the function(s) required
2. Make this class available on the application's classpath at runtime as described in Section 3.1
3. Specify the class's name to the application, as the value of the `jel.classes` system property (colon-separated if there are several) as described in Section 3.3

Any public static methods defined in the classes thus specified will then be available for use. They should be defined to take and return the relevant primitive or Object types for the function required. For instance a class written as follows would define a three-value average:

```
public class AuxFuncs {
    public static double average3(double x, double y, double z) {
        return (x + y + z) / 3.0;
    }
}
```

and the command

```
stilts tpipe cmd='addcol AVERAGE "average3($1,$2,$3)"'
```

would add a new column named **AVERAGE** giving the average of the first three existing columns. Exactly how you would build this is dependent on your system, but it might involve doing something like the following:

1. Writing a file named `AuxFuncs.java` containing the above code
2. Compiling it using a command like `"javac AuxFuncs.java"`
3. Running `tpipe` using the flags `"stilts -classpath . -Djel.classes=AuxFuncs tpipe"`

Note that (in versions later than STILTS 3.0-6) variable-length argument lists are supported where the final declared argument is an array, so for instance a method declared like:

```
public static double sum(double[] values) {
    double total = 0;
    for (int i = 0; i < values.length; i++) {
        total += values[i];
    }
    return total;
}
```

can be invoked like `"sum(UMAG,GMAG,RMAG,IMAG,ZMAG)"`. The alternative form `"double... values"` can be used in the declaration with identical effect.

11 Programmatic Invocation

The STILTS package provides some capabilities, for instance plotting, that might be useful as part of other Java applications. The code that forms STILTS is fully documented at the API level; there are comprehensive javadocs throughout for the `uk.ac.starlink.ttools` package, its subpackages, and most of the other classes in the `uk.ac.starlink` tree on which it relies. Anybody is welcome to use these classes at their own risk, but the code does not form a stable API intended for public use: the javadocs are not distributed as part of the package (though you may be able to find them here), tutorial documentation is not provided, and there is no commitment to API stability between releases.

With this in mind, there are facilities for invoking the STILTS commands programmatically from third-party java code. Of course it is possible to do this by just calling the static `main(String[])` method of the application Main-Class (`Stilts`) but we document here how it can be done in a way which allows more control, using the `uk.ac.starlink.task` parameter handling framework.

Each of the STILTS tasks listed in Appendix B is represented by a class implementing the `Task` interface; these all have no-arg constructors. To run it, you need to create an instance of the class, pass it an `Environment` object which can acquire values for parameters by name, and then execute it. The `MapEnvironment` class, based on a `Map` containing name/value pairs, is provided for this purpose. As well as managing parameter values, `MapEnvironment` captures table and text output in a way that lets you retrieve it after the task has executed. Here is a simple example for invoking the `calc` task to perform a simple calcation:

```
MapEnvironment env = new MapEnvironment();
env.setValue( "expression", "sqrt(3*3+4*4)" );
Task calcTask = new uk.ac.starlink.ttools.task.Calc();
calcTask.createExecutable( env ).execute();
String result = env.getOutputText();
```

The execution corresponds exactly to the command-line:

```
stilts calc expression="sqrt(3*3+4*4)"
```

The Usage section for the `calc` task notes that the corresponding Task subclass is `Calc`.

Also in the usage section, each parameter reports the data type that it may take, and objects of this type may be used as the parameter value passed in the `MapEnvironment` as an alternative to passing string values. For the case of the input table parameters, this is `StarTable`, so in a task like `tpipe` (`TablePipe`), if you want to read a file "data.fits", you can either write

```
env.setValue( "in", "data.fits" );
```

or

```
StarTable table = new StarTableFactory().readStarTable( "data.fits" );
env.setValue( "in", table );
```

That doesn't buy you much, but the table could equally be obtained from any other source, including being a user-defined iterable over existing data structures. See SUN/252 for more information on `StarTable` handling.

For some short examples of programs which invoke STILTS tasks in this way, see the source code of some of the examples in the `uk.ac.starlink.ttools.example` directory: `Calculator` and `Head10`.

Some commands provide additional methods for use with parameter-based invocation. In particular the plotting commands can be used to create `JComponent` objects that can be incorporated into an existing GUI. A working example of this can be found in the source code for the example

EnvPlanePlotter class. For some more tutorial introductions to using the plotting classes programmatically, see also the example classes SinePlot, ApiPlanePlotter, and BasicPlotGui in the same place.

A Commands By Category

This section lists the commands available broken down by the category of function they provide. Some commands appear in more than one category. Detailed descriptions and examples for each command can be found in Appendix B.

Format conversion:

- `tcopy` (Appendix B.26): Converts between table formats
- `votcopy` (Appendix B.37): Transforms between VOTable encodings

See also Section 5.

Generic table manipulation:

- `tcopy` (Appendix B.26): Converts between table formats
- `tpipe` (Appendix B.35): Performs pipeline processing on a table
- `tmulti` (Appendix B.33): Writes multiple tables to a single container file
- `tmultin` (Appendix B.34): Writes multiple processed tables to single container file
- `tcat` (Appendix B.24): Concatenates multiple similar tables
- `tcatn` (Appendix B.25): Concatenates multiple tables
- `tloop` (Appendix B.28): Generates a single-column table from a loop variable
- `tjoin` (Appendix B.29): Joins multiple tables side-to-side
- `tcube` (Appendix B.27): Calculates N-dimensional histograms

See also Section 6.

Crossmatching:

- `tmatch1` (Appendix B.30): Performs a crossmatch internal to a single table
- `tmatch2` (Appendix B.31): Crossmatches 2 tables using flexible criteria
- `tmatchn` (Appendix B.32): Crossmatches multiple tables using flexible criteria
- `tskymatch2` (Appendix B.36): Crossmatches 2 tables on sky position
- `cdsskymatch` (Appendix B.2): Crossmatches table on sky position against VizieR/SIMBAD table
- `coneskymatch` (Appendix B.3): Crossmatches table on sky position against remote cone service
- `sqlskymatch` (Appendix B.18): Crossmatches table on sky position against SQL table

See also Section 7.

Plotting:

- `plot2plane` (Appendix B.7): Draws a plane plot
- `plot2sky` (Appendix B.8): Draws a sky plot
- `plot2cube` (Appendix B.9): Draws a cube plot
- `plot2sphere` (Appendix B.10): Draws a sphere plot
- `plot2time` (Appendix B.11): Draws a time plot
- `plot2d` (Appendix B.12) (*deprecated*): Old-style 2D Scatter Plot
- `plot3d` (Appendix B.13) (*deprecated*): Old-style 3D Scatter Plot
- `plothist` (Appendix B.14) (*deprecated*): Old-style Histogram

See also Section 9.

Sky Pixel Operations:

- `pixfoot` (Appendix B.5): Generates Multi-Order Coverage maps
- `pixsample` (Appendix B.6): Samples from a HEALPix pixel data file

VOTables:

- `votcopy` (Appendix B.37): Transforms between VOTable encodings
- `votlint` (Appendix B.38): Validates VOTable documents

Virtual Observatory service access:

- `cdsskymatch` (Appendix B.2): Crossmatches table on sky position against VizieR/SIMBAD table
- `coneskymatch` (Appendix B.3): Crossmatches table on sky position against remote cone service
- `tapskymatch` (Appendix B.23): Crossmatches table on sky position against TAP table
- `tapquery` (Appendix B.21): Queries a Table Access Protocol server
- `tapresume` (Appendix B.22): Resumes a previous query to a Table Access Protocol server
- `taplint` (Appendix B.20): Tests TAP services
- `regquery` (Appendix B.15): Queries the VO registry

SQL Database access:

- `sqlclient` (Appendix B.17): Executes SQL statements
- `sqlupdate` (Appendix B.19): Updates values in an SQL table
- `sqlskymatch` (Appendix B.18): Crossmatches table on sky position against SQL table

Miscellaneous:

- `server` (Appendix B.16): Runs an HTTP server to perform STILTS commands
- `calc` (Appendix B.1): Evaluates expressions
- `funcs` (Appendix B.4): Browses functions used by algebraic expression language

B Command Reference

This appendix provides the reference documentation for the commands in the package. For each one a description of its purpose, a list of its command-line arguments, and some examples are given.

B.1 `calc`: Evaluates expressions

`calc` is a very simple utility for evaluating expressions. It uses the same expression evaluator as is used in `tpipe` and the other generic table tasks for things like creating new columns, so it can be used as a quick test to see what expressions work, or in order to evaluate expressions using the various algebraic functions documented in Section 10.5. Since usually no table is involved, you can't refer to column names in the expressions. It has one mandatory parameter, the expression to evaluate, and writes the result to the screen.

B.1.1 Usage

The usage of `calc` is

```
stilts <stilts-flags> calc table=<table>
                             [expression=]<expr>
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.Calc`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

expression = <expr> (*String*)

An expression to evaluate. The functions in Section 10.5 can be used.

table = <table> (*StarTable*)

A table which provides the context within which `expression` is evaluated. This parameter is optional, and will usually not be required; its only purpose is to allow use of constant expressions (table parameters) associated with the table. These can be referenced using identifiers of the form `param$*`, `ucd$*` or `utype$*` - see Section 10.2 for more detail.

B.1.2 Examples

Here are some examples of using `calc`:

```
stilts calc 1+2
```

Calculates one plus two. Writes "3" to standard output.

```
stilts calc 'isoToMjd("2005-12-25T00:00:00")'
```

Works out the Modified Julian Day corresponding to Christmas 2005. The output is "53729.0".

```
stilts calc 'param$author' table=catalogue.xml
```

In this case the expression is evaluated in the context of the supplied table, which means that the table's parameters can be referenced in the expression. This example just outputs the value of the table parameter named "author".

B.2 cdsskymatch: Crossmatches table on sky position against VizieR/SIMBAD table

`cdsskymatch` uses the CDS X-Match service to join a local table to one of the tables hosted by the Centre de Données astronomiques de Strasbourg. This includes all of the VizieR tables and the SIMBAD database. The service is very fast, and in most cases it is the best way to match a local table against a large external table hosted by a service. It is almost certainly much better than using `coneskymatch`, though it is less flexible than TAP (see the `tapquery` task for flexible access to TAP services, and `tapskymatch` for positional matches).

The local table is uploaded to the X-Match service in chunks, and the matches for each chunk are retrieved in turn and eventually stitched together to form the final result. The tool only uploads sky position and an identifier for each row of the input table, but all columns of the input table are reinstated in the result for reference.

The remote table in most cases contains only a subset of the the columns in the relevant VizieR table, including the most useful ones. The service currently provides no straightforward way to acquire columns which are not returned by default.

Acknowledgement: CDS note that if the use of the X-Match service is useful to your research, they would appreciate the following acknowledgement:

"This research made use of the cross-match service provided by CDS, Strasbourg."

B.2.1 Usage

The usage of `cdsskymatch` is

```
stilts <stilts-flags> cdsskymatch ifmt=<in-format> istream=true|false
                                icmd=<cmds> ocmd=<cmds>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plast
                                out=<out-table> ofmt=<out-format>
                                ra=<expr> dec=<expr>
                                radius=<value/arcsec> cdstable=<value>
                                find=all|best|best-remote|each|each-dist
                                blocksize=<int-value> maxrec=<int-value>
                                compress=true|false
                                serviceurl=<url-value> usemoc=true|false
                                presort=true|false fixcols=none|dups|all
                                suffixin=<label> suffixremote=<label>
                                [in=<table>
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.CdsUploadSkyMatch`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

blocksize = <int-value> (Integer)

The CDS Xmatch service operates limits on the maximum number of rows that can be uploaded and the maximum number of rows that is returned as a result from a single query. In the case of large input tables, they are broken down into smaller blocks, and one request is sent to the external service for each block. This parameter controls the number of rows in each block. For an input table with fewer rows than this value, the whole thing is done as a single request.

At time of writing, the maximum upload size is 100Mb (about 3Mrow; this does not depend on

the width of your table), and the maximum return size is 2Mrow.

Large block sizes tend to be good (up to a point) for reducing the total amount of time a large xmatch operation takes, but they can make it harder to see the job progressing. There is also the danger (for ALL-type find modes) of exceeding the return size limit, which will result in truncation of the returned result.

[Default: 50000]

cdstable = <value> (*String*)

Identifier of the table from the CDS crossmatch service that is to be matched against the local table. This identifier may be the standard VizieR identifier (e.g. "II/246/out" for the 2MASS Point Source Catalogue) or "simbad" to indicate SIMBAD data.

See for instance the TAPVizieR table searching facility at <http://tapvizier.u-strasbg.fr/adql/> to find VizieR catalogue identifiers.

compress = true|false (*Boolean*)

If true, the service is requested to provide HTTP-level compression for the response stream (Accept-Encoding header is set to "gzip", see RFC 2616). This does not guarantee that compression will happen but if the service honours this request it may result in a smaller amount of network traffic at the expense of more processing on the server and client.

[Default: true]

dec = <expr> (*String*)

Declination in degrees in the ICRS coordinate system for the position of each row of the input table. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

find = all|best|best-remote|each|each-dist (*UserFindMode*)

Determines which pair matches are included in the result.

- all: All matches
- best: Matched rows, best remote row for each input row
- best-remote: Matched rows, best input row for each remote row
- each: One row per input row, contains best remote match or blank
- each-dist: One row per input row, column giving distance only for best match

Note only the all mode is symmetric between the two tables.

Note also that there is a bug in best-remote matching. If the match is done in multiple blocks, it's possible for a remote table row to appear matched against one local table row per uploaded block, rather than just once for the whole result. If you're worried about that, set `blocksize >= rowCount`. This may be fixed in a future release.

[Default: all]

fixcols = none|dups|all (*Fixer*)

Determines how input columns are renamed before use in the output table. The choices are:

- none: columns are not renamed
- dups: columns which would otherwise have duplicate names in the output will be renamed to indicate which table they came from
- all: all columns will be renamed to indicate which table they came from

If columns are renamed, the new ones are determined by `suffix*` parameters.

[Default: dups]

icmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter

commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = <in-format> (String)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istream = true|false (Boolean)

If set true, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

maxrec = <int-value> (Integer)

Limit to the number of rows resulting from this operation. If the value is negative (the default) no limit is imposed. Note however that there can be truncation of the result if the number of records returned from a single chunk exceeds the service hard limit (2,000,000 at time of writing).

[Default: `-1`]

ocmd = <cmds> (ProcessingStep[])

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters

and/or semicolons.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

**omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(*ProcessingMode*)**

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

presort = true|false (*Boolean*)

If true, the rows are sorted by HEALPix index before they are uploaded to the CDS X-Match service. If the match is done in multiple blocks, this may improve efficiency, since when matching against a large remote catalogue the X-Match service likes to process requests in which sources are grouped into a small region rather than scattered all over the sky.

Note this will have a couple of other side effects that may be undesirable: it will read all the input rows into the task at once, which may make it harder to assess progress, and it will affect the order of the rows in the output table.

It is *probably* only worth setting true for rather large (multi-million-row?) multi-block matches, where both local and remote catalogues are spread over a significant fraction of the sky. But feel free to experiment.

[Default: false]

ra = <expr> (*String*)

Right ascension in degrees in the ICRS coordinate system for the position of each row of the input table. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

radius = <value/arcsec> (*Double*)

Maximum distance from the local table (ra,dec) position at which counterparts from the remote table will be identified. This is a fixed value given in arcseconds, and must be in the range [0,180] (this limit is currently enforced by the CDS Xmatch service).

serviceurl = <url-value> (*URL*)

The URL at which the CDS Xmatch service can be found. Normally this should not be altered from the default, but if other implementations of the same service are known, this parameter can be used to access them.

[Default: <http://cdsxmatch.u-strasbg.fr/xmatch/api/v1/sync>]

suffixin = <label> (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from the input table.

[Default: `_in`]

suffixremote = <label> (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from the CDS result table.

[Default: `_cds`]

usemoc = true|false (*Boolean*)

If true, first acquire a MOC coverage map from CDS, and use that to pre-filter rows before uploading them for matching. This should improve efficiency, but have no effect on the result.

[Default: `true`]

B.2.2 Examples

Here are some examples of `cdsskymatch`:

```
stilts cdsskymatch cdstable=II/246/out find=all
              in=dr5qso.fits ra=RA dec=DEC radius=1 out=qso_2mass.fits
```

Matches a local catalogue `dr5qso.fits` against the VizieR table `II/246/out` (the 2MASS Point Source Catalogue). The search radius is 1 arcsecond, and all 2MASS sources within the radius of each input source are returned.

```
stilts cdsskymatch cdstable=simbad find=best
              in=sources.txt ifmt=ascii ra=RAJ2000 dec=DEJ2000 radius=8.5
              blocksize=1000 icmd=progress omode=topcat
```

This finds the closest object in the SIMBAD database within 8.5 arcsec for each row of an input ASCII table. Uploads are done in blocks of 1,000 rows at a time, and progress is displayed on the console. When the match is complete, the result is sent directly to a running instance of TOPCAT.

```
stilts cdsskymatch in=3XMM_DR4cat_slim_v1.0.fits
              icmd='select "SC_POSERR < 1 && SC_EXTENT == 0"'
              cdstable=B/mk/mktypes
```

```
ra=SC_RA dec=SC_DEC radius=1.5
find=best suffixin=_XMM suffixremote=_MK fixcols=all
ocmd='select startsWith(spType_MK,\"G\")'
out=xmm_gtype.fits
```

This locates XMM-Newton point-like sources identified as being of spectral type G. It uses the 3XMM-DR4 XMM-Newton serendipitous source catalogue as input. The `icmd` filter selects the objects in that catalogue with well-defined point-like positions. It then matches them with Skiff's MK spectral classification catalogue (B/mk/mktypes in Vizier) and finally filters the result to include only those sources identified as being of spectral type G. Thanks to Ada Nebot (CDS) for this example.

B.3 coneskmatch: Crossmatches table on sky position against remote cone service

Note: this command is very inefficient for large tables, and in most cases `cdsskymatch` or `tapskymatch` provide better alternatives.

`coneskmatch` is a utility which performs a cone search-like query to a remote server for each row of an input table. Each of these queries returns a table with one row for each item held by the server in the region of sky represented by the input row. The results of all the queries are then concatenated into one big output table which is the output of this command.

The type of virtual observatory service queried is determined by the `servicetype` parameter. Typically it will be a Cone Search service, which queries a remote catalogue for astronomical objects or sources in a particular region. However, you can also query Simple Image Access and Simple Spectral Access services in just the same way, to return tables of available image and spectral resources in the relevant regions.

The identity of the server to query is given by the `serviceurl` parameter. Some advice about how to locate URLs for suitable services is given in Appendix B.3.3.

The effect of this command is like doing a positional crossmatch where one of the catalogues is local and the other is remote and exposes its data via a cone search/SIA/SSA service. Because of both the network communication and the necessarily naive crossmatching algorithm (which scales linearly with the size of the local catalogue) however, it is only suitable if the local catalogue has a reasonably small number of rows, unless you are prepared to wait a long time.

The `parallel` parameter allows you to perform multiple cone searches concurrently, so that instead of completing the first cone search, then the second, then the third, the program can be executing a number of them at once. This can speed up operation considerably, especially in the face of network latency, but beware that submitting a very large number of queries simultaneously to the same server may overload it, resulting in some combination of failed queries, ultimately slower runtimes, and unpopularity with server admins. Best to start with a low parallelism and cautiously increase it to see whether there are gains in performance.

Note that when running, `coneskmatch` can generate a lot of WARNING messages. Most of these are complaining about badly formed VOTables being returned from the cone search services. STILTS does its best to work out what the service responses mean in this case, and usually makes a good enough job of it.

Note: this task was known as `multicone` in its experimental form in STILTS v1.2 and v1.3.

B.3.1 Usage

The usage of `coneskmatch` is

```

stilts <stilts-flags> coneskymatch ifmt=<in-format> istream=true|false
                                icmd=<cmds> ocmd=<cmds>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plast
                                out=<out-table> ofmt=<out-format>
                                ra=<expr> dec=<expr> sr=<expr/deg>
                                find=best|all|each usefoot=true|false
                                footsize=<int-value>
                                copycols=<colid-list>
                                scorecol=<col-name> parallel=<n>
                                erract=abort|ignore|retry|retry<n>
                                ostream=true|false fixcols=none|dups|all
                                suffix0=<label> suffix1=<label>
                                servicetype=cone|sia|ssa
                                serviceurl=<url-value> verb=1|2|3
                                dataformat=<value> emptyok=true|false
                                compress=true|false
                                [in=]<table>

```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.MultiCone`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

compress = true|false (*Boolean*)

If true, the service is requested to provide HTTP-level compression for the response stream (Accept-Encoding header is set to "gzip", see RFC 2616). This does not guarantee that compression will happen but if the service honours this request it may result in a smaller amount of network traffic at the expense of more processing on the server and client.

[Default: true]

copycols = <colid-list> (*String*)

List of columns from the input table which are to be copied to the output table. Each column identified here will be prepended to the columns of the combined output table, and its value for each row taken from the input table row which provided the parameters of the query which produced it. See Section 6.3 for list syntax. The default setting is "*", which means that all columns from the input table are included in the output.

[Default: *]

dataformat = <value> (*String*)

Indicates the format of data objects described in the returned table. The meaning of this is dependent on the value of the `servicetype` parameter:

- `servicetype=cone`: not used
- `servicetype=sia`: gives the MIME type of images referenced in the output table, also special values "GRAPHIC" and "ALL". (value of the SIA FORMAT parameter)
- `servicetype=ssa`: gives the MIME type of spectra referenced in the output table, also special values "votable", "fits", "compliant", "graphic", "all", and others (value of the SSA FORMAT parameter).

dec = <expr> (*String*)

Declination in degrees in the ICRS coordinate system for the position of each row of the input table. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

emptyok = true|false (*Boolean*)

Whether the table metadata which is returned from a search result with zero rows is to be believed. According to the spirit, though not the letter, of the cone search standard, a cone

search service which returns no data ought nevertheless to return the correct column headings. Unfortunately this is not always the case. If this parameter is set `true`, it is assumed that the service behaves properly in this respect; if it does not an error may result. In that case, set this parameter `false`. A consequence of setting it `false` is that in the event of no results being returned, the task will return no table at all, rather than an empty one.

[Default: `true`]

erract = abort|ignore|retry|retry<n> (ConeErrorPolicy)

Determines what will happen if any of the individual cone search requests fails. By default the task aborts. That may be the best thing to do, but for unreliable or poorly implemented services you may find that some searches fail and others succeed so it can be best to continue operation in the face of a few failures. The options are:

- `abort`: failure of any query terminates the task
- `ignore`: failure of a query is treated the same as a query which returns no rows
- `retry`: failed queries are retried until they succeed; use with care - if the failure is for some good, or at least reproducible reason this could prevent the task from ever completing
- `retry<n>`: failed queries are retried at most a fixed number `<n>` of times. If they still fail the task terminates.

[Default: `abort`]

find = best|all|each (String)

Determines which matches are retained.

- `best`: Only the matching query table row closest to the input table row will be output. Input table rows with no matches will be omitted. (Note this corresponds to the `best1` option in the pair matching commands, and `best1` is a permitted alias).
- `all`: All query table rows which match the input table row will be output. Input table rows with no matches will be omitted.
- `each`: There will be one output table row for each input table row. If matches are found, the closest one from the query table will be output, and in the case of no matches, the query table columns will be blank.

[Default: `all`]

fixcols = none|dups|all (Fixer)

Determines how input columns are renamed before use in the output table. The choices are:

- `none`: columns are not renamed
- `dups`: columns which would otherwise have duplicate names in the output will be renamed to indicate which table they came from
- `all`: all columns will be renamed to indicate which table they came from

If columns are renamed, the new ones are determined by `suffix*` parameters.

[Default: `dups`]

footnside = <int-value> (Integer)

Determines the HEALPix Nside parameter for use with the MOC footprint service. This tuning parameter determines the resolution of the footprint if available. Larger values give better resolution, hence a better chance of avoiding unnecessary queries, but processing them takes longer and retrieving and storing them is more expensive.

The value must be a power of 2, and at the time of writing, the MOC service will not supply footprints at resolutions greater than `nside=512`, so it should be `<=512`.

Only used if `usefoot=true`.

icmd = <cmds> (ProcessingStep[])

Specifies processing to be performed on the input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = <in-format> (String)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istream = true|false (Boolean)

If set true, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

ocmd = <cmds> (ProcessingStep[])

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (String)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special

value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

`omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui`

(ProcessingMode)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (`<mode-args>`) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

`ostream = true|false` *(Boolean)*

If set `true`, this will cause the operation to stream on output, so that the output table is built up as the results are obtained from the cone search service. The disadvantage of this is that some output modes and formats need multiple passes through the data to work, so depending on the output destination, the operation may fail if this is set. Use with care (or be prepared for the operation to fail).

[Default: false]

`out = <out-table>` *(TableConsumer)*

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

`parallel = <n>` *(Integer)*

Allows multiple cone searches to be performed concurrently. If set to the default value, 1, the cone query corresponding to the first row of the input table will be dispatched, when that is completed the query corresponding to the second row will be dispatched, and so on. If set to `<n>`, then queries will be overlapped in such a way that up to approximately `<n>` may be running at any one time.

Whether increasing `<n>` is a good idea, and what might be a sensible maximum value, depends on the characteristics of the service being queried. In particular, setting it to too large a number may overload the service resulting in some combination of failed queries, ultimately slower runtimes, and unpopularity with server admins.

The maximum value permitted for this parameter by default is 10. This limit may be raised by use of the `service.maxparallel` system property but use that option with great care since you may overload services and make yourself unpopular with data centre admins. As a rule, you should only increase this value if you have obtained permission from the data centres whose services on which you will be using the increased parallelism.

[Default: 1]

ra = <expr> (*String*)

Right ascension in degrees in the ICRS coordinate system for the position of each row of the input table. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

scorecol = <col-name> (*String*)

Gives the name of a column in the output table to contain the distance between the requested central position and the actual position of the returned row. The distance returned is an angular distance in degrees. If a null value is chosen, no distance column will appear in the output table.

[Default: Separation]

servicetype = cone|sia|ssa (*ServiceType*)

Selects the type of data access service to contact. Most commonly this will be the Cone Search service itself, but there are one or two other possibilities:

- **cone**: Cone Search protocol - returns a table of objects found near each location. See Cone Search standard.
- **sia**: Simple Image Access protocol - returns a table of images near each location. See SIA standard.
- **ssa**: Simple Spectral Access protocol - returns a table of spectra near each location. See SSA standard.

[Default: cone]

serviceurl = <url-value> (*URL*)

The base part of a URL which defines the queries to be made. Additional parameters will be appended to this using CGI syntax ("name=value", separated by '&' characters). If this value does not end in either a '?' or a '&', one will be added as appropriate.

See Appendix B.3.3 for discussion of how to locate service URLs corresponding to given datasets.

sr = <expr/deg> (*String*)

Expression which evaluates to the search radius in degrees for the request at each row of the input table. This will often be a constant numerical value, but may be the name or ID of a column in the input table, or a function involving one.

suffix0 = <label> (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from the input table.

[Default: _0]

suffix1 = <label> (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from the cone result table.

[Default: _1]

usefoot = true|false (*Boolean*)

Determines whether an attempt will be made to restrict searches in accordance with available footprint information. If this is set true, then before any of the per-row queries are performed, an attempt may be made to acquire footprint information about the service. If such information can be obtained, then queries which fall outside the footprint, and hence which are known to yield no results, are skipped. This can speed up the search considerably.

Currently, the only footprints available are those provided by the CDS MOC (Multi-Order Coverage map) service, which covers VizieR and a few other cone search services.

[Default: true]

verb = 1|2|3 (*String*)

Verbosity level of the tables returned by the query service. A value of 1 indicates the bare minimum and 3 indicates all available information.

B.3.2 Examples

Here are some examples of `coneskysearch`:

```
stilts coneskysearch serviceurl=http://archive.stsci.edu/hst/search.php \
in=messier.xml sr=0.05 out=matches.xml
```

This queries the HST cone search service from Space Telescope for records within .05 degrees of each Messier object contained in a local VOTable `messier.xml`. The sky positions in the input catalogue are guessed from the available table metadata. The result is written to a new VOTable, `matches.xml`. Since the `servicetype` parameter is not given, the default (cone search) service type is assumed.

```
stilts coneskysearch
servicetype=sia \
serviceurl=http://irsa.ipac.caltech.edu/cgi-bin/2MASS/IM/nph-im_sia?type=ql&ds=asky \
in=messier.xml ra=RA dec=DEC \
dataformat=image/fits \
out=fitsimages.xml
```

This is similar to the previous example, but instead of querying an HST cone search server for catalogue objects near the input table positions, it queries a 2MASS Simple Image Access (SIA) server for images. It also explicitly names the columns holding the J2000 positions of each record in the input catalogue as `RA` and `DEC`. The search radius parameter (`sr`) is not set here; for SIA queries the default search radius is zero, which has the special meaning of including any image which covers the requested position. Setting `dataformat=image/fits` (which is the default) requests only records describing FITS-format images to be returned; setting it to an empty value might return other formats such as JPEG too.

```
stilts coneskysearch \
serviceurl='http://www.nofs.navy.mil/cgi-bin/vo_cone.cgi?CAT=NOMAD' \
in=vizier.xml#7 \
icmd='addskycoords -inunit sex fk4 fk5 RAB1950 DEB1950 RAJ2000 DEJ2000' \
icmd='progress'
ra=RAJ2000 dec=DEJ2000 sr=0.01 \
ocmd='replacecol -units deg RA hmsToDegrees(RA[0],RA[1],RA[2])' \
ocmd='replacecol -units deg DEC dmsToDegrees(DEC[0],DEC[1],DEC[2])' \
omode=topcat
```

In this example some pre-processing of the input catalogue and post-processing of the output catalogue is performed as well as the multiple cone search itself.

The input catalogue, which is the 8th TABLE element in a VOTable file, contains sky positions in sexagesimal FK4 (B1950) coordinates. The `icmd=addskycoords...` parameter specifies a filter which will add new columns in FK5 (J2000) degrees, which are what the `coneskysearch` command requires. The `icmd=progress` parameter specifies a filter which will

write progress information to the terminal so you can see how the queries are progressing.

The NOMAD service specified by the `serviceurl` parameter used here happens to return results with the RA/DEC columns represented in a rather eccentric format, namely 3-element floating point arrays representing (hours,minutes,seconds)/(degrees,minutes,seconds). The two `ocmd=replacecol...` filters replace the values of these columns with the scalar equivalents in degrees. Finally, the `omode=topcat` parameter causes the result table to be loaded directly into TOPCAT (if it is available).

```
stilts coneskymatch serviceurl='http://archive.stsci.edu/iue/search.php?' \
                    in=queries.txt ifmt=ascii \
                    ra='$1' dec='$2' \
                    sr='$3' copycols='$4' \
                    out=found.fits
```

Here the input is a plain text table with four unnamed columns, giving in order the right ascension, declination, positional error and name of target objects. The command carries out a cone search to the named service for each one. Note in this case the search radius (`sr` parameter) is taken from the table and so varies for each query. The `copycols` parameter has the value '\$4', which means that the value of the fourth column of the input table will be prepended to each row of the output table for which it is responsible. Output is to a FITS table.

B.3.3 Locating Cone Query Service URLs

To use the `coneskymatch` command you need the **service URL** (also known as the **base URL** or **access URL**) of a cone search, SIA or SSA service to use. If you know one of these representing a service that you wish to use, you can use it directly.

If you don't, you will need to find the URL from somewhere. It is the job of the Virtual Observatory **Registry** to keep a record of where you can find various astronomical services, so this is where you should look.

There are various ways you can interrogate the registry; the easiest is probably to use a graphical registry search tool. One such tool is AstroGrid's **VOExplorer**, which allows you to perform sophisticated searches for cone search, SIA or SSA services. Another option is to use TOPCAT; the Cone Search, SIA and SSA load dialogues allow you to search the registry for these services prior to performing a query; you can just use the registry part and cut'n'paste the URL which is shown.

Other registry querying tools are available, including STILTS's `regquery` (Appendix B.15) command. See that section of the manual for details, but for instance to locate registered Cone Search services which have something to do with SDSS data, you could execute the following:

```
stilts regquery query="capability/@standardID = 'ivo://ivoa.net/std/ConeSearch' and title
ocmd="keepcols 'shortName AccessUrl'" \
ofmt=ascii
```

Writing just `query="capability/@standardID = 'ivo://ivoa.net/std/ConeSearch'"` with no further qualification would give you *all* registered cone search services.

B.4 funcs: Browses functions used by algebraic expression language

`funcs` is a utility which allows you to browse the functions you can use in STILTS's algebraic expression language. Invoking the command causes a window to pop up on the display with two parts. The left hand panel contains a tree-like representation of the functions available - the top level shows the classes (categories) into which the functions are divided, and if you open these up (by double clicking on them) each contains a list of functions and constants in that class. If you click on any of these classes or their constituent functions or constants, a full description of what they are

and how to use them will appear in the right hand panel.

The information available from this command is the same as that given in Section 10.5, but the graphical browser may be a more convenient way to view the documentation. There are no parameters.

B.4.1 Usage

The usage of `funcs` is

```
stilts <stilts-flags> funcs
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.ShowFunctions`.

This task has no parameters.

B.5 `pixfoot`: Generates Multi-Order Coverage maps

`pixfoot` takes a list of sky positions from an input table and generates a pixel map describing a sky region which includes them all. Currently the output is to a format known as a Multi-Order Coverage map (MOC), which is a HEALPix-based format composed of a list of HEALPix pixels of different sizes, which can efficiently describe complex regions. Other output formats may be introduced in the future.

See also the Coverage class for MOC-related functions.

B.5.1 Usage

The usage of `pixfoot` is

```
stilts <stilts-flags> pixfoot ifmt=<in-format> istream=true|false
                             icmd=<cmds> order=<int-value> ra=<expr>
                             dec=<expr> radius=<expr> mocfmt=fits|ascii
                             out=<out-file>
                             [in=<table>]
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.PixFootprint`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

dec = <expr> (*String*)

Declination in degrees for the position of each row of the input table. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDS, column names and unit annotations what expression to use.

icmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this

way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = <in-format> (String)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istream = true|false (Boolean)

If set true, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

mocfmt = fits|ascii (MocFormat)

Determines the output format for the MOC file.

[Default: `fits`]

order = <int-value> (Integer)

Maximum HEALPix order for the MOC. This defines the maximum resolution of the output coverage map. The angular resolution corresponding to order k is approximately $180/\sqrt{3\pi}/2^k$ (3520*2^{-k} arcmin).

[Default: 13]

out = <out-file> (uk.ac.starlink.util.Destination)

The location of the output file. This is usually a filename to write to. If it is equal to the special value "-" the output will be written to standard output.

[Default: -]

ra = <expr> (String)

Right ascension in degrees for the position of each row of the input table. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

radius = <expr> (*String*)

Expression which evaluates to the radius in degrees of the cone at each row of the input table. The default is "0", which treats each position as a point rather than a cone, but a constant or an expression as described in Section 10 may be used instead.

[Default: 0]

B.5.2 Examples

Here are some examples of `pixfoot`:

```
stilts pixfoot in=survey.vot order=8 mocfmt=fits out=sfoot.fits
```

Generates an order-8 FITS MOC file from the point positions of rows in the given VOTable. The columns representing sky position are determined automatically (if possible) by examining the metadata in the input table.

```
stilts pixfoot in='jdbc:mysql://localhost/astro1#SELECT * FROM first1'
icmd='addskycoords galactic icrs GLON GLAT ALPHA DELTA'
ra=ALPHA dec=DELTA radius=20./3600.
order=13 mocfmt=fits out=first.moc
```

Generates an order-13 FITS MOC file from positions in a table held in a database. The positions in the original table are in galactic coordinates, so have to be converted to equatorial (ICRS) first. The map is formed in this case by surrounding each point by a disc of 20 arcsec. Note that JDBC database access will have to be set up as per Section 3.4 for this command to work.

B.6 `pixsample`: Samples from a HEALPix pixel data file

`pixsample` samples data at the sky position represented by each row from an all-sky map contained in a HEALPix-format pixel data file. Such files are actually tables (usually in FITS format) in which the row number corresponds to a HEALPix pixel index, and the pixel values are cell contents; one or more columns may be present containing values for one or more all-sky maps. The result of this command is to add a column to the input table representing the pixel data at the position of each input row for each of the data columns in the HEALPix table.

This command does not attempt to convert between coordinate systems except as instructed, so it is important to know what coordinate system the HEALPix file is in, and ensure that the coordinates supplied from the input table match this. You may need to examine the documentation or headers of the HEALPix file in question to find out. See the Examples section for some examples.

There is a choice of how the sampling is done; the simplest way is just to use the value of the pixel covering the indicated position. An alternative is to average over a disc of given radius (perhaps a function of the input row). Other options (e.g. max/min) could easily be added.

Although HEALPix is not a common format for storing image data in general, it is used for storing a number of important all-sky data sets such as the WMAP results and Schlegel dust maps. The NASA LAMBDA (<http://lambda.gsfc.nasa.gov/>) (Legacy Archive for Microwave Background Data Analysis) archive has a number of maps in a suitable format, including foreground data like predicted reddening as well as CMB maps.

B.6.1 Usage

The usage of `pixsample` is

```

stilts <stilts-flags> pixsample in=<table> ifmt=<in-format> icmd=<cmds>
                                pixdata=<pix-table> pfmt=<in-format>
                                pcmd=<cmds> ocmd=<cmds>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic
                                out=<out-table> ofmt=<out-format>
                                pixorder=nested|ring|(auto) stat=point|mean
                                lon=<expr> lat=<expr>
                                insys=icrs|fk5|fk4|galactic|supergalactic|ecliptic
                                pixsys=icrs|fk5|fk4|galactic|supergalactic|ecliptic
                                radius=<expr>

```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.PixSample`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

icmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (`;`). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character `@`. Thus a value of `"@filename"` causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

insys = icrs|fk5|fk4|galactic|supergalactic|ecliptic (*SkySystem*)

Specifies the sky coordinate system in which sample positions are provided by the `lon/lat` parameters. If the sample positions are given in the same coordinate system as that given by the pixel data table, both the `insys` and `pixsys` parameters may be set `null`.

The available coordinate systems are:

- `icrs`: ICRS (Hipparcos) (Right Ascension, Declination)

- `fk5`: FK5 J2000.0 (Right Ascension, Declination)
- `fk4`: FK4 B1950.0 (Right Ascension, Declination)
- `galactic`: IAU 1958 Galactic (Longitude, Latitude)
- `supergalactic`: de Vaucouleurs Supergalactic (Longitude, Latitude)
- `ecliptic`: Ecliptic (Longitude, Latitude)

lat = `<expr>` (*String*)

Expression which evaluates to the latitude coordinate in degrees in the input table at which positions are to be sampled from the pixel data table. This will usually be the name or ID of a column in the input table, or an expression involving one. If this coordinate does not match the coordinate system used by the pixel data table, both coordinate systems must be set using the `insys` and `pixsys` parameters.

lon = `<expr>` (*String*)

Expression which evaluates to the longitude coordinate in degrees in the input table at which positions are to be sampled from the pixel data table. This will usually be the name or ID of a column in the input table, or an expression involving one. If this coordinate does not match the coordinate system used by the pixel data table, both coordinate systems must be set using the `insys` and `pixsys` parameters.

ocmd = `<cmds>` (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = `<out-format>` (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

omode = `out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui`
(*ProcessingMode*)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (`<mode-args>`) are required to determine the exact behaviour.

Possible values are

- `out`
- `meta`
- `stats`
- `count`
- `cgi`

- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

pcmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on pixel data table as specified by parameter `pixdata`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

pfmt = <in-format> (*String*)

File format for the HEALPix pixel data table. This is usually, but not necessarily, FITS.

[Default: fits]

pixdata = <pix-table> (*StarTable*)

The location of the table containing the pixel data. The data must be in the form of a HEALPix table, with one pixel per row in HEALPix order. These files are typically, but not necessarily, FITS tables. A filename or URL may be used, but a local file will be more efficient.

Some HEALPix format FITS tables seem to have rows which contain 1024-element arrays of pixels instead of single pixel values. This (rather perverse?) format is not currently supported here, but if there is demand support could be added.

pixorder = nested|ring|(auto) (*HealpixScheme*)

Selects the pixel ordering scheme used by the pixel data file. There are two different ways of ordering pixels in a HEALPix file, "ring" and "nested", and the sampler needs to know which one is in use. If you know which is in use, choose the appropriate value for this parameter; if (auto) is used it will attempt to work it out from headers in the file (the ORDERING header). If no reliable ordering scheme can be determined, the command will fail with an error.

[Default: (auto)]

pixsys = icrs|fk5|fk4|galactic|supergalactic|ecliptic (*SkySystem*)

Specifies the sky coordinate system used for the HEALPix data in the `pixdata` file. If the sample positions are given in the same coordinate system as that given by the pixel data table, both the `insys` and `pixsys` parameters may be set null.

The available coordinate systems are:

- icrs: ICRS (Hipparcos) (Right Ascension, Declination)
- fk5: FK5 J2000.0 (Right Ascension, Declination)

- `fk4`: FK4 B1950.0 (Right Ascension, Declination)
- `galactic`: IAU 1958 Galactic (Longitude, Latitude)
- `supergalactic`: de Vaucouleurs Supergalactic (Longitude, Latitude)
- `ecliptic`: Ecliptic (Longitude, Latitude)

radius = <expr> (*String*)

Determines the radius in degrees over which pixels will be sampled to generate the output statistic in accordance with the value of the `stat` parameter. This will typically be a constant value, but it may be an algebraic expression based on columns from the input table.

Not used if `stat=point`.

stat = `point|mean` (*StatMode*)

Determines how the pixel values will be sampled to generate an output value. The options are:

- `point`: Uses the value at the pixel covering the supplied position. In this case the `radius` parameter is not used.
- `mean`: Averages the values over all the pixels within a radius given by the `radius` parameter. This averaging is somewhat approximate; all pixels which are mostly within the radius are averaged with equal weights.

[Default: `point`]

B.6.2 Examples

Here are some examples of `pixsample`:

```
stilts pixsample in=szdata.fits pixdata=wmap_ilc_7yr_v4.fits
               lat=GAL_LAT lon=GAL_LON pcmd='keepcols TEMPERATURE'
               out=szdata_cmb.fits
```

Samples from a HEALPix file containing WMAP data are added to an input file `szdata.fits`, giving an output file `szdata_cmb.fits` which is the same but with an additional column `TEMPERATURE`. The sampling is done using the default statistical mode `point`, which just takes a point sample at the input position. The HEALPix file must have its pixels ordered using galactic coordinates, since that is the coordinate system available from the input table.

The `pixdata` file used here can be found (at time of writing) at http://lambda.gsfc.nasa.gov/data/map/dr4/dfp/ilc/wmap_ilc_7yr_v4.fits (24 Mbyte).

```
stilts pixsample in=messier.xml pixdata=lambda_sfd_ebv.fits
               stat=mean radius=5./60.
               insys=icrs pixsys=galactic lon=RA2000 lat=DEC2000
```

Samples data from a HEALPix table, averaging over a sampling radius of 5 arcmin. The coordinates in the input table are only available as ICRS (RA,Dec) coordinates, and the arrangement of the HEALPix pixels in the pixel data file uses galactic coordinates (you can only determine this by looking at the FITS headers or documentation of that file), so it is necessary to use the `insys` and `pixsys` parameters for conversion.

The `pixdata` file used here can be found (at time of writing) at http://lambda.gsfc.nasa.gov/data/foregrounds/SFD/lambda_sfd_ebv.fits (25 Mbyte).

B.7 `plot2plane`: Draws a plane plot

`plot2plane` draws plots on a Cartesian 2-dimensional surface.

Positional coordinates are specified as *x, y* pairs, e.g.:

```
plot2plane layer1=mark in1=cat.fits x1=RMAG y1=RMAG-BMAG
```

Content is added to the plot by specifying one or more *plot layers* using the *layerN* parameter. The *N* part is a suffix applied to all the parameters affecting a given layer; any suffix (including the empty string) may be used. Available layers for this plot type are: *mark* (Section 8.3.1), *size* (Section 8.3.2), *sizexy* (Section 8.3.3), *xyvector* (Section 8.3.4), *xyerror* (Section 8.3.5), *xyellipse* (Section 8.3.6), *link2* (Section 8.3.7), *mark2* (Section 8.3.8), *line* (Section 8.3.9), *linearfit* (Section 8.3.10), *label* (Section 8.3.11), *contour* (Section 8.3.12), *density* (Section 8.3.13), *fill* (Section 8.3.14), *histogram* (Section 8.3.15), *kde* (Section 8.3.16), *knn* (Section 8.3.17), *densogram* (Section 8.3.18), *function* (Section 8.3.19).

B.7.1 Usage

The usage of *plot2plane* is

```
stilts <stilts-flags> plot2plane xpix=<int-value> ypix=<int-value>
                                insets=<top>,<left>,<bottom>,<right>
                                omode=swing|out|cgi|discard|auto
                                storage=simple|cache|basic-cache
                                seq=<suffix>[,...] legend=true|false
                                legborder=true|false legopaque=true|false
                                legseq=<suffix>[,...] legpos=<xfrac,yfrac>
                                title=<value>
                                auxmap=inferno|magma|plasma|...
                                auxclip=<lo>,<hi> auxflip=true|false
                                auxquant=<number>
                                auxfunc=log|linear|sqrt|square
                                auxmin=<number> auxmax=<number>
                                auxlabel=<text> auxcrowd=<factor>
                                auxvisible=true|false
                                forcebitmap=true|false compositor=0..1
                                animate=<table> afmt=<in-format>
                                astream=true|false acmd=<cmds>
                                parallel=<int-value> xlog=true|false
                                ylog=true|false xflip=true|false
                                yflip=true|false xlabel=<text>
                                ylabel=<text> aspect=<number>
                                grid=true|false xcrowd=<number>
                                ycrowd=<number> minor=true|false
                                gridcolor=<rrggb>|red|blue|...
                                labelcolor=<rrggb>|red|blue|...
                                texttype=plain|antialias|latex
                                fontsize=<int-value>
                                fontstyle=standard|serif|mono
                                fontweight=plain|bold|italic|bold_italic
                                xmin=<number> xmax=<number> xsub=<lo>,<hi>
                                ymin=<number> ymax=<number> ysub=<lo>,<hi>
                                navaxes=xy|x|y xanchor=true|false
                                yanchor=true|false zoomfactor=<number>
                                leglabelN=<text>
                                layerN=<layer-type> <layerN-specific-params>
```

If you don't have the *stilts* script installed, write "java -jar stilts.jar" instead of "stilts" - see Section 3. The available *<stilts-flags>* are listed in Section 2.1. For programmatic invocation, the *Task* class for this command is `uk.ac.starlink.ttools.plot2.task.PlanePlot2Task`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

acmd = <cmds> (*ProcessingStep*[])

Specifies processing to be performed on the animation control table as specified by parameter

`animate`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

afmt = <in-format> (*String*)

Specifies the format of the animation control table as specified by parameter `animate`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

animate = <table> (*StarTable*)

If not null, this parameter causes the command to create a sequence of plots instead of just one. The parameter value is a table with one row for each frame to be produced. Columns in the table are interpreted as parameters which may take different values for each frame; the column name is the parameter name, and the value for a given frame is its value from that row. Animating like this is considerably more efficient than invoking the `STILTS` command in a loop.

The location of the animation control table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `afmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

aspect = <number> (*Double*)

Ratio of the unit length on the X axis to the unit length on the Y axis. If set to 1, the space will be isotropic. If not set (the default) the ratio will be determined by the given or calculated data bounds on both axes and the shape of the plotting region.

astream = true|false (*Boolean*)

If set true, the animation control table specified by the `animate` parameter will be read as a stream. It is necessary to give the `afmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as `VOTable`).

[Default: `false`]

auxclip = <lo>,<hi> (*Subrange*)

Defines a subrange of the colour ramp to be used for Aux shading. The is specified as a (low,high) comma-separated pair of two numbers between 0 and 1.

If the full range 0,1 (the default) is used, the whole range of colours specified by the selected shader will be used. But if, for instance a value of 0,0.5 is given, only those colours at the left

hand end of the ramp will be seen.

[Default: 0,1]

auxcrowd = <factor> (*Double*)

Determines how closely the tick marks are spaced on the Aux axis, if visible. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

[Default: 1.0]

auxflip = true|false (*Boolean*)

If true, the colour map on the Aux axis will be reversed.

[Default: false]

auxfunc = log|linear|sqrt|square (*Scaling*)

Defines the way that values in the Aux range are mapped to the selected colour ramp.

The available options are:

- log: Logarithmic scaling
- linear: Linear scaling
- sqrt: Square root scaling
- square: Square scaling

[Default: linear]

auxlabel = <text> (*String*)

Sets the label used to annotate the aux axis, if it is visible.

auxmap = inferno|magma|plasma|... (*Shader*)

Color map used for Aux axis shading.

A mixed bag of colour ramps are available: inferno, magma, plasma, viridis, cubehelix, sron, rainbow, rainbow2, rainbow3, pastel, accent, gnuplot, gnuplot2, speckby, set1, paired, hotcold, rdbu, piyg, brbg, cyan-magenta, red-blue, brg, heat, cold, light, greyscale, colour, standard, bugn, bupu, orrd, pubu, purd, huecl, hue, intensity, rgb_red, rgb_green, rgb_blue, hsv_h, hsv_s, hsv_v, yuv_y, yuv_u, yuv_v, scale_hsv_s, scale_hsv_v, scale_yuv_y, mask, blacker, whiter, transparency. *Note:* many of these, including rainbow-like ones, are frowned upon by the visualisation community.

[Default: inferno]

auxmax = <number> (*Double*)

Maximum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

auxmin = <number> (*Double*)

Minimum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

auxquant = <number> (*Double*)

Allows the colour map used for the Aux axis to be quantised. If an integer value N is chosen then the colour map will be viewed as N discrete evenly-spaced levels, so that only N different colours will appear in the plot. This can be used to generate a contour-like effect, and may make it easier to trace the boundaries of regions of interest by eye.

If left blank, the colour map is nominally continuous (though in practice it may be quantised to a medium-sized number like 256).

auxvisible = true|false (*Boolean*)

Determines whether the aux axis colour ramp is displayed alongside the plot.

If not supplied (the default), the aux axis will be visible when aux shading is used in any of the plotted layers.

compositor = 0..1 (*Compositor*)

Defines how multiple overplotted partially transparent pixels are combined to form a resulting colour. The way this is used depends on the details of the specified plot.

Currently, this parameter takes a "boost" value in the range 0..1. If the value is zero, saturation semantics are used: RGB colours are added in proportion to their associated alpha value until the total alpha is saturated (reaches 1), after which additional pixels have no further effect. For larger boost values, the effect is similar, but any non-zero alpha in the output is boosted to the given minimum value. The effect of this is that even very slightly populated pixels can be visually distinguished from unpopulated ones which may not be the case for saturation composition.

[Default: 0.05]

fontsize = <int-value> (*Integer*)

Size of the text font in points.

[Default: 12]

fontstyle = standard|serif|mono (*FontType*)

Font style for text.

The available options are:

- standard
- serif
- mono

[Default: standard]

fontweight = plain|bold|italic|bold_italic (*FontWeight*)

Font weight for text.

The available options are:

- plain
- bold
- italic
- bold_italic

[Default: plain]

forcebitmap = true|false (*Boolean*)

This option only has an effect when writing output to vector graphics formats (PDF and PostScript). If set `true`, the data contents of the plot are drawn as a pixel map embedded into the output file rather than plotting each point in the output. This may make the output less beautiful (round markers will no longer be perfectly round), but it may result in a much smaller file if there are very many data points. Plot annotations such as axis labels will not be affected - they are still drawn as vector text. Note that in some cases (e.g. `shadingN=auto` or `shadingN=density`) this kind of pixellisation will happen in any case.

[Default: false]

grid = true|false (*Boolean*)

If `true`, grid lines are drawn on the plot at positions determined by the major tick marks. If `false`, they are absent.

[Default: false]

gridcolor = <rrggbb>|red|blue|... (*Color*)

The color of the plot grid.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: light_grey]

insets = <top>,<left>,<bottom>,<right> (*Insets*)

Defines the amount of space in pixels around the actual plotting area. This space is used for axis labels, and other decorations and any left over forms an empty border.

The size and position of the actual plotting area is determined by this parameter along with `xpix` and `ypix`. If no value is set (the default), the insets will be determined automatically according to how much space is required for labels etc.

labelcolor = <rrggbb>|red|blue|... (*Color*)

The color of axis labels and other plot annotations.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: black]

layerN = <layer-type> <layerN-specific-params> (*LayerType*)

Selects one of the available plot types for layerN. A plot consists of a plotting surface, set up using the various unsuffixed parameters of the plotting command, and zero or more plot layers. Each layer is introduced by a parameter with the name `layer<N>` where the suffix "<N>" is a label identifying the layer and is appended to all the parameter names which configure that layer. Suffixes may be any string, including the empty string.

This parameter may take one of the following values, described in more detail in Section 8.3:

- mark (Section 8.3.1)
- size (Section 8.3.2)
- sizexy (Section 8.3.3)
- xyvector (Section 8.3.4)
- xyerror (Section 8.3.5)
- xyellipse (Section 8.3.6)
- link2 (Section 8.3.7)
- mark2 (Section 8.3.8)
- line (Section 8.3.9)
- linearfit (Section 8.3.10)
- label (Section 8.3.11)
- contour (Section 8.3.12)
- density (Section 8.3.13)
- fill (Section 8.3.14)
- histogram (Section 8.3.15)
- kde (Section 8.3.16)
- knn (Section 8.3.17)
- densogram (Section 8.3.18)
- function (Section 8.3.19)

Each of these layer types comes with a list of type-specific parameters to define the details of that layer, including some or all of the following groups:

- input table parameters (e.g. `inN`, `icmdN`)
- coordinate params referring to input table columns (e.g. `xN`, `yN`)
- layer style parameters (e.g. `shadingN`, `colorN`)

Every parameter notionally carries the same suffix *N*. However, if the suffix is not present, the application will try looking for a parameter with the same name with no suffix instead. In this way, if several layers have the same value for a given parameter (for instance input table), you can supply it using one unsuffixed parameter to save having to supply several parameters with the same value but different suffixes.

legborder = true|false (*Boolean*)

If true, a line border is drawn around the legend.

[Default: true]

legend = true|false (*Boolean*)

Whether to draw a legend or not. If no value is supplied, the decision is made automatically: a legend is drawn only if it would have more than one entry.

leglabelN = <text> (*String*)

Sets the presentation label for the layer with a given suffix. This is the text which is displayed in the legend, if present. Multiple layers may use the same label, in which case they will be combined to form a single legend entry.

If no value is supplied (the default), the suffix itself is used as the label.

legopaque = true|false (*Boolean*)

If true, the background of the legend is opaque, and the legend obscures any plot components behind it. Otherwise, it's transparent.

[Default: true]

legpos = <xfrac,yfrac> (*double[]*)

Determines the internal position of the legend on the plot. The value is a comma-separated pair of values giving the X and Y positions of the legend within the plotting bounds, so for instance "0.5,0.5" will put the legend right in the middle of the plot. If no value is supplied, the legend will appear outside the plot boundary.

legseq = <suffix>[,...] (*String[]*)

Determines which layers are represented in the legend (if present) and in which order they appear. The legend has a line for each layer label (as determined by the `leglabelN` parameter). If multiple layers have the same label, they will contribute to the same entry in the legend, with style icons plotted over each other. The value of this parameter is a sequence of layer suffixes, which determines the order in which the legend entries appear. Layers with suffixes missing from this list do not show up in the legend at all.

If no value is supplied (the default), the sequence is the same as the layer plotting sequence (see `seq`).

minor = true|false (*Boolean*)

If true, minor tick marks are painted along the axes as well as the major tick marks. Minor tick marks do not have associated grid lines.

[Default: true]

navaxes = xy|x|y (*boolean[]*)

Determines the axes which are affected by the interactive navigation actions (pan and zoom). The default is `xy`, which means that the various mouse gestures will provide panning and zooming in both X and Y directions. However, if it is set to (for instance) `x` then the mouse will only allow panning and zooming in the horizontal direction, with the vertical extent fixed.

[Default: xy]

omode = swing|out|cgi|discard|auto (*PaintMode*)

Determines how the drawn plot will be output, see Section 8.5.

- `swing`: Plot will be displayed in a window on the screen. This plot is "live"; it can be

resized and (except for old-style plots) navigated around with mouse actions in the same way as plots in TOPCAT.

- `out`: Plot will be written to a file given by `out` using the graphics format given by `ofmt`.
- `cgi`: Plot will be written in a way suitable for CGI use direct from a web server. The output is in the graphics format given by `ofmt`, preceded by a suitable "Content-type" declaration.
- `discard`: Plot is drawn, but discarded. There is no output.
- `auto`: Behaves as `swing` or `out` mode depending on presence of `out` parameter

[Default: `auto`]

parallel = `<int-value>` (*Integer*)

Determines how many threads will run in parallel if animation output is being produced. Only used if the `animate` parameter is supplied. The default value is the number of processors apparently available to the JVM.

[Default: 8]

seq = `<suffix>[,...]` (*String[]*)

Contains a comma-separated list of layer suffixes to determine the order in which layers are drawn on the plot. This can affect which symbol are plotted on top of, and so potentially obscure, which other ones.

When specifying a plot, multiple layers may be specified, each introduced by a parameter `layer<N>`, where `<N>` is a different (arbitrary) suffix labelling the layer, and is appended to all the parameters specific to defining that layer.

By default the layers are drawn on the plot in the order in which the `layer*` parameters appear on the command line. However if this parameter is specified, each comma-separated element is interpreted as a layer suffix, giving the ordered list of layers to plot. Every element of the list must be a suffix with a corresponding `layer` parameter, but missing or repeated elements are allowed.

storage = `simple|cache|basic-cache` (*DataStoreFactory*)

Determines the way that data is accessed when constructing the plot. There are two basic options, `cached` or `not`.

If no caching is used (`simple`) then rows are read sequentially from the specified input table(s) every time they are required. This generally requires a small memory footprint (though that can depend on how the table is specified) and makes sense if the data only needs to be scanned once or perhaps if the table is very large.

If caching is used (`cache`) then the required data is read once from the specified input table(s) and cached before any plotting is performed, and plots are done using this cached data. This may use a significant amount of memory for large tables but it's usually more sensible (faster) if the data will need to be scanned multiple times.

The default value is `cache` if a live plot is being generated (`omode=swing`), since in that case the plot needs to be redrawn every time the user performs plot navigation actions or resizes the window, or if animations are being produced. Otherwise (e.g. output to a graphics file) the default is `simple`.

[Default: `simple`]

texttype = `plain|antialias|latex` (*TextSyntax*)

Determines how to turn label text into characters on the plot. `Plain` and `Antialias` both take the text at face value, but `Antialias` smooths the characters. `LaTeX` interprets the text as LaTeX source code and typesets it accordingly.

When not using LaTeX, antialiased text usually looks nicer, but can be perceptibly slower to plot. At time of writing, on MacOS antialiased text seems to be required to stop the writing coming out upside-down for non-horizontal text (MacOS java bug).

[Default: plain]

title = <value> (*String*)

Text of a title to be displayed at the top of the plot. If null, the default, no title is shown and there's more space for the graphics.

xanchor = true|false (*Boolean*)

If true, then zoom actions will work in such a way that the zero point on the X axis stays in the same position on the plot.

[Default: false]

xcrowd = <number> (*Double*)

Determines how closely the tick marks are spaced on the X axis. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

[Default: 1]

xflip = true|false (*Boolean*)

If true, the scale on the X axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

xlabel = <text> (*String*)

Gives a label to be used for annotating axis X. A default value based on the plotted data will be used if no value is supplied.

[Default: x]

xlog = true|false (*Boolean*)

If false (the default), the scale on the X axis is linear, if true it is logarithmic.

[Default: false]

xmax = <number> (*Double*)

Maximum value of the data coordinate on the X axis. This sets the value before any sub-ranging is applied. If not supplied, the value is determined from the plotted data.

xmin = <number> (*Double*)

Minimum value of the data coordinate on the X axis. This sets the value before any sub-ranging is applied. If not supplied, the value is determined from the plotted data.

xpix = <int-value> (*Integer*)

Size of the output image in the X direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also *insets*.

[Default: 500]

xsub = <lo>,<hi> (*Subrange*)

Defines a normalised adjustment to the data range of the X axis. The value may be specified as a comma-separated pair of two numbers, giving the lower and upper bounds of the range of interest respectively. This sub-range is applied to the data range that would otherwise be used, either automatically calculated or explicitly supplied; zero corresponds to the lower bound and one to the upper.

The default value "0,1" therefore has no effect. The range could be restricted to its lower half with the value 0,0.5.

[Default: 0,1]

yanchor = true|false (*Boolean*)

If true, then zoom actions will work in such a way that the zero point on the Y axis stays in the same position on the plot.

[Default: false]

ycrowd = <number> (*Double*)

Determines how closely the tick marks are spaced on the Y axis. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

[Default: 1]

yflip = true|false (*Boolean*)

If true, the scale on the Y axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

ylabel = <text> (*String*)

Gives a label to be used for annotating axis Y. A default value based on the plotted data will be used if no value is supplied.

[Default: y]

ylog = true|false (*Boolean*)

If false (the default), the scale on the Y axis is linear, if true it is logarithmic.

[Default: false]

ymax = <number> (*Double*)

Maximum value of the data coordinate on the Y axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

ymin = <number> (*Double*)

Minimum value of the data coordinate on the Y axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

ypix = <int-value> (*Integer*)

Size of the output image in the Y direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also *insets*.

[Default: 400]

ysub = <lo>,<hi> (*Subrange*)

Defines a normalised adjustment to the data range of the Y axis. The value may be specified as a comma-separated pair of two numbers, giving the lower and upper bounds of the range of interest respectively. This sub-range is applied to the data range that would otherwise be used, either automatically calculated or explicitly supplied; zero corresponds to the lower bound and one to the upper.

The default value "0,1" therefore has no effect. The range could be restricted to its lower half with the value 0,0.5.

[Default: 0,1]

zoomfactor = <number> (*Double*)

Sets the amount by which the plot view zooms in or out for each unit of mouse wheel movement. A value of 1 means that mouse wheel zooming has no effect. A higher value means that the mouse wheel zooms faster and a value nearer 1 means it zooms slower. Values below 1 are not permitted.

[Default: 1.2]

B.7.2 Examples

Here are some examples of `plot2plane`:

```
stilts plot2plane yflip=true layer_1=mark in_1=cat.fits x_1=BMAG-RMAG y_1=BMAG
```

This is a colour-magnitude diagram where the input table has columns named RMAG and BMAG. The Y axis is inverted so that the magnitude values increase downwards not up. The plot is displayed in a window on the screen, and may be panned and zoomed with the mouse.

```
stilts plot2plane layer=histogram in=hip_main.fits x=plx xlog=true
xlabel=Parallax ylabel=
```

Plots a histogram of parallaxes for Hipparcos data, with a logarithmic X axis. The axes are labelled explicitly, with an empty string in the case of the Y axis.

```
stilts plot2plane xpix=600 ypix=500
in=gavo_g2.fits x=X y=Y
shading=aux aux='atan2(vely,velx)' auxmap=hue auxvisible=false
layer_m=mark shape_m=cross size_m=4
layer_v=xyvector xdelta_v=velx ydelta_v=vely scale_v=2
out=velocities.pdf
```

Two layers are plotted, point markers representing position (4 pixels radius, shaped like crosses) and vectors representing velocity. Both markers and vectors are coloured according to the direction ($\arctan(\text{vely}/\text{velx})$) of the arrows, so it's easy to see points moving in similar directions; the "hue" colour map is good for this, since it's periodic, so values of $+\pi$ and $-\pi$ have the same colour. Since it's not very revealing in this case, display of the aux axis colour ramp beside the plot has been turned off. Since the X and Y coordinates and the colouring is common to both layers, the relevant parameters can be given without suffixes to avoid having to repeat them. Output is to a PDF file.

```
stilts plot2plane xmin=0 xmax=6.283 ymin=-1 ymax=1 xlabel=Time
layer=function axis=horizontal xname=time fexpr='sin(time)'
dash=3,2 thick=4 color=ee6aa7
```

Plots a sine curve to the screen. Initially the view is of one period, but you can pan and zoom interactively to see any range. The line is plotted in hot pink, four pixels wide, with a custom dash pattern. Since the function layer type has no data coordinates, no input table is required. The layer suffix here is the empty string; since there's only one layer, it doesn't cause any problems.

```
stilts plot2plane ylog=true xflip=true xmin=-5.2 xmax=3.8 ymin=250 ymax=3.5e5
in1=6dfgs_E7.fits x1=bmag-rmag y1=vel
layer1a=mark color1a=cyan
layer1b=contour color1b=yellow smooth1b=9 scaling1b=log
layer1c=mark icmd1c='every 35;select star'
shapelc=filled_triangle_down size1c=5 color1c=red
shading1c=transparent opaquelc=3
layer2=function fexpr2='exp(x*2+12)' color2=black antialias2=true
dash2=dash thick2=3
leglabella=Population leglabel1c=Sample legpos=.95,.95 legseq=1a,1c
fontsize=16 texttype=latex ylabel="v\,/\/,km.s^{-1}" xlabel=colour
```

There are four layers: 1a, 1b and 1c use the same positional data from the same input file, so the positional coordinates common to them are given the suffix "1". Layer "2" is unrelated, and has no input data, since it's just an analytic function. The legend is positioned to taste, and its content is manipulated so that only datasets 1a and 1c are described, and they are given custom names (the default would be their suffix names).

B.8 `plot2sky`: Draws a sky plot

`plot2sky` draws plots on the celestial sphere. This can be represented in a number of ways, controlled by the `projection` parameter; by default the view is of a rotatable sphere seen from the outside (which approximates to a tangent projection for small regions of the sky), but Aitoff and Plate Carée projections are also available. A number of options are also provided for drawing and labelling the grid showing celestial coordinates.

Positional coordinates are specified as `lon`, `lat` pairs giving longitude and latitude in decimal degrees. By default these are represented in the output in the same, unlabelled, coordinate system. However the command can also transform between different coordinate systems if you specify the data and view systems e.g.:

```
plot2sky viewsys=galactic
        layer1=mark in1=cat.fits lon1=RA2000 lat1=DEC2000 datasys1=equatorial
```

Content is added to the plot by specifying one or more *plot layers* using the `layerN` parameter. The `N` part is a suffix applied to all the parameters affecting a given layer; any suffix (including the empty string) may be used. Available layers for this plot type are: `mark` (Section 8.3.1), `size` (Section 8.3.2), `sizexy` (Section 8.3.3), `skyvector` (Section 8.3.20), `skyellipse` (Section 8.3.21), `link2` (Section 8.3.7), `mark2` (Section 8.3.8), `label` (Section 8.3.11), `contour` (Section 8.3.12), `skydensity` (Section 8.3.22).

B.8.1 Usage

The usage of `plot2sky` is

```
stilts <stilts-flags> plot2sky xpix=<int-value> ypix=<int-value>
                                insets=<top>,<left>,<bottom>,<right>
                                omode=swing|out|cgi|discard|auto
                                storage=simple|cache|basic-cache
                                seq=<suffix>[,...] legend=true|false
                                legborder=true|false legopaque=true|false
                                legseq=<suffix>[,...] legpos=<xfrac,yfrac>
                                title=<value>
                                auxmap=inferno|magma|plasma|...
                                auxclip=<lo>,<hi> auxflip=true|false
                                auxquant=<number>
                                auxfunc=log|linear|sqrt|square
                                auxmin=<number> auxmax=<number>
                                auxlabel=<text> auxcrowd=<factor>
                                auxvisible=true|false forcebitmap=true|false
                                compositor=0..1 animate=<table>
                                afmt=<in-format> astream=true|false
                                acmd=<cmds> parallel=<int-value>
                                projection=sin|aitoff|car
                                viewsys=equatorial|galactic|supergalactic|ecliptic
                                reflectlon=true|false grid=true|false
                                labelpos=Auto|External|Internal|Basic|Hybrid|None
                                sex=true|false crowd=<number>
                                gridcolor=<rrggb>|red|blue|...
                                labelcolor=<rrggb>|red|blue|...
                                gridaa=true|false
                                texttype=plain|antialias|latex
                                fontsize=<int-value>
                                fontstyle=standard|serif|mono
                                fontweight=plain|bold|italic|bold_italic
                                clon=<number> clat=<number> radius=<number>
                                zoomfactor=<number> leglabelN=<text>
                                layerN=<layer-type> <layerN-specific-params>
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the `Task` class for this command is `uk.ac.starlink.ttools.plot2.task.SkyPlot2Task`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

acmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the animation control table as specified by parameter `animate`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

afmt = <in-format> (*String*)

Specifies the format of the animation control table as specified by parameter `animate`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

animate = <table> (*StarTable*)

If not null, this parameter causes the command to create a sequence of plots instead of just one. The parameter value is a table with one row for each frame to be produced. Columns in the table are interpreted as parameters which may take different values for each frame; the column name is the parameter name, and the value for a given frame is its value from that row. Animating like this is considerably more efficient than invoking the `STILTS` command in a loop.

The location of the animation control table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `afmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

astream = true|false (*Boolean*)

If set true, the animation control table specified by the `animate` parameter will be read as a stream. It is necessary to give the `afmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as `VOTable`).

[Default: `false`]

auxclip = <lo>,<hi> (*Subrange*)

Defines a subrange of the colour ramp to be used for Aux shading. The is specified as a (low,high) comma-separated pair of two numbers between 0 and 1.

If the full range 0,1 (the default) is used, the whole range of colours specified by the selected shader will be used. But if, for instance a value of 0,0.5 is given, only those colours at the left hand end of the ramp will be seen.

[Default: 0,1]

auxcrowd = <factor> (Double)

Determines how closely the tick marks are spaced on the Aux axis, if visible. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

[Default: 1.0]

auxflip = true|false (Boolean)

If true, the colour map on the Aux axis will be reversed.

[Default: false]

auxfunc = log|linear|sqrt|square (Scaling)

Defines the way that values in the Aux range are mapped to the selected colour ramp.

The available options are:

- log: Logarithmic scaling
- linear: Linear scaling
- sqrt: Square root scaling
- square: Square scaling

[Default: linear]

auxlabel = <text> (String)

Sets the label used to annotate the aux axis, if it is visible.

auxmap = inferno|magma|plasma|... (Shader)

Color map used for Aux axis shading.

A mixed bag of colour ramps are available: inferno, magma, plasma, viridis, cubehelix, sron, rainbow, rainbow2, rainbow3, pastel, accent, gnuplot, gnuplot2, specxby, set1, paired, hotcold, rdbu, piyg, brbg, cyan-magenta, red-blue, brg, heat, cold, light, greyscale, colour, standard, bugn, bupu, orrd, pubu, purd, huecl, hue, intensity, rgb_red, rgb_green, rgb_blue, hsv_h, hsv_s, hsv_v, yuv_y, yuv_u, yuv_v, scale_hsv_s, scale_hsv_v, scale_yuv_y, mask, blacker, whiter, transparency. *Note:* many of these, including rainbow-like ones, are frowned upon by the visualisation community.

[Default: inferno]

auxmax = <number> (Double)

Maximum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

auxmin = <number> (Double)

Minimum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

auxquant = <number> (Double)

Allows the colour map used for the Aux axis to be quantised. If an integer value N is chosen then the colour map will be viewed as N discrete evenly-spaced levels, so that only N different colours will appear in the plot. This can be used to generate a contour-like effect, and may make it easier to trace the boundaries of regions of interest by eye.

If left blank, the colour map is nominally continuous (though in practice it may be quantised to a medium-sized number like 256).

auxvisible = true|false (Boolean)

Determines whether the aux axis colour ramp is displayed alongside the plot.

If not supplied (the default), the aux axis will be visible when aux shading is used in any of the plotted layers.

clat = <number> (*Double*)

Latitude of the central position of the plot in decimal degrees. Use with **clon** and **radius**. If the center is not specified, the field of view is determined from the data.

clon = <number> (*Double*)

Longitude of the central position of the plot in decimal degrees. Use with **clat** and **radius**. If the center is not specified, the field of view is determined from the data.

compositor = 0..1 (*Compositor*)

Defines how multiple overplotted partially transparent pixels are combined to form a resulting colour. The way this is used depends on the details of the specified plot.

Currently, this parameter takes a "boost" value in the range 0..1. If the value is zero, saturation semantics are used: RGB colours are added in proportion to their associated alpha value until the total alpha is saturated (reaches 1), after which additional pixels have no further effect. For larger boost values, the effect is similar, but any non-zero alpha in the output is boosted to the given minimum value. The effect of this is that even very slightly populated pixels can be visually distinguished from unpopulated ones which may not be the case for saturation composition.

[Default: 0.05]

crowd = <number> (*Double*)

Determines how closely sky grid lines are spaced. The default value is 1, meaning normal crowding. Larger values result in more grid lines, and smaller values in fewer grid lines.

[Default: 1]

fontsize = <int-value> (*Integer*)

Size of the text font in points.

[Default: 12]

fontstyle = standard|serif|mono (*FontType*)

Font style for text.

The available options are:

- standard
- serif
- mono

[Default: standard]

fontweight = plain|bold|italic|bold_italic (*FontWeight*)

Font weight for text.

The available options are:

- plain
- bold
- italic
- bold_italic

[Default: plain]

forcebitmap = true|false (*Boolean*)

This option only has an effect when writing output to vector graphics formats (PDF and PostScript). If set **true**, the data contents of the plot are drawn as a pixel map embedded into the output file rather than plotting each point in the output. This may make the output less

beautiful (round markers will no longer be perfectly round), but it may result in a much smaller file if there are very many data points. Plot annotations such as axis labels will not be affected - they are still drawn as vector text. Note that in some cases (e.g. `shadingN=auto` or `shadingN=density`) this kind of pixellisation will happen in any case.

[Default: `false`]

grid = true|false (*Boolean*)

If true, sky coordinate grid lines are drawn on the plot. If false, they are absent.

[Default: `true`]

griddaa = true|false (*Boolean*)

If true, grid lines are drawn with antialiasing. Antialiased lines look smoother, but may take perceptibly longer to draw. Only has any effect for bitmapped output formats.

[Default: `false`]

gridcolor = <rrggb>|red|blue|... (*Color*)

The color of the plot grid.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: `light_grey`]

insets = <top>,<left>,<bottom>,<right> (*Insets*)

Defines the amount of space in pixels around the actual plotting area. This space is used for axis labels, and other decorations and any left over forms an empty border.

The size and position of the actual plotting area is determined by this parameter along with `xpix` and `ypix`. If no value is set (the default), the insets will be determined automatically according to how much space is required for labels etc.

labelcolor = <rrggb>|red|blue|... (*Color*)

The color of axis labels and other plot annotations.

The value may be a six-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colors. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black, light_grey, white.

[Default: `black`]

labelpos = Auto|External|Internal|Basic|Hybrid|None (*SkyAxisLabeller*)

Controls whether and where the numeric annotations of the lon/lat axes are displayed. The default option `Auto` usually does the sensible thing, but other options exist to force labelling internally or externally to the plot region, or to remove numeric labels altogether.

The available options are:

- `Auto`: Uses `External` or `Internal` policy according to whether the sky fills the plot bounds or not
- `External`: Labels are drawn outside the plot bounds
- `Internal`: Labels are drawn inside the plot bounds
- `Basic`: Labels are drawn somewhere near the grid line
- `Hybrid`: Grid lines are labelled outside the plot bounds where possible, but inside if they would otherwise be invisible
- `None`: Axes are not labelled

[Default: `Auto`]

layerN = <layer-type> <layerN-specific-params> (*LayerType*)

Selects one of the available plot types for layerN. A plot consists of a plotting surface, set up using the various unsuffixed parameters of the plotting command, and zero or more plot layers. Each layer is introduced by a parameter with the name `layer<N>` where the suffix "`<N>`" is a label identifying the layer and is appended to all the parameter names which configure that layer. Suffixes may be any string, including the empty string.

This parameter may take one of the following values, described in more detail in Section 8.3:

- `mark` (Section 8.3.1)
- `size` (Section 8.3.2)
- `sizexy` (Section 8.3.3)
- `skyvector` (Section 8.3.20)
- `skyellipse` (Section 8.3.21)
- `link2` (Section 8.3.7)
- `mark2` (Section 8.3.8)
- `label` (Section 8.3.11)
- `contour` (Section 8.3.12)
- `skydensity` (Section 8.3.22)

Each of these layer types comes with a list of type-specific parameters to define the details of that layer, including some or all of the following groups:

- input table parameters (e.g. `inN`, `icmdN`)
- coordinate params referring to input table columns (e.g. `xN`, `yN`)
- layer style parameters (e.g. `shadingN`, `colorN`)

Every parameter notionally carries the same suffix `N`. However, if the suffix is not present, the application will try looking for a parameter with the same name with no suffix instead. In this way, if several layers have the same value for a given parameter (for instance input table), you can supply it using one unsuffixed parameter to save having to supply several parameters with the same value but different suffixes.

legborder = `true|false` (*Boolean*)

If true, a line border is drawn around the legend.

[Default: `true`]

legend = `true|false` (*Boolean*)

Whether to draw a legend or not. If no value is supplied, the decision is made automatically: a legend is drawn only if it would have more than one entry.

leglabelN = <text> (*String*)

Sets the presentation label for the layer with a given suffix. This is the text which is displayed in the legend, if present. Multiple layers may use the same label, in which case they will be combined to form a single legend entry.

If no value is supplied (the default), the suffix itself is used as the label.

legopaque = `true|false` (*Boolean*)

If true, the background of the legend is opaque, and the legend obscures any plot components behind it. Otherwise, it's transparent.

[Default: `true`]

legpos = <xfrac,yfrac> (*double[]*)

Determines the internal position of the legend on the plot. The value is a comma-separated pair of values giving the X and Y positions of the legend within the plotting bounds, so for instance "`0.5,0.5`" will put the legend right in the middle of the plot. If no value is supplied, the legend will appear outside the plot boundary.

legseq = <suffix>[,...] (*String[]*)

Determines which layers are represented in the legend (if present) and in which order they appear. The legend has a line for each layer label (as determined by the `leglabelN` parameter). If multiple layers have the same label, they will contribute to the same entry in the legend, with style icons plotted over each other. The value of this parameter is a sequence of layer suffixes, which determines the order in which the legend entries appear. Layers with suffixes missing from this list do not show up in the legend at all.

If no value is supplied (the default), the sequence is the same as the layer plotting sequence (see `seq`).

omode = swing|out|cgi|discard|auto (*PaintMode*)

Determines how the drawn plot will be output, see Section 8.5.

- `swing`: Plot will be displayed in a window on the screen. This plot is "live"; it can be resized and (except for old-style plots) navigated around with mouse actions in the same way as plots in TOPCAT.
- `out`: Plot will be written to a file given by `out` using the graphics format given by `ofmt`.
- `cgi`: Plot will be written in a way suitable for CGI use direct from a web server. The output is in the graphics format given by `ofmt`, preceded by a suitable "Content-type" declaration.
- `discard`: Plot is drawn, but discarded. There is no output.
- `auto`: Behaves as `swing` or `out` mode depending on presence of `out` parameter

[Default: `auto`]

parallel = <int-value> (*Integer*)

Determines how many threads will run in parallel if animation output is being produced. Only used if the `animate` parameter is supplied. The default value is the number of processors apparently available to the JVM.

[Default: 8]

projection = sin|aitoff|car (*Projection*)

Sky projection used to display the plot.

The available options are:

- `sin`: rotatable sphere
- `aitoff`: Hammer-Aitoff projection
- `car`: Plate Carree projection (lon/lat on Cartesian axes)

[Default: `sin`]

radius = <number> (*Double*)

Approximate radius of the plot field of view in degrees. Only used if `c lon` and `c lat` are also specified.

[Default: 1]

reflectlon = true|false (*Boolean*)

Whether to invert the celestial sphere by displaying the longitude axis increasing right-to-left rather than left-to-right. It is conventional to display the celestial sphere in this way because that's what it looks like from the earth, so the default is `true`. Set it false to see the sphere from the outside.

[Default: `true`]

seq = <suffix>[,...] (*String[]*)

Contains a comma-separated list of layer suffixes to determine the order in which layers are drawn on the plot. This can affect which symbol are plotted on top of, and so potentially obscure, which other ones.

When specifying a plot, multiple layers may be specified, each introduced by a parameter

`layer<N>`, where `<N>` is a different (arbitrary) suffix labelling the layer, and is appended to all the parameters specific to defining that layer.

By default the layers are drawn on the plot in the order in which the `layer*` parameters appear on the command line. However if this parameter is specified, each comma-separated element is interpreted as a layer suffix, giving the ordered list of layers to plot. Every element of the list must be a suffix with a corresponding `layer` parameter, but missing or repeated elements are allowed.

sex = true|false (*Boolean*)

If true, grid line labels are written in sexagesimal notation, if false in decimal degrees.

[Default: true]

storage = simple|cache|basic-cache (*DataStoreFactory*)

Determines the way that data is accessed when constructing the plot. There are two basic options, cached or not.

If no caching is used (`simple`) then rows are read sequentially from the specified input table(s) every time they are required. This generally requires a small memory footprint (though that can depend on how the table is specified) and makes sense if the data only needs to be scanned once or perhaps if the table is very large.

If caching is used (`cache`) then the required data is read once from the specified input table(s) and cached before any plotting is performed, and plots are done using this cached data. This may use a significant amount of memory for large tables but it's usually more sensible (faster) if the data will need to be scanned multiple times.

The default value is `cache` if a live plot is being generated (`omode=swing`), since in that case the plot needs to be redrawn every time the user performs plot navigation actions or resizes the window, or if animations are being produced. Otherwise (e.g. output to a graphics file) the default is `simple`.

[Default: simple]

texttype = plain|antialias|latex (*TextSyntax*)

Determines how to turn label text into characters on the plot. `Plain` and `Antialias` both take the text at face value, but `Antialias` smooths the characters. `LaTeX` interprets the text as LaTeX source code and typesets it accordingly.

When not using LaTeX, antialiased text usually looks nicer, but can be perceptibly slower to plot. At time of writing, on MacOS antialiased text seems to be required to stop the writing coming out upside-down for non-horizontal text (MacOS java bug).

[Default: plain]

title = <value> (*String*)

Text of a title to be displayed at the top of the plot. If null, the default, no title is shown and there's more space for the graphics.

viewsys = equatorial|galactic|supergalactic|ecliptic (*SkySys*)

The sky coordinate system used for the generated plot.

Choice of this value goes along with the data coordinate system that may be specified for plot layers. If unspecified, a generic longitude/latitude system is used, and all lon/lat coordinates in the plotted data layers are assumed to be in the same system. If a value is supplied for this parameter, then a sky system must (implicitly or explicitly) be supplied for each data layer, and the coordinates are converted from data to view system before being plotted.

The available options are:

- `equatorial`: J2000 equatorial system
- `galactic`: IAU 1958 galactic system
- `supergalactic`: De Vaucouleurs supergalactic system

- `ecliptic`: ecliptic system based on conversion at 2000.0

xpix = <int-value> *(Integer)*

Size of the output image in the X direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also `insets`.

[Default: 500]

ypix = <int-value> *(Integer)*

Size of the output image in the Y direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also `insets`.

[Default: 400]

zoomfactor = <number> *(Double)*

Sets the amount by which the plot view zooms in or out for each unit of mouse wheel movement. A value of 1 means that mouse wheel zooming has no effect. A higher value means that the mouse wheel zooms faster and a value nearer 1 means it zooms slower. Values below 1 are not permitted.

[Default: 1.2]

B.8.2 Examples

Here are some examples of `plot2sky`:

```
stilts plot2sky in=messier.xml lon=RA lat=DEC
               layer.pos=mark size.pos=4
               layer.txt=label label.txt=Name layer.color=grey
```

Plots the positions of all the Messier objects on the sky, with text labels giving their object names. This displays a sphere on the screen that you can rotate/zoom using the mouse.

```
stilts plot2sky projection=aitoff
               xpix=600 ypix=300
               gridcolour=green labelcolour=black
               fontsize=10 gridaa=true texttype=antialias
               sex=true crowd=4
```

This just plots a celestial coordinate grid with no data. Various options are tweaked to adjust the appearance of the grid.

```
stilts plot2sky xpix=1000 ypix=500 fontsize=18 crowd=2
               projection=aitoff viewsys=galactic
               layer1=mark size1=0
               shading1=density densemap1=gnuplot2 densefunc1=log
               densesub1=0.5,.95 denseclip1=0.02,1
               in1=gums_mw_all.fits
               lon1=alpha lat1=delta datasys1=equatorial
               icmd1=progress out=mw.pdf
```

Makes an all-sky plot using an Aitoff projection into galactic coordinates of a large dataset. Density shading means that the colour at each point is dependent on how many points are plotted; the density colour map has been fine-tuned here to get a specific visual effect. The sky coordinates in the input file (alpha and delta) are equatorial, but these are transformed to galactic coordinates for plotting. The progress filter applied to the input table displays a progress indicator on the console to see how far it's got. The result is written to a PDF file.

This command was used to plot the GUMS-10 MW dataset, a simulation of the milky way stars seen by the Gaia satellite; The 2.1 billion row plot took about 45 minutes.

B.9 plot2cube: Draws a cube plot

`plot2cube` draws plots in a Cartesian 3-dimensional space. The plotting volume is a cube, which is viewed from the outside and usually bounded by an annotated wire frame.

Positional coordinates are specified as *x*, *y*, *z* triples, e.g.:

```
plot2cube layer1=mark in1=sim.fits x1=XPOS y1=YPOS z1=ZPOS
```

Content is added to the plot by specifying one or more *plot layers* using the `layerN` parameter. The *N* part is a suffix applied to all the parameters affecting a given layer; any suffix (including the empty string) may be used. Available layers for this plot type are: `mark` (Section 8.3.1), `size` (Section 8.3.2), `sizexy` (Section 8.3.3), `xyzvector` (Section 8.3.23), `xyzerror` (Section 8.3.24), `link2` (Section 8.3.7), `mark2` (Section 8.3.8), `label` (Section 8.3.11), `contour` (Section 8.3.12).

B.9.1 Usage

The usage of `plot2cube` is

```
stilts <stilts-flags> plot2cube xpix=<int-value> ypix=<int-value>
insets=<top>,<left>,<bottom>,<right>
omode=swing|out|cgi|discard|auto
storage=simple|cache|basic-cache
seq=<suffix>[,...] legend=true|false
legborder=true|false legopaque=true|false
legseq=<suffix>[,...] legpos=<xfrac,yfrac>
title=<value>
auxmap=inferno|magma|plasma|...
auxclip=<lo>,<hi> auxflip=true|false
auxquant=<number>
auxfunc=log|linear|sqrt|square
auxmin=<number> auxmax=<number>
auxlabel=<text> auxcrowd=<factor>
auxvisible=true|false
forcebitmap=true|false compositor=0..1
animate=<table> afmt=<in-format>
astream=true|false acmd=<cmds>
parallel=<int-value> xlog=true|false
ylog=true|false zlog=true|false
xflip=true|false yflip=true|false
zflip=true|false xlabel=<text>
ylabel=<text> ylabel=<text> xcrowd=<number>
ycrowd=<number> zcrowd=<number>
frame=true|false minor=true|false
griddaa=true|false
texttype=plain|antialias|latex
fontsize=<int-value>
fontstyle=standard|serif|mono
fontweight=plain|bold|italic|bold_italic
xmin=<number> xmax=<number> xsub=<lo>,<hi>
ymin=<number> ymax=<number> ysub=<lo>,<hi>
zmin=<number> zmax=<number> zsub=<lo>,<hi>
theta=<degrees> phi=<degrees> zoom=<factor>
xoff=<pixels> yoff=<pixels>
zoomaxes=[[x][y][z]] zoomfactor=<number>
leglabelN=<text>
layerN=<layer-type> <layerN-specific-params>
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the `Task` class for this command is `uk.ac.starlink.ttools.plot2.task.CubePlot2Task`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

acmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the animation control table as specified by parameter `animate`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

afmt = <in-format> (*String*)

Specifies the format of the animation control table as specified by parameter `animate`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

animate = <table> (*StarTable*)

If not null, this parameter causes the command to create a sequence of plots instead of just one. The parameter value is a table with one row for each frame to be produced. Columns in the table are interpreted as parameters which may take different values for each frame; the column name is the parameter name, and the value for a given frame is its value from that row. Animating like this is considerably more efficient than invoking the `STILTS` command in a loop.

The location of the animation control table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `afmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

astream = `true|false` (*Boolean*)

If set true, the animation control table specified by the `animate` parameter will be read as a stream. It is necessary to give the `afmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as `VOTable`).

[Default: `false`]

auxclip = <lo>,<hi> (*Subrange*)

Defines a subrange of the colour ramp to be used for Aux shading. The is specified as a (low,high) comma-separated pair of two numbers between 0 and 1.

If the full range 0,1 (the default) is used, the whole range of colours specified by the selected shader will be used. But if, for instance a value of 0,0.5 is given, only those colours at the left hand end of the ramp will be seen.

[Default: 0,1]

auxcrowd = <factor> (*Double*)

Determines how closely the tick marks are spaced on the Aux axis, if visible. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

[Default: 1.0]

auxflip = true|false (*Boolean*)

If true, the colour map on the Aux axis will be reversed.

[Default: false]

auxfunc = log|linear|sqrt|square (*Scaling*)

Defines the way that values in the Aux range are mapped to the selected colour ramp.

The available options are:

- log: Logarithmic scaling
- linear: Linear scaling
- sqrt: Square root scaling
- square: Square scaling

[Default: linear]

auxlabel = <text> (*String*)

Sets the label used to annotate the aux axis, if it is visible.

auxmap = inferno|magma|plasma|... (*Shader*)

Color map used for Aux axis shading.

A mixed bag of colour ramps are available: inferno, magma, plasma, viridis, cubehelix, sron, rainbow, rainbow2, rainbow3, pastel, accent, gnuplot, gnuplot2, specxby, set1, paired, hotcold, rdbu, piyg, brbg, cyan-magenta, red-blue, brg, heat, cold, light, greyscale, colour, standard, bugn, bupu, orrd, pubu, purd, huecl, hue, intensity, rgb_red, rgb_green, rgb_blue, hsv_h, hsv_s, hsv_v, yuv_y, yuv_u, yuv_v, scale_hsv_s, scale_hsv_v, scale_yuv_y, mask, blacker, whiter, transparency. *Note:* many of these, including rainbow-like ones, are frowned upon by the visualisation community.

[Default: inferno]

auxmax = <number> (*Double*)

Maximum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

auxmin = <number> (*Double*)

Minimum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

auxquant = <number> (*Double*)

Allows the colour map used for the Aux axis to be quantised. If an integer value N is chosen then the colour map will be viewed as N discrete evenly-spaced levels, so that only N different colours will appear in the plot. This can be used to generate a contour-like effect, and may make it easier to trace the boundaries of regions of interest by eye.

If left blank, the colour map is nominally continuous (though in practice it may be quantised to a medium-sized number like 256).

auxvisible = true|false (*Boolean*)

Determines whether the aux axis colour ramp is displayed alongside the plot.

If not supplied (the default), the aux axis will be visible when aux shading is used in any of the plotted layers.

compositor = 0..1 (*Compositor*)

Defines how multiple overplotted partially transparent pixels are combined to form a resulting colour. The way this is used depends on the details of the specified plot.

Currently, this parameter takes a "boost" value in the range 0..1. If the value is zero, saturation semantics are used: RGB colours are added in proportion to their associated alpha value until the total alpha is saturated (reaches 1), after which additional pixels have no further effect. For larger boost values, the effect is similar, but any non-zero alpha in the output is boosted to the given minimum value. The effect of this is that even very slightly populated pixels can be visually distinguished from unpopulated ones which may not be the case for saturation composition.

[Default: 0.05]

fontsize = <int-value> (*Integer*)

Size of the text font in points.

[Default: 12]

fontstyle = standard|serif|mono (*FontType*)

Font style for text.

The available options are:

- standard
- serif
- mono

[Default: standard]

fontweight = plain|bold|italic|bold_italic (*FontWeight*)

Font weight for text.

The available options are:

- plain
- bold
- italic
- bold_italic

[Default: plain]

forcebitmap = true|false (*Boolean*)

This option only has an effect when writing output to vector graphics formats (PDF and PostScript). If set `true`, the data contents of the plot are drawn as a pixel map embedded into the output file rather than plotting each point in the output. This may make the output less beautiful (round markers will no longer be perfectly round), but it may result in a much smaller file if there are very many data points. Plot annotations such as axis labels will not be affected - they are still drawn as vector text. Note that in some cases (e.g. `shadingN=auto` or `shadingN=density`) this kind of pixellisation will happen in any case.

[Default: false]

frame = true|false (*Boolean*)

If `true`, a cube wire frame with labelled axes is drawn to indicate the limits of the plotted 3D region. If `false`, no wire frame and no axes are drawn.

[Default: true]

griddaa = true|false (*Boolean*)

If `true`, grid lines are drawn with antialiasing. Antialiased lines look smoother, but may take perceptibly longer to draw. Only has any effect for bitmapped output formats.

[Default: false]

insets = <top>,<left>,<bottom>,<right> (*Insets*)

Defines the amount of space in pixels around the actual plotting area. This space is used for axis labels, and other decorations and any left over forms an empty border.

The size and position of the actual plotting area is determined by this parameter along with `xpix` and `ypix`. If no value is set (the default), the insets will be determined automatically according to how much space is required for labels etc.

layerN = <layer-type> <layerN-specific-params> (*LayerType*)

Selects one of the available plot types for layerN. A plot consists of a plotting surface, set up using the various unsuffixed parameters of the plotting command, and zero or more plot layers. Each layer is introduced by a parameter with the name `layer<N>` where the suffix "<N>" is a label identifying the layer and is appended to all the parameter names which configure that layer. Suffixes may be any string, including the empty string.

This parameter may take one of the following values, described in more detail in Section 8.3:

- `mark` (Section 8.3.1)
- `size` (Section 8.3.2)
- `sizexy` (Section 8.3.3)
- `xyzvector` (Section 8.3.23)
- `xyzerror` (Section 8.3.24)
- `link2` (Section 8.3.7)
- `mark2` (Section 8.3.8)
- `label` (Section 8.3.11)
- `contour` (Section 8.3.12)

Each of these layer types comes with a list of type-specific parameters to define the details of that layer, including some or all of the following groups:

- input table parameters (e.g. `inN`, `icmdN`)
- coordinate params referring to input table columns (e.g. `xN`, `yN`)
- layer style parameters (e.g. `shadingN`, `colorN`)

Every parameter notionally carries the same suffix `N`. However, if the suffix is not present, the application will try looking for a parameter with the same name with no suffix instead. In this way, if several layers have the same value for a given parameter (for instance input table), you can supply it using one unsuffixed parameter to save having to supply several parameters with the same value but different suffixes.

legborder = `true|false` (*Boolean*)

If true, a line border is drawn around the legend.

[Default: `true`]

legend = `true|false` (*Boolean*)

Whether to draw a legend or not. If no value is supplied, the decision is made automatically: a legend is drawn only if it would have more than one entry.

leglabelN = <text> (*String*)

Sets the presentation label for the layer with a given suffix. This is the text which is displayed in the legend, if present. Multiple layers may use the same label, in which case they will be combined to form a single legend entry.

If no value is supplied (the default), the suffix itself is used as the label.

legopaque = `true|false` (*Boolean*)

If true, the background of the legend is opaque, and the legend obscures any plot components behind it. Otherwise, it's transparent.

[Default: `true`]

legpos = <xfrac,yfrac> (*double[]*)

Determines the internal position of the legend on the plot. The value is a comma-separated pair of values giving the X and Y positions of the legend within the plotting bounds, so for instance "0.5,0.5" will put the legend right in the middle of the plot. If no value is supplied, the legend will appear outside the plot boundary.

legseq = <suffix>[,...] (*String[]*)

Determines which layers are represented in the legend (if present) and in which order they appear. The legend has a line for each layer label (as determined by the `leglabelN` parameter). If multiple layers have the same label, they will contribute to the same entry in the legend, with style icons plotted over each other. The value of this parameter is a sequence of layer suffixes, which determines the order in which the legend entries appear. Layers with suffixes missing from this list do not show up in the legend at all.

If no value is supplied (the default), the sequence is the same as the layer plotting sequence (see `seq`).

minor = true|false (*Boolean*)

If true, minor tick marks are painted along the axes as well as the major tick marks. Minor tick marks do not have associated grid lines.

[Default: true]

omode = swing|out|cgi|discard|auto (*PaintMode*)

Determines how the drawn plot will be output, see Section 8.5.

- `swing`: Plot will be displayed in a window on the screen. This plot is "live"; it can be resized and (except for old-style plots) navigated around with mouse actions in the same way as plots in TOPCAT.
- `out`: Plot will be written to a file given by `out` using the graphics format given by `ofmt`.
- `cgi`: Plot will be written in a way suitable for CGI use direct from a web server. The output is in the graphics format given by `ofmt`, preceded by a suitable "Content-type" declaration.
- `discard`: Plot is drawn, but discarded. There is no output.
- `auto`: Behaves as `swing` or `out` mode depending on presence of `out` parameter

[Default: auto]

parallel = <int-value> (*Integer*)

Determines how many threads will run in parallel if animation output is being produced. Only used if the `animate` parameter is supplied. The default value is the number of processors apparently available to the JVM.

[Default: 8]

phi = <degrees> (*Double*)

Rotation around the Z axis of the plotted 3d space applied before the plot is viewed.

[Default: -30]

seq = <suffix>[,...] (*String[]*)

Contains a comma-separated list of layer suffixes to determine the order in which layers are drawn on the plot. This can affect which symbol are plotted on top of, and so potentially obscure, which other ones.

When specifying a plot, multiple layers may be specified, each introduced by a parameter `layer<N>`, where `<N>` is a different (arbitrary) suffix labelling the layer, and is appended to all the parameters specific to defining that layer.

By default the layers are drawn on the plot in the order in which the `layer*` parameters appear on the command line. However if this parameter is specified, each comma-separated element is interpreted as a layer suffix, giving the ordered list of layers to plot. Every element of the list

must be a suffix with a corresponding `layer` parameter, but missing or repeated elements are allowed.

storage = simple|cache|basic-cache (*DataStoreFactory*)

Determines the way that data is accessed when constructing the plot. There are two basic options, cached or not.

If no caching is used (`simple`) then rows are read sequentially from the specified input table(s) every time they are required. This generally requires a small memory footprint (though that can depend on how the table is specified) and makes sense if the data only needs to be scanned once or perhaps if the table is very large.

If caching is used (`cache`) then the required data is read once from the specified input table(s) and cached before any plotting is performed, and plots are done using this cached data. This may use a significant amount of memory for large tables but it's usually more sensible (faster) if the data will need to be scanned multiple times.

The default value is `cache` if a live plot is being generated (`omode=swing`), since in that case the plot needs to be redrawn every time the user performs plot navigation actions or resizes the window, or if animations are being produced. Otherwise (e.g. output to a graphics file) the default is `simple`.

[Default: `simple`]

texttype = plain|antialias|latex (*TextSyntax*)

Determines how to turn label text into characters on the plot. `Plain` and `Antialias` both take the text at face value, but `Antialias` smooths the characters. `LaTeX` interprets the text as LaTeX source code and typesets it accordingly.

When not using LaTeX, antialiased text usually looks nicer, but can be perceptibly slower to plot. At time of writing, on MacOS antialiased text seems to be required to stop the writing coming out upside-down for non-horizontal text (MacOS java bug).

[Default: `plain`]

theta = <degrees> (*Double*)

Rotation towards the viewer in degrees of the plotted 3d space.

[Default: 15]

title = <value> (*String*)

Text of a title to be displayed at the top of the plot. If null, the default, no title is shown and there's more space for the graphics.

xcrowd = <number> (*Double*)

Determines how closely the tick marks are spaced on the X axis. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

[Default: 1]

xflip = true|false (*Boolean*)

If true, the scale on the X axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: `false`]

xlabel = <text> (*String*)

Gives a label to be used for annotating axis X A default value based on the plotted data will be used if no value is supplied.

[Default: `x`]

xlog = true|false (*Boolean*)

If false (the default), the scale on the X axis is linear, if true it is logarithmic.

[Default: false]

xmax = <number> (Double)

Maximum value of the data coordinate on the X axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

xmin = <number> (Double)

Minimum value of the data coordinate on the X axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

xoff = <pixels> (Double)

Shifts the whole plot within the plotting region by the given number of pixels in the horizontal direction.

[Default: 0]

xpix = <int-value> (Integer)

Size of the output image in the X direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also *insets*.

[Default: 500]

xsub = <lo>,<hi> (Subrange)

Defines a normalised adjustment to the data range of the X axis. The value may be specified as a comma-separated pair of two numbers, giving the lower and upper bounds of the range of interest respectively. This sub-range is applied to the data range that would otherwise be used, either automatically calculated or explicitly supplied; zero corresponds to the lower bound and one to the upper.

The default value "0,1" therefore has no effect. The range could be restricted to its lower half with the value 0,0.5.

[Default: 0,1]

ycrowd = <number> (Double)

Determines how closely the tick marks are spaced on the Y axis. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

[Default: 1]

yflip = true|false (Boolean)

If true, the scale on the Y axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

ylabel = <text> (String)

Gives a label to be used for annotating axis Y. A default value based on the plotted data will be used if no value is supplied.

[Default: Y]

ylog = true|false (Boolean)

If false (the default), the scale on the Y axis is linear, if true it is logarithmic.

[Default: false]

ymax = <number> (Double)

Maximum value of the data coordinate on the Y axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

ymin = <number> (Double)

Minimum value of the data coordinate on the Y axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

yoff = <pixels> (*Double*)

Shifts the whole plot within the plotting region by the given number of pixels in the vertical direction.

[Default: 0]

ypix = <int-value> (*Integer*)

Size of the output image in the Y direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also *insets*.

[Default: 400]

ysub = <lo>,<hi> (*Subrange*)

Defines a normalised adjustment to the data range of the Y axis. The value may be specified as a comma-separated pair of two numbers, giving the lower and upper bounds of the range of interest respectively. This sub-range is applied to the data range that would otherwise be used, either automatically calculated or explicitly supplied; zero corresponds to the lower bound and one to the upper.

The default value "0,1" therefore has no effect. The range could be restricted to its lower half with the value 0,0.5.

[Default: 0,1]

zcrowd = <number> (*Double*)

Determines how closely the tick marks are spaced on the Z axis. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

[Default: 1]

zflip = true|false (*Boolean*)

If true, the scale on the Z axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

zlabel = <text> (*String*)

Gives a label to be used for annotating axis Z. A default value based on the plotted data will be used if no value is supplied.

[Default: z]

zlog = true|false (*Boolean*)

If false (the default), the scale on the Z axis is linear, if true it is logarithmic.

[Default: false]

zmax = <number> (*Double*)

Maximum value of the data coordinate on the Z axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

zmin = <number> (*Double*)

Minimum value of the data coordinate on the Z axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

zoom = <factor> (*Double*)

Sets the magnification factor at which the the plotted 3D region itself is viewed, without affecting its contents. The default value is 1, which means the cube fits into the plotting space however it is rotated. Much higher zoom factors will result in parts of the plotting region and axes being drawn outside of the plotting region (so invisible).

[Default: 1]

zoomaxes = [[x][y][z]] *(boolean[])*

Determines which axes are affected by zoom navigation actions.

If no value is supplied (the default), the mouse wheel zooms around the center of the cube, and right-button (or CTRL-) drag zooms in the two dimensions most closely aligned with the plane of the screen, with the reference position set by the initial position of the mouse.

If this value is set (legal values are x, y, z, xy, yz, xz and xyz) then all zoom operations are around the cube center and affect the axes named.

zoomfactor = <number> *(Double)*

Sets the amount by which the plot view zooms in or out for each unit of mouse wheel movement. A value of 1 means that mouse wheel zooming has no effect. A higher value means that the mouse wheel zooms faster and a value nearer 1 means it zooms slower. Values below 1 are not permitted.

[Default: 1.2]

zsub = <lo>,<hi> *(Subrange)*

Defines a normalised adjustment to the data range of the Z axis. The value may be specified as a comma-separated pair of two numbers, giving the lower and upper bounds of the range of interest respectively. This sub-range is applied to the data range that would otherwise be used, either automatically calculated or explicitly supplied; zero corresponds to the lower bound and one to the upper.

The default value "0,1" therefore has no effect. The range could be restricted to its lower half with the value 0,0.5.

[Default: 0,1]

B.9.2 Examples

Some examples of `plot2cube` are shown below. See Section 8.1.3 for some examples of producing **animations**, for instance of a rotating cube.

```
stilts plot2cube
```

Just displays a unit cube wireframe in a window. You can rotate it with the mouse.

```
stilts plot2cube layer.1=mark in.1=sim.fits x.1=x y.1=y z.1=z
                  shading.1=density densemap.1=pastel
```

Plots markers with x,y,z positions on the screen. You can rotate, zoom and pan the cube on the window this produces. Density shading is used, which means you can see the lines of sight along which most objects fall, though single points are still visible. Density shading is usually a good choice if there is just one dataset, though it can get confusing with more than one.

```
stilts plot2cube in=gavo_g2.fits
                  x=X y=Y z=Z
                  shading=aux aux=HALOID opaque=2.5 auxmap=red-blue
                  layer_m=mark shape_m=open_circle size_m=2
                  layer_v=xyzvector xdelta_v=velX ydelta_v=velY zdelta_v=velZ
```

Plots points in three dimensions with little arrows representing velocity as well as position markers; layer `_m` draws the markers and layer `_v` draws the arrows. Points and vectors are coloured according to the HALOID data value. The positional coordinates (x, y, z) and the shading options are common to both layers, so they can be specified without a prefix.

B.10 `plot2sphere`: Draws a sphere plot

`plot2sphere` draws plots in an isotropic 3-dimensional space using spherical polar coordinates. The plotting volume is a cube, which is viewed from the outside and usually bounded by a wire frame annotated by Cartesian coordinates. This viewing cube is not necessarily centered on the coordinate origin.

This plotting geometry is like that used by `plot2cube`, but the coordinate unit size is always the same in the three dimensions, and the coordinates are specified differently.

Positional coordinates are specified as `lon`, `lat`, `r` triples, e.g.:

```
plot2sphere layer1=mark in1=survey.fits lon1=RA lat1=DEC r1=REDSHIFT
```

Content is added to the plot by specifying one or more *plot layers* using the `layerN` parameter. The `N` part is a suffix applied to all the parameters affecting a given layer; any suffix (including the empty string) may be used. Available layers for this plot type are: `mark` (Section 8.3.1), `size` (Section 8.3.2), `sizexy` (Section 8.3.3), `link2` (Section 8.3.7), `mark2` (Section 8.3.8), `label` (Section 8.3.11), `contour` (Section 8.3.12).

B.10.1 Usage

The usage of `plot2sphere` is

```
stilts <stilts-flags> plot2sphere xpix=<int-value> ypix=<int-value>
                                insets=<top>,<left>,<bottom>,<right>
                                omode=swing|out|cgi|discard|auto
                                storage=simple|cache|basic-cache
                                seq=<suffix>[,...] legend=true|false
                                legborder=true|false legopaque=true|false
                                legseq=<suffix>[,...]
                                legpos=<xfrac,yfrac> title=<value>
                                auxmap=inferno|magma|plasma|...
                                auxclip=<lo>,<hi> auxflip=true|false
                                auxquant=<number>
                                auxfunc=log|linear|sqrt|square
                                auxmin=<number> auxmax=<number>
                                auxlabel=<text> auxcrowd=<factor>
                                auxvisible=true|false
                                forcebitmap=true|false compositor=0..1
                                animate=<table> afmt=<in-format>
                                astream=true|false acmd=<cmds>
                                parallel=<int-value> crowd=<number>
                                frame=true|false minor=true|false
                                gridaa=true|false
                                texttype=plain|antialias|latex
                                fontsize=<int-value>
                                fontstyle=standard|serif|mono
                                fontweight=plain|bold|italic|bold_italic
                                cx=<number> cy=<number> cz=<number>
                                scale=<number> theta=<degrees>
                                phi=<degrees> zoom=<factor> xoff=<pixels>
                                yoff=<pixels> zoomfactor=<number>
                                leglabelN=<text>
                                layerN=<layer-type> <layerN-specific-params>
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.plot2.task.SpherePlot2Task`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

acmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the animation control table as specified by parameter `animate`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

afmt = <in-format> (*String*)

Specifies the format of the animation control table as specified by parameter `animate`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

animate = <table> (*StarTable*)

If not null, this parameter causes the command to create a sequence of plots instead of just one. The parameter value is a table with one row for each frame to be produced. Columns in the table are interpreted as parameters which may take different values for each frame; the column name is the parameter name, and the value for a given frame is its value from that row. Animating like this is considerably more efficient than invoking the `STILTS` command in a loop.

The location of the animation control table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `afmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

astream = `true|false` (*Boolean*)

If set true, the animation control table specified by the `animate` parameter will be read as a stream. It is necessary to give the `afmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as `VOTable`).

[Default: `false`]

auxclip = <lo>,<hi> (*Subrange*)

Defines a subrange of the colour ramp to be used for Aux shading. The is specified as a (low,high) comma-separated pair of two numbers between 0 and 1.

If the full range 0,1 (the default) is used, the whole range of colours specified by the selected shader will be used. But if, for instance a value of 0,0.5 is given, only those colours at the left hand end of the ramp will be seen.

[Default: 0,1]

auxcrowd = <factor> (*Double*)

Determines how closely the tick marks are spaced on the Aux axis, if visible. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

[Default: 1.0]

auxflip = true|false (*Boolean*)

If true, the colour map on the Aux axis will be reversed.

[Default: false]

auxfunc = log|linear|sqrt|square (*Scaling*)

Defines the way that values in the Aux range are mapped to the selected colour ramp.

The available options are:

- log: Logarithmic scaling
- linear: Linear scaling
- sqrt: Square root scaling
- square: Square scaling

[Default: linear]

auxlabel = <text> (*String*)

Sets the label used to annotate the aux axis, if it is visible.

auxmap = inferno|magma|plasma|... (*Shader*)

Color map used for Aux axis shading.

A mixed bag of colour ramps are available: inferno, magma, plasma, viridis, cubehelix, sron, rainbow, rainbow2, rainbow3, pastel, accent, gnuplot, gnuplot2, specxby, set1, paired, hotcold, rdbu, piyg, brbg, cyan-magenta, red-blue, brg, heat, cold, light, greyscale, colour, standard, bugn, bupu, orrd, pubu, purd, huecl, hue, intensity, rgb_red, rgb_green, rgb_blue, hsv_h, hsv_s, hsv_v, yuv_y, yuv_u, yuv_v, scale_hsv_s, scale_hsv_v, scale_yuv_y, mask, blacker, whiter, transparency. *Note:* many of these, including rainbow-like ones, are frowned upon by the visualisation community.

[Default: inferno]

auxmax = <number> (*Double*)

Maximum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

auxmin = <number> (*Double*)

Minimum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

auxquant = <number> (*Double*)

Allows the colour map used for the Aux axis to be quantised. If an integer value N is chosen then the colour map will be viewed as N discrete evenly-spaced levels, so that only N different colours will appear in the plot. This can be used to generate a contour-like effect, and may make it easier to trace the boundaries of regions of interest by eye.

If left blank, the colour map is nominally continuous (though in practice it may be quantised to a medium-sized number like 256).

auxvisible = true|false (*Boolean*)

Determines whether the aux axis colour ramp is displayed alongside the plot.

If not supplied (the default), the aux axis will be visible when aux shading is used in any of the plotted layers.

compositor = 0..1 (*Compositor*)

Defines how multiple overplotted partially transparent pixels are combined to form a resulting colour. The way this is used depends on the details of the specified plot.

Currently, this parameter takes a "boost" value in the range 0..1. If the value is zero, saturation semantics are used: RGB colours are added in proportion to their associated alpha value until the total alpha is saturated (reaches 1), after which additional pixels have no further effect. For larger boost values, the effect is similar, but any non-zero alpha in the output is boosted to the given minimum value. The effect of this is that even very slightly populated pixels can be visually distinguished from unpopulated ones which may not be the case for saturation composition.

[Default: 0.05]

crowd = <number> (*Double*)

Determines how closely tick marks are spaced on the wire frame axes. The default value is 1, meaning normal crowding. Larger values result in more grid lines, and smaller values in fewer grid lines.

[Default: 1]

cx = <number> (*Double*)

Gives the central coordinate in the X dimension. This will be determined from the data range if not supplied.

cy = <number> (*Double*)

Gives the central coordinate in the Y dimension. This will be determined from the data range if not supplied.

cz = <number> (*Double*)

Gives the central coordinate in the Z dimension. This will be determined from the data range if not supplied.

fontsize = <int-value> (*Integer*)

Size of the text font in points.

[Default: 12]

fontstyle = standard|serif|mono (*FontType*)

Font style for text.

The available options are:

- standard
- serif
- mono

[Default: standard]

fontweight = plain|bold|italic|bold_italic (*FontWeight*)

Font weight for text.

The available options are:

- plain
- bold
- italic
- bold_italic

[Default: plain]

forcebitmap = true|false (*Boolean*)

This option only has an effect when writing output to vector graphics formats (PDF and PostScript). If set `true`, the data contents of the plot are drawn as a pixel map embedded into

the output file rather than plotting each point in the output. This may make the output less beautiful (round markers will no longer be perfectly round), but it may result in a much smaller file if there are very many data points. Plot annotations such as axis labels will not be affected - they are still drawn as vector text. Note that in some cases (e.g. `shadingN=auto` or `shadingN=density`) this kind of pixellisation will happen in any case.

[Default: `false`]

frame = true|false (*Boolean*)

If true, a cube wire frame with labelled axes is drawn to indicate the limits of the plotted 3D region. If false, no wire frame and no axes are drawn.

[Default: `true`]

gridaa = true|false (*Boolean*)

If true, grid lines are drawn with antialiasing. Antialiased lines look smoother, but may take perceptibly longer to draw. Only has any effect for bitmapped output formats.

[Default: `false`]

insets = <top>,<left>,<bottom>,<right> (*Insets*)

Defines the amount of space in pixels around the actual plotting area. This space is used for axis labels, and other decorations and any left over forms an empty border.

The size and position of the actual plotting area is determined by this parameter along with `xpix` and `ypix`. If no value is set (the default), the insets will be determined automatically according to how much space is required for labels etc.

layerN = <layer-type> <layerN-specific-params> (*LayerType*)

Selects one of the available plot types for layerN. A plot consists of a plotting surface, set up using the various unsuffixed parameters of the plotting command, and zero or more plot layers. Each layer is introduced by a parameter with the name `layer<N>` where the suffix "`<N>`" is a label identifying the layer and is appended to all the parameter names which configure that layer. Suffixes may be any string, including the empty string.

This parameter may take one of the following values, described in more detail in Section 8.3:

- `mark` (Section 8.3.1)
- `size` (Section 8.3.2)
- `sizexy` (Section 8.3.3)
- `link2` (Section 8.3.7)
- `mark2` (Section 8.3.8)
- `label` (Section 8.3.11)
- `contour` (Section 8.3.12)

Each of these layer types comes with a list of type-specific parameters to define the details of that layer, including some or all of the following groups:

- input table parameters (e.g. `inN`, `icmdN`)
- coordinate params referring to input table columns (e.g. `xN`, `yN`)
- layer style parameters (e.g. `shadingN`, `colorN`)

Every parameter notionally carries the same suffix `N`. However, if the suffix is not present, the application will try looking for a parameter with the same name with no suffix instead. In this way, if several layers have the same value for a given parameter (for instance input table), you can supply it using one unsuffixed parameter to save having to supply several parameters with the same value but different suffixes.

legborder = true|false (*Boolean*)

If true, a line border is drawn around the legend.

[Default: `true`]

legend = true|false (*Boolean*)

Whether to draw a legend or not. If no value is supplied, the decision is made automatically: a legend is drawn only if it would have more than one entry.

leglabelN = <text> (*String*)

Sets the presentation label for the layer with a given suffix. This is the text which is displayed in the legend, if present. Multiple layers may use the same label, in which case they will be combined to form a single legend entry.

If no value is supplied (the default), the suffix itself is used as the label.

legopaque = true|false (*Boolean*)

If true, the background of the legend is opaque, and the legend obscures any plot components behind it. Otherwise, it's transparent.

[Default: true]

legpos = <xfrac,yfrac> (*double[]*)

Determines the internal position of the legend on the plot. The value is a comma-separated pair of values giving the X and Y positions of the legend within the plotting bounds, so for instance "0.5,0.5" will put the legend right in the middle of the plot. If no value is supplied, the legend will appear outside the plot boundary.

legseq = <suffix>[,...] (*String[]*)

Determines which layers are represented in the legend (if present) and in which order they appear. The legend has a line for each layer label (as determined by the `leglabelN` parameter). If multiple layers have the same label, they will contribute to the same entry in the legend, with style icons plotted over each other. The value of this parameter is a sequence of layer suffixes, which determines the order in which the legend entries appear. Layers with suffixes missing from this list do not show up in the legend at all.

If no value is supplied (the default), the sequence is the same as the layer plotting sequence (see `seq`).

minor = true|false (*Boolean*)

If true, minor tick marks are painted along the axes as well as the major tick marks. Minor tick marks do not have associated grid lines.

[Default: true]

omode = swing|out|cgi|discard|auto (*PaintMode*)

Determines how the drawn plot will be output, see Section 8.5.

- `swing`: Plot will be displayed in a window on the screen. This plot is "live"; it can be resized and (except for old-style plots) navigated around with mouse actions in the same way as plots in TOPCAT.
- `out`: Plot will be written to a file given by `out` using the graphics format given by `ofmt`.
- `cgi`: Plot will be written in a way suitable for CGI use direct from a web server. The output is in the graphics format given by `ofmt`, preceded by a suitable "Content-type" declaration.
- `discard`: Plot is drawn, but discarded. There is no output.
- `auto`: Behaves as `swing` or `out` mode depending on presence of `out` parameter

[Default: auto]

parallel = <int-value> (*Integer*)

Determines how many threads will run in parallel if animation output is being produced. Only used if the `animate` parameter is supplied. The default value is the number of processors apparently available to the JVM.

[Default: 8]

phi = <degrees> (*Double*)

Rotation around the Z axis of the plotted 3d space applied before the plot is viewed.

[Default: -30]

scale = <number> (Double)

The length of the cube sides in data coordinates. This will be determined from the data range if not supplied.

seq = <suffix>[,...] (String[])

Contains a comma-separated list of layer suffixes to determine the order in which layers are drawn on the plot. This can affect which symbol are plotted on top of, and so potentially obscure, which other ones.

When specifying a plot, multiple layers may be specified, each introduced by a parameter `layer<N>`, where `<N>` is a different (arbitrary) suffix labelling the layer, and is appended to all the parameters specific to defining that layer.

By default the layers are drawn on the plot in the order in which the `layer*` parameters appear on the command line. However if this parameter is specified, each comma-separated element is interpreted as a layer suffix, giving the ordered list of layers to plot. Every element of the list must be a suffix with a corresponding `layer` parameter, but missing or repeated elements are allowed.

storage = simple|cache|basic-cache (DataStoreFactory)

Determines the way that data is accessed when constructing the plot. There are two basic options, cached or not.

If no caching is used (`simple`) then rows are read sequentially from the specified input table(s) every time they are required. This generally requires a small memory footprint (though that can depend on how the table is specified) and makes sense if the data only needs to be scanned once or perhaps if the table is very large.

If caching is used (`cache`) then the required data is read once from the specified input table(s) and cached before any plotting is performed, and plots are done using this cached data. This may use a significant amount of memory for large tables but it's usually more sensible (faster) if the data will need to be scanned multiple times.

The default value is `cache` if a live plot is being generated (`omode=swing`), since in that case the plot needs to be redrawn every time the user performs plot navigation actions or resizes the window, or if animations are being produced. Otherwise (e.g. output to a graphics file) the default is `simple`.

[Default: simple]

texttype = plain|antialias|latex (TextSyntax)

Determines how to turn label text into characters on the plot. `Plain` and `Antialias` both take the text at face value, but `Antialias` smooths the characters. `LaTeX` interprets the text as LaTeX source code and typesets it accordingly.

When not using LaTeX, antialiased text usually looks nicer, but can be perceptibly slower to plot. At time of writing, on MacOS antialiased text seems to be required to stop the writing coming out upside-down for non-horizontal text (MacOS java bug).

[Default: plain]

theta = <degrees> (Double)

Rotation towards the viewer in degrees of the plotted 3d space.

[Default: 15]

title = <value> (String)

Text of a title to be displayed at the top of the plot. If null, the default, no title is shown and there's more space for the graphics.

xoff = <pixels> (*Double*)

Shifts the whole plot within the plotting region by the given number of pixels in the horizontal direction.

[Default: 0]

xpix = <int-value> (*Integer*)

Size of the output image in the X direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also *insets*.

[Default: 500]

yoff = <pixels> (*Double*)

Shifts the whole plot within the plotting region by the given number of pixels in the vertical direction.

[Default: 0]

ypix = <int-value> (*Integer*)

Size of the output image in the Y direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also *insets*.

[Default: 400]

zoom = <factor> (*Double*)

Sets the magnification factor at which the the plotted 3D region itself is viewed, without affecting its contents. The default value is 1, which means the cube fits into the plotting space however it is rotated. Much higher zoom factors will result in parts of the plotting region and axes being drawn outside of the plotting region (so invisible).

[Default: 1]

zoomfactor = <number> (*Double*)

Sets the amount by which the plot view zooms in or out for each unit of mouse wheel movement. A value of 1 means that mouse wheel zooming has no effect. A higher value means that the mouse wheel zooms faster and a value nearer 1 means it zooms slower. Values below 1 are not permitted.

[Default: 1.2]

B.10.2 Examples

Some examples of `plot2cube` are shown below. See Section 8.1.3 for some examples of producing **animations**, for instance of a rotating cube.

```
stilts plot2sphere in=hip_main.fits lon=radeg lat=dedeg r=plx
                  layer1=mark shading1=density densemap1=cyan-magenta
```

Plots points with RA, Dec and parallax coordinates in 3D. Density shading is used, which means you can see the lines of sight along which most objects fall, though single points are still visible. Density shading is usually a good choice if there is just one dataset, though it can get confusing with more than one.

```
stilts plot2sphere in=hip_main.fits lon=radeg lat=dedeg r=plx
                  layer1=mark shading1=density densemap1=cyan-magenta
                  cx=0 cy=0 cz=0 scale=38 texttype=antialias gridaa=true
```

The same as the previous example but with some more configuration of the axes. The data origin is placed at the centre of the visible cube (this is the position around which the cube will rotate when you drag the mouse), and the size of the cube sides in data coordinates is set explicitly.

B.11 plot2time: Draws a time plot

`plot2time` draws plots where the horizontal axis represents time. The time axis can be labelled in various different ways including MJD, decimal year and ISO-8601 form.

Positional coordinates are specified as t, y pairs, e.g.:

```
plot2time in1=series.cdf layer1=line t1=EPOCH y1=ENERGY
```

This command, unlike the other `plot2*` commands at time of writing, can be used to draw *multi-zone* plots. These are plots with different panels stacked vertically so that different datasets can share the same horizontal (time) axis, but have separate vertical axes, colour maps, legends etc. The horizontal axes are always synchronized between zones. This is currently controlled with the `zoneN` parameter. For any layer with a layer suffix N , you can specify a zone identifier as an arbitrary string, z , by supplying the parameter `zoneN=z`. Layers with the same value of `zoneN` are plotted in the same zone, and layers with different values are plotted in different zones. If no `zoneN` is given, the layer is assigned to a single (unnamed) zone, so with no zone parameters specified all plots appear in a single zone. Parameters specific to a given zone can then be suffixed with the same z zone identifier. The examples section illustrates what this looks like in practice.

Note: this plot type, and the multi-zone feature, is experimental. As currently implemented it lacks some important features. The interface may be changed in a future version.

Content is added to the plot by specifying one or more *plot layers* using the `layerN` parameter. The N part is a suffix applied to all the parameters affecting a given layer; any suffix (including the empty string) may be used. Available layers for this plot type are: `mark` (Section 8.3.1), `line` (Section 8.3.9), `fill` (Section 8.3.14), `yerror` (Section 8.3.25), `spectrogram` (Section 8.3.26), `label` (Section 8.3.11), `function` (Section 8.3.19).

B.11.1 Usage

The usage of `plot2time` is

```
stilts <stilts-flags> plot2time xpix=<int-value> ypix=<int-value>
insets=<top>,<left>,<bottom>,<right>
omode=swing|out|cgi|discard|auto
storage=simple|cache|basic-cache
seq=<suffix>[,...] legend=true|false
legborder=true|false legopaque=true|false
legseq=<suffix>[,...] legpos=<xfrac,yfrac>
title=<value>
auxmap=inferno|magma|plasma|...
auxclip=<lo>,<hi> auxflip=true|false
auxquant=<number>
auxfunc=log|linear|sqrt|square
auxmin=<number> auxmax=<number>
auxlabel=<text> auxcrowd=<factor>
auxvisible=true|false
forcebitmap=true|false compositor=0..1
animate=<table> afmt=<in-format>
astream=true|false acmd=<cmds>
parallel=<int-value> ylog=true|false
yflip=true|false tlabel=<text>
ylabel=<text> grid=true|false
tcrowd=<number> ycrowd=<number>
tformat=iso-8601|year|mjd|unix
minor=true|false
texttype=plain|antialias|latex
fontsize=<int-value>
fontstyle=standard|serif|mono
fontweight=plain|bold|italic|bold_italic
```

```

tmin=<year-or-iso8601>
tmax=<year-or-iso8601> tsub=<lo>,<hi>
ymin=<number> ymax=<number> ysub=<lo>,<hi>
navaxes=t|y|ty zoomfactor=<number>
leglabelN=<text>
layerN=<layer-type> <layerN-specific-params>
zoneN=<text>

```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.plot2.task.TimePlot2Task`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

acmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the animation control table as specified by parameter `animate`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of `"@filename"` causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

afmt = <in-format> (*String*)

Specifies the format of the animation control table as specified by parameter `animate`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

animate = <table> (*StarTable*)

If not null, this parameter causes the command to create a sequence of plots instead of just one. The parameter value is a table with one row for each frame to be produced. Columns in the table are interpreted as parameters which may take different values for each frame; the column name is the parameter name, and the value for a given frame is its value from that row. Animating like this is considerably more efficient than invoking the `STILTS` command in a loop.

The location of the animation control table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `afmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

astream = true|false (*Boolean*)

If set true, the animation control table specified by the `animate` parameter will be read as a

stream. It is necessary to give the `afmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

auxclip = <lo>,<hi> (Subrange)

Defines a subrange of the colour ramp to be used for Aux shading. The is specified as a (low,high) comma-separated pair of two numbers between 0 and 1.

If the full range 0,1 (the default) is used, the whole range of colours specified by the selected shader will be used. But if, for instance a value of 0,0.5 is given, only those colours at the left hand end of the ramp will be seen.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxclipZ` affects only zone `z`.

[Default: 0,1]

auxcrowd = <factor> (Double)

Determines how closely the tick marks are spaced on the Aux axis, if visible. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxcrowdZ` affects only zone `z`.

[Default: 1.0]

auxflip = true|false (Boolean)

If true, the colour map on the Aux axis will be reversed.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxflipZ` affects only zone `z`.

[Default: `false`]

auxfunc = log|linear|sqrt|square (Scaling)

Defines the way that values in the Aux range are mapped to the selected colour ramp.

The available options are:

- `log`: Logarithmic scaling
- `linear`: Linear scaling
- `sqrt`: Square root scaling
- `square`: Square scaling

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxfuncZ` affects only zone `z`.

[Default: `linear`]

auxlabel = <text> (String)

Sets the label used to annotate the aux axis, if it is visible.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxlabelZ` affects only zone `z`.

auxmap = inferno|magma|plasma|... (Shader)

Color map used for Aux axis shading.

A mixed bag of colour ramps are available: `inferno`, `magma`, `plasma`, `viridis`, `cubehelix`, `sron`, `rainbow`, `rainbow2`, `rainbow3`, `pastel`, `accent`, `gnuplot`, `gnuplot2`, `specxby`, `set1`,

paired, hotcold, rdbu, piyg, brbg, cyan-magenta, red-blue, brg, heat, cold, light, greyscale, colour, standard, bugn, bupu, orrd, pubu, purd, huecl, hue, intensity, rgb_red, rgb_green, rgb_blue, hsv_h, hsv_s, hsv_v, yuv_y, yuv_u, yuv_v, scale_hsv_s, scale_hsv_v, scale_yuv_y, mask, blacker, whiter, transparency. *Note:* many of these, including rainbow-like ones, are frowned upon by the visualisation community.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxmapZ` affects only zone `z`.

[Default: `inferno`]

`auxmax = <number>` (*Double*)

Maximum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxmaxZ` affects only zone `z`.

`auxmin = <number>` (*Double*)

Minimum value of the data coordinate on the Aux axis. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxminZ` affects only zone `z`.

`auxquant = <number>` (*Double*)

Allows the colour map used for the Aux axis to be quantised. If an integer value `N` is chosen then the colour map will be viewed as `N` discrete evenly-spaced levels, so that only `N` different colours will appear in the plot. This can be used to generate a contour-like effect, and may make it easier to trace the boundaries of regions of interest by eye.

If left blank, the colour map is nominally continuous (though in practice it may be quantised to a medium-sized number like 256).

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxquantZ` affects only zone `z`.

`auxvisible = true|false` (*Boolean*)

Determines whether the aux axis colour ramp is displayed alongside the plot.

If not supplied (the default), the aux axis will be visible when aux shading is used in any of the plotted layers.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `auxvisibleZ` affects only zone `z`.

`compositor = 0..1` (*Compositor*)

Defines how multiple overplotted partially transparent pixels are combined to form a resulting colour. The way this is used depends on the details of the specified plot.

Currently, this parameter takes a "boost" value in the range 0..1. If the value is zero, saturation semantics are used: RGB colours are added in proportion to their associated alpha value until the total alpha is saturated (reaches 1), after which additional pixels have no further effect. For larger boost values, the effect is similar, but any non-zero alpha in the output is boosted to the given minimum value. The effect of this is that even very slightly populated pixels can be visually distinguished from unpopulated ones which may not be the case for saturation composition.

[Default: 0.05]

`fontsize = <int-value>` (*Integer*)

Size of the text font in points.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `fontsizeZ`

affects only zone z.

[Default: 12]

fontstyle = standard|serif|mono (*FontType*)

Font style for text.

The available options are:

- standard
- serif
- mono

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. fontstyleZ affects only zone z.

[Default: standard]

fontweight = plain|bold|italic|bold_italic (*FontWeight*)

Font weight for text.

The available options are:

- plain
- bold
- italic
- bold_italic

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. fontweightZ affects only zone z.

[Default: plain]

forcebitmap = true|false (*Boolean*)

This option only has an effect when writing output to vector graphics formats (PDF and PostScript). If set `true`, the data contents of the plot are drawn as a pixel map embedded into the output file rather than plotting each point in the output. This may make the output less beautiful (round markers will no longer be perfectly round), but it may result in a much smaller file if there are very many data points. Plot annotations such as axis labels will not be affected - they are still drawn as vector text. Note that in some cases (e.g. shadingN=auto or shadingN=density) this kind of pixellisation will happen in any case.

[Default: false]

grid = true|false (*Boolean*)

If true, grid lines are drawn on the plot at positions determined by the major tick marks. If false, they are absent.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. gridZ affects only zone z.

[Default: false]

insets = <top>,<left>,<bottom>,<right> (*Insets*)

Defines the amount of space in pixels around the actual plotting area. This space is used for axis labels, and other decorations and any left over forms an empty border.

The size and position of the actual plotting area is determined by this parameter along with `xpix` and `ypix`. If no value is set (the default), the insets will be determined automatically according to how much space is required for labels etc.

layerN = <layer-type> <layerN-specific-params> (*LayerType*)

Selects one of the available plot types for layerN. A plot consists of a plotting surface, set up using the various unsuffixed parameters of the plotting command, and zero or more plot layers. Each layer is introduced by a parameter with the name `layer<N>` where the suffix "<N>" is a

label identifying the layer and is appended to all the parameter names which configure that layer. Suffixes may be any string, including the empty string.

This parameter may take one of the following values, described in more detail in Section 8.3:

- mark (Section 8.3.1)
- line (Section 8.3.9)
- fill (Section 8.3.14)
- yerror (Section 8.3.25)
- spectrogram (Section 8.3.26)
- label (Section 8.3.11)
- function (Section 8.3.19)

Each of these layer types comes with a list of type-specific parameters to define the details of that layer, including some or all of the following groups:

- input table parameters (e.g. `inN`, `icmdN`)
- coordinate params referring to input table columns (e.g. `xN`, `yN`)
- layer style parameters (e.g. `shadingN`, `colorN`)

Every parameter notionally carries the same suffix `N`. However, if the suffix is not present, the application will try looking for a parameter with the same name with no suffix instead. In this way, if several layers have the same value for a given parameter (for instance input table), you can supply it using one unsuffixed parameter to save having to supply several parameters with the same value but different suffixes.

legborder = true|false (*Boolean*)

If true, a line border is drawn around the legend.

[Default: true]

legend = true|false (*Boolean*)

Whether to draw a legend or not. If no value is supplied, the decision is made automatically: a legend is drawn only if it would have more than one entry.

leglabelN = <text> (*String*)

Sets the presentation label for the layer with a given suffix. This is the text which is displayed in the legend, if present. Multiple layers may use the same label, in which case they will be combined to form a single legend entry.

If no value is supplied (the default), the suffix itself is used as the label.

legopaque = true|false (*Boolean*)

If true, the background of the legend is opaque, and the legend obscures any plot components behind it. Otherwise, it's transparent.

[Default: true]

legpos = <xfrac,yfrac> (*double[]*)

Determines the internal position of the legend on the plot. The value is a comma-separated pair of values giving the X and Y positions of the legend within the plotting bounds, so for instance "0.5,0.5" will put the legend right in the middle of the plot. If no value is supplied, the legend will appear outside the plot boundary.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `legposZ` affects only zone `Z`.

legseq = <suffix>[,...] (*String[]*)

Determines which layers are represented in the legend (if present) and in which order they appear. The legend has a line for each layer label (as determined by the `leglabelN` parameter). If multiple layers have the same label, they will contribute to the same entry in the legend, with style icons plotted over each other. The value of this parameter is a sequence of layer suffixes,

which determines the order in which the legend entries appear. Layers with suffixes missing from this list do not show up in the legend at all.

If no value is supplied (the default), the sequence is the same as the layer plotting sequence (see `seq`).

minor = true|false (*Boolean*)

If true, minor tick marks are painted along the axes as well as the major tick marks. Minor tick marks do not have associated grid lines.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `minorZ` affects only zone `z`.

[Default: true]

navaxes = t|y|ty (*boolean[]*)

Determines the axes which are affected by the interactive navigation actions (pan and zoom). The default is `t` which means that the various mouse gestures will provide panning and zooming in the Time direction only. However, if it is set to `ty` mouse actions will affect both the horizontal and vertical axes.

[Default: t]

omode = swing|out|cgi|discard|auto (*PaintMode*)

Determines how the drawn plot will be output, see Section 8.5.

- `swing`: Plot will be displayed in a window on the screen. This plot is "live"; it can be resized and (except for old-style plots) navigated around with mouse actions in the same way as plots in TOPCAT.
- `out`: Plot will be written to a file given by `out` using the graphics format given by `ofmt`.
- `cgi`: Plot will be written in a way suitable for CGI use direct from a web server. The output is in the graphics format given by `ofmt`, preceded by a suitable "Content-type" declaration.
- `discard`: Plot is drawn, but discarded. There is no output.
- `auto`: Behaves as `swing` or `out` mode depending on presence of `out` parameter

[Default: auto]

parallel = <int-value> (*Integer*)

Determines how many threads will run in parallel if animation output is being produced. Only used if the `animate` parameter is supplied. The default value is the number of processors apparently available to the JVM.

[Default: 8]

seq = <suffix>[,...] (*String[]*)

Contains a comma-separated list of layer suffixes to determine the order in which layers are drawn on the plot. This can affect which symbol are plotted on top of, and so potentially obscure, which other ones.

When specifying a plot, multiple layers may be specified, each introduced by a parameter `layer<N>`, where `<N>` is a different (arbitrary) suffix labelling the layer, and is appended to all the parameters specific to defining that layer.

By default the layers are drawn on the plot in the order in which the `layer*` parameters appear on the command line. However if this parameter is specified, each comma-separated element is interpreted as a layer suffix, giving the ordered list of layers to plot. Every element of the list must be a suffix with a corresponding `layer` parameter, but missing or repeated elements are allowed.

storage = simple|cache|basic-cache (*DataStoreFactory*)

Determines the way that data is accessed when constructing the plot. There are two basic options, cached or not.

If no caching is used (`simple`) then rows are read sequentially from the specified input table(s) every time they are required. This generally requires a small memory footprint (though that can depend on how the table is specified) and makes sense if the data only needs to be scanned once or perhaps if the table is very large.

If caching is used (`cache`) then the required data is read once from the specified input table(s) and cached before any plotting is performed, and plots are done using this cached data. This may use a significant amount of memory for large tables but it's usually more sensible (faster) if the data will need to be scanned multiple times.

The default value is `cache` if a live plot is being generated (`omode=swing`), since in that case the plot needs to be redrawn every time the user performs plot navigation actions or resizes the window, or if animations are being produced. Otherwise (e.g. output to a graphics file) the default is `simple`.

[Default: `simple`]

tcrowd = <number> (*Double*)

Determines how closely the tick marks are spaced on the Time axis. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `tcrowdZ` affects only zone `Z`.

[Default: 1]

texttype = `plain|antialias|latex` (*TextSyntax*)

Determines how to turn label text into characters on the plot. `Plain` and `Antialias` both take the text at face value, but `Antialias` smooths the characters. `LaTeX` interprets the text as LaTeX source code and typesets it accordingly.

When not using LaTeX, antialiased text usually looks nicer, but can be perceptibly slower to plot. At time of writing, on MacOS antialiased text seems to be required to stop the writing coming out upside-down for non-horizontal text (MacOS java bug).

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `texttypeZ` affects only zone `Z`.

[Default: `plain`]

tformat = `iso-8601|year|mjd|unix` (*TimeFormat*)

Selects the way in which time values are represented when using them to label the time axis.

The available options are:

- `iso-8601`: ISO 8601 date, of the form `yyyy-mm-ddThh:mm:ss.s` (e.g. "2012-03-13T04")
- `year`: Decimal year (e.g. "2012.197")
- `mjd`: Modified Julian Date (e.g. "55999.2")
- `unix`: Seconds since midnight of 1 Jan 1970 (e.g. "1331613420")

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `tformatZ` affects only zone `Z`.

[Default: `iso-8601`]

title = <value> (*String*)

Text of a title to be displayed at the top of the plot. If null, the default, no title is shown and there's more space for the graphics.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `titleZ` affects only zone `Z`.

tlabel = <text> (*String*)

Gives a label to be used for annotating the Time axis. If not supplied no label will be drawn.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `tlabelZ` affects only zone `z`.

tmax = <year-or-iso8601> (*Double*)

Maximum value of the time coordinate plotted. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

The value may be set with a string that can be interpreted as a decimal year (e.g. "2007.521") or an ISO-8601 string (e.g. "2007-07-10T03:57:36", "2007-07-10T03" or "2007-07-10"). Note however that the numeric value of this configuration item if accessed programmatically is seconds since 1 Jan 1970.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `tmaxZ` affects only zone `z`.

tmin = <year-or-iso8601> (*Double*)

Minimum value of the time coordinate plotted. This sets the value before any subranging is applied. If not supplied, the value is determined from the plotted data.

The value may be set with a string that can be interpreted as a decimal year (e.g. "2007.521") or an ISO-8601 string (e.g. "2007-07-10T03:57:36", "2007-07-10T03" or "2007-07-10"). Note however that the numeric value of this configuration item if accessed programmatically is seconds since 1 Jan 1970.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `tminZ` affects only zone `z`.

tsub = <lo>,<hi> (*Subrange*)

Defines a normalised adjustment to the data range of the Time axis. The value may be specified as a comma-separated pair of two numbers, giving the lower and upper bounds of the range of of interest respectively. This sub-range is applied to the data range that would otherwise be used, either automatically calculated or explicitly supplied; zero corresponds to the lower bound and one to the upper.

The default value "0,1" therefore has no effect. The range could be restricted to its lower half with the value 0,0.5.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `tsubZ` affects only zone `z`.

[Default: 0,1]

xpix = <int-value> (*Integer*)

Size of the output image in the X direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also `insets`.

[Default: 500]

ycrowd = <number> (*Double*)

Determines how closely the tick marks are spaced on the Y axis. The default value is 1, meaning normal crowding. Larger values result in more ticks, and smaller values fewer ticks. Tick marks will not however be spaced so closely that the labels overlap each other, so to get very closely spaced marks you may need to reduce the font size as well.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `ycrowdZ` affects only zone `z`.

[Default: 1]

yflip = true|false (*Boolean*)

If true, the scale on the Y axis will increase in the opposite sense from usual (e.g. right to left

rather than left to right).

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `yflipZ` affects only zone `Z`.

[Default: `false`]

ylabel = <text> (String)

Gives a label to be used for annotating axis Y. A default value based on the plotted data will be used if no value is supplied.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `ylabelZ` affects only zone `Z`.

[Default: `Y`]

ylog = true|false (Boolean)

If `false` (the default), the scale on the Y axis is linear, if `true` it is logarithmic.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `ylogZ` affects only zone `Z`.

[Default: `false`]

ymax = <number> (Double)

Maximum value of the data coordinate on the Y axis. This sets the value before any sub-ranging is applied. If not supplied, the value is determined from the plotted data.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `ymaxZ` affects only zone `Z`.

ymin = <number> (Double)

Minimum value of the data coordinate on the Y axis. This sets the value before any sub-ranging is applied. If not supplied, the value is determined from the plotted data.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `yminZ` affects only zone `Z`.

ypix = <int-value> (Integer)

Size of the output image in the Y direction in pixels. This includes space for any axis labels, padding and other decoration outside the plot area itself. See also `insets`.

[Default: `400`]

ysub = <lo>,<hi> (Subrange)

Defines a normalised adjustment to the data range of the Y axis. The value may be specified as a comma-separated pair of two numbers, giving the lower and upper bounds of the range of interest respectively. This sub-range is applied to the data range that would otherwise be used, either automatically calculated or explicitly supplied; zero corresponds to the lower bound and one to the upper.

The default value `"0,1"` therefore has no effect. The range could be restricted to its lower half with the value `0,0.5`.

If a zone suffix is appended to the parameter name, only that zone is affected, e.g. `ysubZ` affects only zone `Z`.

[Default: `0,1`]

zoneN = <text> (String)

Defines which plot zone the layer with suffix `N` will appear in. This only makes sense for multi-zone plots. The actual value of the parameter is not significant, it just serves as a label, but different layers will end up in the same plot zone if they give the same values for this parameter.

zoomfactor = <number> (Double)

Sets the amount by which the plot view zooms in or out for each unit of mouse wheel movement. A value of 1 means that mouse wheel zooming has no effect. A higher value means that the mouse wheel zooms faster and a value nearer 1 means it zooms slower. Values below 1 are not permitted.

[Default: 1.2]

B.11.2 Examples

Here are some examples of `plot2time`:

```
stilts plot2time xpix=1000 ypix=300
               in=ACE_data.vot t=epoch
               layer.r=line y.r=Br color.r=grey
               layer.t=line y.t=Bt color.t=cyan
               layer.n=line y.n=Bn color.n=pink
```

Three time series are plotted on the same axes as lines in different colours.

```
stilts plot2time xpix=1000 ypix=1000
               in=ACE_data.vot t=epoch
               layer.r=line y.r=Br zone.r=ZR
               layer.t=line y.t=Bt zone.t=ZT
               layer.n=line y.n=Bn zone.n=ZN
               titleZR="Br" titleZT="Bt" titleZN="Bn"
```

The same data is plotted as in the previous example, but in this case each line is drawn in a different panel (zone), stacked vertically. The default colour is used for each line. Each plot is given a different title; note the `title` parameter suffixes refer to the zone identifiers not the layer identifiers.

```
stilts plot2time tmin=2007-06-07T02:40 tmax=2007-06-07T06:20 tformat=mjd
               in=STEREO_STA_L1_SEPT_20070607_V05.cdf t=epoch_ns
               ylabel=Channel
               layer_3=spectrogram spectrum_3=Spec_0_NS
               auxmap=accent auxfunc=log
```

Plots a spectrogram from a CDF file. The range along the horizontal axis is specified explicitly using ISO-8601 date strings, but it is labelled in Modified Julian Date.

```
stilts plot2time in=STEREO_STA_L1_SEPT_20070607_V05.cdf t=epoch_ns
               layer_1=spectrogram spectrum_1=spec_0_ns zone_1=A
               layer_2=spectrogram spectrum_2=spec_0_e zone_2=B
               layer_3=line y_3='mean(spec_0_ns)' color_3=red zone_3=C
               layer_4=line y_4='mean(spec_0_e)' color_4=blue zone_4=C
               ylogC=true
               auxfunc=sqrt auxmapA=viridis auxmapB=magma
```

This is a 3-zone plot; zones A and B each contains a spectrogram (layers `_1` and `_2`), and zone C contains two line plots (layers `_3` and `_4`). The Y axis is set to logarithmic for zone C only. The colour ramps are configured with `aux*` parameters; the stretch function is set to `sqrt` for all zones, and the colour map is set to different values for zones A and B.

B.12 `plot2d`: Old-style 2D Scatter Plot

This section describes a deprecated command. It still works, but you are advised to use the more capable `plot2plane` instead.

`plot2d` performs two-dimensional scatter plots, sending the output to a graphical display or writing it to a file in some vector or bitmapped graphics format. You need to supply it with values for one

or more X and Y datasets, in terms of table columns, and it will generate a plot with a point for each row. There are many options available to configure the detailed appearance of the plot, but in its simplest form invocation is quite straightforward. See Section 9 for more discussion on use of the plotting commands.

B.12.1 Usage

The usage of `plot2d` is

```
stilts <stilts-flags> plot2d xpix=<int-value> ypix=<int-value>
font=dialog|serif|... fontsize=<int-value>
fontstyle=plain|bold|italic|bold-italic
legend=true|false title=<value>
omode=swing|out|cgi|discard|auto
out=<out-file>
ofmt=png|png-transp|gif|jpeg|pdf|eps|eps-gzip
inN=<table> ifmtN=<in-format>
istreamN=true|false cmdN=<cmds> xdataN=<expr>
ydataN=<expr> auxdataN=<expr>
xlo=<float-value> ylo=<float-value>
auxlo=<float-value> xhi=<float-value>
yhi=<float-value> auxhi=<float-value>
xlog=true|false ylog=true|false
auxlog=true|false xflip=true|false
yflip=true|false auxflip=true|false
xlabel=<value> ylabel=<value> auxlabel=<value>
xerrorN=<expr>| [<lo-expr>],[<hi-expr>]
yerrorN=<expr>| [<lo-expr>],[<hi-expr>]
auxshader=rainbow|pastel|... txtlabelN=<value>
subsetNS=<expr> nameNS=<value>
colourNS=<rrggbb>|red|blue|...
shapeNS=filled_circle|open_circle|...
sizeNS=<int-value> transparencyNS=<int-value>
lineNS=DotToDot|LinearRegression
linewidthNS=<int-value>
dashNS=dot|dash|...|<a,b,...>
hideNS=true|false
errstyleNS=lines|capped_lines|...
grid=true|false antialias=true|false
sequence=<suffix>,<suffix>,...
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TablePlot2D`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

antialias = true|false (*Boolean*)

Controls whether lines are drawn using antialiasing, where applicable. If lines are drawn to a bitmapped-type graphics output format setting this parameter to true smooths the lines out by using gradations of colour for diagonal lines, and setting it false simply sets each pixel in the line to on or off. For vector-type graphics output formats, or for cases in which no diagonal lines are drawn, the setting of this parameter has no effect. Setting it true may slow the plot down slightly.

[Default: true]

auxdataN = <expr> (*String*)

Gives a column name or expression for the aux axis data for table N. The expression is a numeric algebraic expression based on column names as described in Section 10

auxflip = true|false (*Boolean*)

If set true, the scale on the aux axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

auxhi = <float-value> (*Double*)

The upper limit for the plotted aux axis. If not set, a value will be chosen which is high enough to accommodate all the data.

auxlabel = <value> (*String*)

Specifies a label to be used for annotating axis aux. A default values based on the plotted data will be used if no value is supplied for this parameter.

auxlo = <float-value> (*Double*)

The lower limit for the plotted aux axis. If not set, a value will be chosen which is low enough to accommodate all the data.

auxlog = true|false (*Boolean*)

If false (the default), the scale on the aux axis is linear; if true it is logarithmic.

[Default: false]

auxshader = rainbow|pastel|... (*Shader*)

Determines how data from auxiliary axes will be displayed. Generally this is some kind of colour ramp. These are the available *colour fixing* options:

- rainbow
- pastel
- standard
- heat
- colour
- hue
- greyscale
- red-blue

and these are the available *colour modifying* options:

- hsv_h
- hsv_s
- hsv_v
- intensity
- rgb_red
- rgb_green
- rgb_blue
- yuv_y
- yuv_u
- yuv_v
- transparency

[Default: rainbow]

cmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the table. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file filename to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

colourNS = <rrggbb>|red|blue|... (*Color*)

Defines the colour of markers plotted. The value may be a 6-digit hexadecimal giving

red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colours. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black and white.

For most purposes, either the American or the British spelling is accepted for this parameter name.

dashNS = dot|dash|...|<a,b,...> (float[])

Defines the dash style for any lines drawn in data set NS To generate a dashed line the value may be one of the named dash types:

- dot
- dash
- longdash
- dotdash

or may be a comma-separated string of on/off length values such as "4,2,8,2". A null value indicates a solid line.

Only has an effect if the lineNS parameter is set to draw lines.

errstyleNS = lines|capped_lines|... (ErrorRenderer)

Defines the way in which error bars (or ellipses, or...) will be represented for data set NS if errors are being displayed. The following options are available:

- none
- lines
- capped_lines
- caps
- arrows
- ellipse
- crosshair_ellipse
- rectangle
- crosshair_rectangle
- filled_ellipse
- filled_rectangle

[Default: lines]

font = dialog|serif|... (String)

Determines the font that will be used for textual annotation of the plot, including axes etc. At least the following fonts will be available:

- serif
- sansserif
- monospaced
- dialog
- dialoginput

as well as a range of system-dependent fonts, possibly including

- abyssinica_sil
- ar_pl_uming_cn
- ar_pl_uming_hk
- ar_pl_uming_tw
- ar_pl_uming_tw_mbe
- bitstream_charter
- caladea
- carlito
- century_schoolbook_l
- cm_roman

- `cm_roman_asian`
- `cm_roman_ce`
- `cm_roman_cyrillic`
- `cm_roman_greek`
- `cm_sans`
- `cm_sans_asian`
- `cm_sans_ce`
- `cm_sans_cyrillic`
- `cm_sans_greek`
- `cm_typewriter`
- `cm_typewriter_asian`
- `cm_typewriter_ce`
- `cm_typewriter_cyrillic`
- `cm_typewriter_greek`
- `...`

[Default: `dialog`]

fontsize = `<int-value>` (*Integer*)

Sets the font size used for plot annotations.

[Default: `12`]

fontstyle = `plain|bold|italic|bold-italic` (*Integer*)

Gives a style in which the font is to be applied for plot annotations. Options are `plain`, `bold`, `italic` and `bold-italic`.

[Default: `plain`]

grid = `true|false` (*Boolean*)

If true, grid lines are drawn on the plot. If false, they are absent.

[Default: `true`]

hideNS = `true|false` (*Boolean*)

Indicates whether the actual markers plotted for each point should be hidden. Normally this is false, but you may want to set it to true if the point positions are being revealed in some other way, for instance by error markers or lines drawn between them.

[Default: `false`]

ifmtN = `<in-format>` (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = `<table>` (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

legend = true|false (*Boolean*)

Determines whether a legend showing which plotting style is used for each data set. Defaults to true if there is more than one set, false otherwise.

lineNS = DotToDot|LinearRegression (*Line*)

Determines what line if any will be plotted along with the data points. The options are:

- `null`: No line is plotted.
- `DotToDot`: Each point is joined to the next one in sequence by a straight line.
- `LinearRegression`: A linear regression line is plotted based on all the points which are visible in the plot. Note that the regression coefficients take no account of points out of the visible range.

linewidthNS = <int-value> (*Integer*)

Sets the line width in pixels for any lines drawn in data set NS.

Only has an effect if the `lineNS` parameter is set to draw lines.

[Default: 1]

nameNS = <value> (*String*)

Provides a name to use for a subset with the symbolic label NS. This name will be used for display in the legend, if one is displayed.

ofmt = png|png-transp|gif|jpeg|pdf|eps|eps-gzip (*GraphicExporter*)

Graphics format in which the plot is written to the output file, see Section 8.6. One of:

- `png`: PNG
- `png-transp`: PNG with transparent background
- `gif`: GIF
- `jpeg`: JPEG
- `pdf`: Portable Document Format
- `eps`: Encapsulated PostScript
- `eps-gzip`: Gzipped Encapsulated PostScript

May default to a sensible value depending on the filename given by `out`.

omode = swing|out|cgi|discard|auto (*PaintMode*)

Determines how the drawn plot will be output, see Section 8.5.

- `swing`: Plot will be displayed in a window on the screen. This plot is "live"; it can be resized and (except for old-style plots) navigated around with mouse actions in the same way as plots in TOPCAT.
- `out`: Plot will be written to a file given by `out` using the graphics format given by `ofmt`.
- `cgi`: Plot will be written in a way suitable for CGI use direct from a web server. The output is in the graphics format given by `ofmt`, preceded by a suitable "Content-type" declaration.
- `discard`: Plot is drawn, but discarded. There is no output.
- `auto`: Behaves as `swing` or `out` mode depending on presence of `out` parameter

[Default: auto]

out = <out-file> (*uk.ac.starlink.util.Destination*)

The location of the output file. This is usually a filename to write to. If it is equal to the special value "-" the output will be written to standard output.

sequence = <suffix>,<suffix>,... (*String[]*)

Can be used to control the sequence in which different datasets and subsets are plotted. This will affect which symbols are plotted on top of, and so potentially obscure, which other ones. The value of this parameter is a comma-separated list of the "NS" suffixes which appear on the parameters which apply to subsets. The sets which are named will be plotted in order, so the first-named one will be at the bottom (most likely to be obscured). Note that if this parameter is supplied, then only those sets which are named will be plotted, so this parameter may also be used to restrict which plots appear (though it may not be the most efficient way of doing this). If no explicit value is supplied for this parameter, sets will be plotted in some sequence decided by STILTS (probably alphabetic by suffix).

shapeNS = filled_circle|open_circle|... (*MarkShape*)

Defines the shapes for the markers that are plotted in data set NS. The following shapes are available:

- filled_circle
- open_circle
- cross
- x
- open_square
- open_diamond
- open_triangle_up
- open_triangle_down
- filled_square
- filled_diamond
- filled_triangle_up
- filled_triangle_down

sizeNS = <int-value> (*Integer*)

Defines the marker size in pixels for markers plotted in data set NS. If the value is negative, an attempt will be made to use a suitable size according to how many points there are to be plotted.

[Default: -1]

subsetNS = <expr> (*String*)

Gives the selection criterion for the subset labelled "NS". This is a boolean expression which may be the name of a boolean-valued column or any other boolean-valued expression. Rows for which the expression evaluates true will be included in the subset, and those for which it evaluates false will not.

title = <value> (*String*)

A one-line title to display at the top of the plot.

transparencyNS = <int-value> (*Integer*)

Determines the transparency of plotted markers for data set NS. A value of <n> means that opacity is only achieved (the background is only blotted out) when <n> pixels of this colour have been plotted on top of each other.

The minimum value is 1, which means opaque markers.

txtlabelN = <value> (*String*)

Gives an expression which will label each plotted point. If given, the text (or number) resulting from evaluating the expression will be written near each point which is plotted.

xdataN = <expr> (String)

Gives a column name or expression for the x axis data for table N. The expression is a numeric algebraic expression based on column names as described in Section 10

xerrorN = <expr>|[<lo-expr>],[<hi-expr>] (String)

Gives expressions for the errors on X coordinates for table N. The following forms are permitted:

- <expr>: symmetric error value
- <lo-expr>,<hi-expr>:distinct lower and upper error values
- <lo-expr>,: lower error value only
- ,<hi-expr>: upper error value only
- null: no errors

The expression in each case is a numeric algebraic expression based on column names as described in Section 10.

xflip = true|false (Boolean)

If set true, the scale on the x axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

xhi = <float-value> (Double)

The upper limit for the plotted x axis. If not set, a value will be chosen which is high enough to accommodate all the data.

xlabel = <value> (String)

Specifies a label to be used for annotating axis x. A default values based on the plotted data will be used if no value is supplied for this parameter.

xlo = <float-value> (Double)

The lower limit for the plotted x axis. If not set, a value will be chosen which is low enough to accommodate all the data.

xlog = true|false (Boolean)

If false (the default), the scale on the x axis is linear; if true it is logarithmic.

[Default: false]

xpix = <int-value> (Integer)

The width of the output graphic in pixels.

[Default: 400]

ydataN = <expr> (String)

Gives a column name or expression for the y axis data for table N. The expression is a numeric algebraic expression based on column names as described in Section 10

yerrorN = <expr>|[<lo-expr>],[<hi-expr>] (String)

Gives expressions for the errors on Y coordinates for table N. The following forms are permitted:

- <expr>: symmetric error value
- <lo-expr>,<hi-expr>:distinct lower and upper error values
- <lo-expr>,: lower error value only
- ,<hi-expr>: upper error value only
- null: no errors

The expression in each case is a numeric algebraic expression based on column names as described in Section 10.

yflip = true|false (Boolean)

If set true, the scale on the y axis will increase in the opposite sense from usual (e.g. right to

left rather than left to right).

[Default: false]

yhi = <float-value> (*Double*)

The upper limit for the plotted y axis. If not set, a value will be chosen which is high enough to accommodate all the data.

ylabel = <value> (*String*)

Specifies a label to be used for annotating axis y. A default values based on the plotted data will be used if no value is supplied for this parameter.

ylo = <float-value> (*Double*)

The lower limit for the plotted y axis. If not set, a value will be chosen which is low enough to accommodate all the data.

ylog = true|false (*Boolean*)

If false (the default), the scale on the y axis is linear; if true it is logarithmic.

[Default: false]

ypix = <int-value> (*Integer*)

The height of the output graphic in pixels.

[Default: 300]

B.12.2 Examples

Here are some examples of `plot2d` in use:

```
stilts plot2d in=cat.xml xdata=RMAG-BMAG ydata=BMAG
```

Plots a colour-magnitude diagram. Since no `omode` or `out` value has been specified, the plot is posted directly to the graphics display for inspection. By adding the parameter `out=xyplot.eps` the plot could be written to an Encapsulated Postscript file instead.

The generated plot is here.

```
stilts plot2d in=6dfgs_mini.xml xdata=RMAG-BMAG ydata=BMAG
subset1=SGFLAG==1 name1=galaxy colour1=blue shapel=open_circle
subset2=SGFLAG==2 name2=star colour2=e010f0 shape2=x size2=3
xlo=-1 xhi=4.5 ylo=10 yhi=20 xpix=500 ypix=250
out=xyplot2.png
```

Plots a colour-magnitude diagram with multiple subsets. The subsets are labelled "1" and "2" with separate sets of parameters applying to each. The selections for the sets are given by the `subset*` parameters; set 1 is those rows with the `SGFLAG` column equal to 1 and set 2 is those rows with the `SGFLAG` column equal to 2. The boundaries of the plot in data coordinates are set explicitly rather than being determined from the data (this is faster) and the plot size in pixels is also set explicitly rather than taking the default values. Output is to a PNG file.

The generated plot is here.

```
stilts plot2d in1=iras_psc.fits cmd1='addskycoords fk5 galactic RA DEC GLON GLAT'
xdata1=GLON ydata1=GLAT
auxdata1=FNU_100 auxlog=true auxflip=true size1=0 transparency1=3
in2=messier.xml cmd2='addskycoords fk5 galactic RA DEC GLON GLAT'
xdata2=GLON ydata2=GLAT
xlabel2=RADIUS>16?("M"+ID):"" cmd2='addcol SIZE sqrt(RADIUS/2)'
xerror2=SIZE yerror2=SIZE
subset2a=true hide2a=true colour2a=black errstyle2a=ellipse
subset2b=true hide2b=true colour2b=black errstyle2b=filled_ellipse
transparency2b=6
xlabel='Galactic Longitude' ylabel='Galactic Latitude' title='The Sky'
```

```

legend=false grid=false fontsize=12 fontstyle=bold-italic
xlo=0 xhi=360 ylo=-90 yhi=+90 xpix=600 ypix=300
out=skyplot.png

```

You can do quite complicated things.

The generated plot is here.

B.13 plot3d: Old-style 3D Scatter Plot

This section describes a deprecated command. It still works, but you are advised to use the more capable `plot2cube` or `plot2sphere` instead.

`plot3d` performs three-dimensional scatter plots, sending the output to a graphical display or writing it to a file in some vector or bitmapped graphics format. You need to supply it with values for one or more X, Y and Z datasets, in terms of table columns, and it will generate a plot with a point for each row. There are many options available to configure the detailed appearance of the plot, but in its simplest form invocation is quite straightforward. See Section 9 for more discussion on use of the plotting commands.

B.13.1 Usage

The usage of `plot3d` is

```

stilts <stilts-flags> plot3d xpix=<int-value> ypix=<int-value>
                             font=dialog|serif|... fontsize=<int-value>
                             fontstyle=plain|bold|italic|bold-italic
                             legend=true|false title=<value>
                             omode=swing|out|cgi|discard|auto
                             out=<out-file>
                             ofmt=png|png-transp|gif|jpeg|pdf|eps|eps-gzip
                             inN=<table> ifmtN=<in-format>
                             istreamN=true|false cmdN=<cmds> xdataN=<expr>
                             ydataN=<expr> zdataN=<expr> auxdataN=<expr>
                             xlo=<float-value> ylo=<float-value>
                             zlo=<float-value> auxlo=<float-value>
                             xhi=<float-value> yhi=<float-value>
                             zhi=<float-value> auxhi=<float-value>
                             xlog=true|false ylog=true|false
                             zlog=true|false auxlog=true|false
                             xflip=true|false yflip=true|false
                             zflip=true|false auxflip=true|false
                             xlabel=<value> ylabel=<value> zlabel=<value>
                             auxlabel=<value>
                             xerrorN=<expr> [<lo-expr>], [<hi-expr>]
                             yerrorN=<expr> [<lo-expr>], [<hi-expr>]
                             zerrorN=<expr> [<lo-expr>], [<hi-expr>]
                             auxshader=rainbow|pastel|... txtlabelN=<value>
                             subsetNS=<expr> nameNS=<value>
                             colourNS=<rrggb>|red|blue|...
                             shapeNS=filled_circle|open_circle|...
                             sizeNS=<int-value> transparencyNS=<int-value>
                             lineNS=DotToDot|LinearRegression
                             linewidthNS=<int-value>
                             dashNS=dot|dash|...|<a,b,...>
                             hideNS=true|false
                             errstyleNS=lines|capped_lines|...
                             grid=true|false antialias=true|false
                             sequence=<suffix>,<suffix>,...
                             fog=<float-value> phi=<float-value>
                             theta=<float-value>

```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TablePlot3D`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

antialias = true|false (*Boolean*)

Controls whether lines are drawn using antialiasing, where applicable. If lines are drawn to a bitmapped-type graphics output format setting this parameter to true smooths the lines out by using gradations of colour for diagonal lines, and setting it false simply sets each pixel in the line to on or off. For vector-type graphics output formats, or for cases in which no diagonal lines are drawn, the setting of this parameter has no effect. Setting it true may slow the plot down slightly.

[Default: true]

auxdataN = <expr> (*String*)

Gives a column name or expression for the aux axis data for table N. The expression is a numeric algebraic expression based on column names as described in Section 10

auxflip = true|false (*Boolean*)

If set true, the scale on the aux axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

auxhi = <float-value> (*Double*)

The upper limit for the plotted aux axis. If not set, a value will be chosen which is high enough to accommodate all the data.

auxlabel = <value> (*String*)

Specifies a label to be used for annotating axis aux. A default values based on the plotted data will be used if no value is supplied for this parameter.

auxlo = <float-value> (*Double*)

The lower limit for the plotted aux axis. If not set, a value will be chosen which is low enough to accommodate all the data.

auxlog = true|false (*Boolean*)

If false (the default), the scale on the aux axis is linear; if true it is logarithmic.

[Default: false]

auxshader = rainbow|pastel|... (*Shader*)

Determines how data from auxiliary axes will be displayed. Generally this is some kind of colour ramp. These are the available *colour fixing* options:

- rainbow
- pastel
- standard
- heat
- colour
- hue
- greyscale
- red-blue

and these are the available *colour modifying* options:

- hsv_h
- hsv_s
- hsv_v
- intensity
- rgb_red
- rgb_green

- `rgb_blue`
- `yuv_y`
- `yuv_u`
- `yuv_v`
- `transparency`

[Default: `rainbow`]

cmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the table. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

colourNS = <rrggb>|red|blue|... (*Color*)

Defines the colour of markers plotted. The value may be a 6-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colours. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black and white.

For most purposes, either the American or the British spelling is accepted for this parameter name.

dashNS = dot|dash|...|<a,b,...> (*float[]*)

Defines the dash style for any lines drawn in data set NS. To generate a dashed line the value may be one of the named dash types:

- `dot`
- `dash`
- `longdash`
- `dotdash`

or may be a comma-separated string of on/off length values such as "4,2,8,2". A `null` value indicates a solid line.

Only has an effect if the `lineNS` parameter is set to draw lines.

errstyleNS = lines|capped_lines|... (*ErrorRenderer*)

Defines the way in which error bars (or ellipses, or...) will be represented for data set NS if errors are being displayed. The following options are available:

- `none`
- `lines`
- `capped_lines`
- `caps`
- `arrows`
- `cuboid`
- `ellipse`
- `crosshair_ellipse`
- `rectangle`
- `crosshair_rectangle`
- `filled_ellipse`
- `filled_rectangle`

[Default: `lines`]

fog = <float-value> (*Double*)

Sets the level of fogging used to provide a visual indication of depth. Object plotted further away from the viewer appear more washed-out by a white fog. The default value gives a bit of fogging; increase it to make the fog thicker, or set to zero if no fogging is required.

[Default: 1.0]

font = dialog|serif|... (*String*)

Determines the font that will be used for textual annotation of the plot, including axes etc. At least the following fonts will be available:

- serif
- sansserif
- monospaced
- dialog
- dialoginput

as well as a range of system-dependent fonts, possibly including

- abyssinica_sil
- ar_pl_uming_cn
- ar_pl_uming_hk
- ar_pl_uming_tw
- ar_pl_uming_tw_mbe
- bitstream_charter
- caladea
- carlito
- century_schoolbook_l
- cm_roman
- cm_roman_asian
- cm_roman_ce
- cm_roman_cyrillic
- cm_roman_greek
- cm_sans
- cm_sans_asian
- cm_sans_ce
- cm_sans_cyrillic
- cm_sans_greek
- cm_typewriter
- cm_typewriter_asian
- cm_typewriter_ce
- cm_typewriter_cyrillic
- cm_typewriter_greek
- ...

[Default: dialog]

fontsize = <int-value> (*Integer*)

Sets the font size used for plot annotations.

[Default: 12]

fontstyle = plain|bold|italic|bold-italic (*Integer*)

Gives a style in which the font is to be applied for plot annotations. Options are plain, bold, italic and bold-italic.

[Default: plain]

grid = true|false (*Boolean*)

If true, grid lines are drawn on the plot. If false, they are absent.

[Default: true]

hideNS = true|false (*Boolean*)

Indicates whether the actual markers plotted for each point should be hidden. Normally this is false, but you may want to set it to true if the point positions are being revealed in some other way, for instance by error markers or lines drawn between them.

[Default: false]

ifmtN = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (*Boolean*)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

legend = true|false (*Boolean*)

Determines whether a legend showing which plotting style is used for each data set. Defaults to true if there is more than one set, false otherwise.

lineNS = DotToDot|LinearRegression (*Line*)

Determines what line if any will be plotted along with the data points. The options are:

- `null`: No line is plotted.
- `DotToDot`: Each point is joined to the next one in sequence by a straight line.
- `LinearRegression`: A linear regression line is plotted based on all the points which are visible in the plot. Note that the regression coefficients take no account of points out of the visible range.

linewidthNS = <int-value> (*Integer*)

Sets the line width in pixels for any lines drawn in data set NS.

Only has an effect if the `lineNS` parameter is set to draw lines.

[Default: 1]

nameNS = <value> (*String*)

Provides a name to use for a subset with the symbolic label NS. This name will be used for display in the legend, if one is displayed.

ofmt = png|png-transp|gif|jpeg|pdf|eps|eps-gzip (*GraphicExporter*)

Graphics format in which the plot is written to the output file, see Section 8.6. One of:

- png: PNG
- png-transp: PNG with transparent background
- gif: GIF
- jpeg: JPEG
- pdf: Portable Document Format
- eps: Encapsulated PostScript
- eps-gzip: Gzipped Encapsulated PostScript

May default to a sensible value depending on the filename given by **out**.

omode = swing|out|cgi|discard|auto (*PaintMode*)

Determines how the drawn plot will be output, see Section 8.5.

- swing: Plot will be displayed in a window on the screen. This plot is "live"; it can be resized and (except for old-style plots) navigated around with mouse actions in the same way as plots in TOPCAT.
- out: Plot will be written to a file given by **out** using the graphics format given by **ofmt**.
- cgi: Plot will be written in a way suitable for CGI use direct from a web server. The output is in the graphics format given by **ofmt**, preceded by a suitable "Content-type" declaration.
- discard: Plot is drawn, but discarded. There is no output.
- auto: Behaves as swing or out mode depending on presence of **out** parameter

[Default: auto]

out = <out-file> (*uk.ac.starlink.util.Destination*)

The location of the output file. This is usually a filename to write to. If it is equal to the special value "-" the output will be written to standard output.

phi = <float-value> (*Double*)

Angle in degrees through which the 3D plot is rotated around the Z axis prior to drawing.

[Default: 30.0]

sequence = <suffix>,<suffix>,... (*String[]*)

Can be used to control the sequence in which different datasets and subsets are plotted. This will affect which symbols are plotted on top of, and so potentially obscure, which other ones. The value of this parameter is a comma-separated list of the "NS" suffixes which appear on the parameters which apply to subsets. The sets which are named will be plotted in order, so the first-named one will be at the bottom (most likely to be obscured). Note that if this parameter is supplied, then only those sets which are named will be plotted, so this parameter may also be used to restrict which plots appear (though it may not be the most efficient way of doing this). If no explicit value is supplied for this parameter, sets will be plotted in some sequence decided by STILTS (probably alphabetic by suffix).

shapeNS = filled_circle|open_circle|... (*MarkShape*)

Defines the shapes for the markers that are plotted in data set NS. The following shapes are available:

- filled_circle
- open_circle
- cross
- x
- open_square

- open_diamond
- open_triangle_up
- open_triangle_down
- filled_square
- filled_diamond
- filled_triangle_up
- filled_triangle_down

sizeNS = <int-value> *(Integer)*

Defines the marker size in pixels for markers plotted in data set NS. If the value is negative, an attempt will be made to use a suitable size according to how many points there are to be plotted.

[Default: -1]

subsetNS = <expr> *(String)*

Gives the selection criterion for the subset labelled "NS". This is a boolean expression which may be the name of a boolean-valued column or any other boolean-valued expression. Rows for which the expression evaluates true will be included in the subset, and those for which it evaluates false will not.

theta = <float-value> *(Double)*

Angle in degrees through which the 3D plot is rotated towards the viewer (i.e. about the horizontal axis of the viewing plane) prior to drawing.

[Default: 15.0]

title = <value> *(String)*

A one-line title to display at the top of the plot.

transparencyNS = <int-value> *(Integer)*

Determines the transparency of plotted markers for data set NS. A value of <n> means that opacity is only achieved (the background is only blotted out) when <n> pixels of this colour have been plotted on top of each other.

The minimum value is 1, which means opaque markers.

txtlabelN = <value> *(String)*

Gives an expression which will label each plotted point. If given, the text (or number) resulting from evaluating the expression will be written near each point which is plotted.

xdataN = <expr> *(String)*

Gives a column name or expression for the x axis data for table N. The expression is a numeric algebraic expression based on column names as described in Section 10

xerrorN = <expr>|[<lo-expr>],[<hi-expr>] *(String)*

Gives expressions for the errors on X coordinates for table N. The following forms are permitted:

- <expr>: symmetric error value
- <lo-expr>,<hi-expr>: distinct lower and upper error values
- <lo-expr>,: lower error value only
- ,<hi-expr>: upper error value only
- null: no errors

The expression in each case is a numeric algebraic expression based on column names as described in Section 10.

xflip = true|false *(Boolean)*

If set true, the scale on the x axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

xhi = <float-value> (*Double*)

The upper limit for the plotted x axis. If not set, a value will be chosen which is high enough to accommodate all the data.

xlabel = <value> (*String*)

Specifies a label to be used for annotating axis x. A default values based on the plotted data will be used if no value is supplied for this parameter.

xlo = <float-value> (*Double*)

The lower limit for the plotted x axis. If not set, a value will be chosen which is low enough to accommodate all the data.

xlog = true|false (*Boolean*)

If false (the default), the scale on the x axis is linear; if true it is logarithmic.

[Default: false]

xpix = <int-value> (*Integer*)

The width of the output graphic in pixels.

[Default: 300]

ydataN = <expr> (*String*)

Gives a column name or expression for the y axis data for table N. The expression is a numeric algebraic expression based on column names as described in Section 10

yerrorN = <expr>|[<lo-expr>],[<hi-expr>] (*String*)

Gives expressions for the errors on Y coordinates for table N. The following forms are permitted:

- <expr>: symmetric error value
- <lo-expr>,<hi-expr>: distinct lower and upper error values
- <lo-expr>,: lower error value only
- ,<hi-expr>: upper error value only
- null: no errors

The expression in each case is a numeric algebraic expression based on column names as described in Section 10.

yflip = true|false (*Boolean*)

If set true, the scale on the y axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

yhi = <float-value> (*Double*)

The upper limit for the plotted y axis. If not set, a value will be chosen which is high enough to accommodate all the data.

ylabel = <value> (*String*)

Specifies a label to be used for annotating axis y. A default values based on the plotted data will be used if no value is supplied for this parameter.

ylo = <float-value> (*Double*)

The lower limit for the plotted y axis. If not set, a value will be chosen which is low enough to accommodate all the data.

ylog = true|false (*Boolean*)

If false (the default), the scale on the y axis is linear; if true it is logarithmic.

[Default: false]

ypix = <int-value> (*Integer*)

The height of the output graphic in pixels.

[Default: 300]

zdataN = <expr> (*String*)

Gives a column name or expression for the z axis data for table N. The expression is a numeric algebraic expression based on column names as described in Section 10

zerrorN = <expr>| [<lo-expr>], [<hi-expr>] (*String*)

Gives expressions for the errors on Z coordinates for table N. The following forms are permitted:

- <expr>: symmetric error value
- <lo-expr>, <hi-expr>: distinct lower and upper error values
- <lo-expr>, : lower error value only
- , <hi-expr>: upper error value only
- null: no errors

The expression in each case is a numeric algebraic expression based on column names as described in Section 10.

zflip = true|false (*Boolean*)

If set true, the scale on the z axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: false]

zhi = <float-value> (*Double*)

The upper limit for the plotted z axis. If not set, a value will be chosen which is high enough to accommodate all the data.

zlabel = <value> (*String*)

Specifies a label to be used for annotating axis z. A default values based on the plotted data will be used if no value is supplied for this parameter.

zlo = <float-value> (*Double*)

The lower limit for the plotted z axis. If not set, a value will be chosen which is low enough to accommodate all the data.

zlog = true|false (*Boolean*)

If false (the default), the scale on the z axis is linear; if true it is logarithmic.

[Default: false]

B.13.2 Examples

Here are some examples of `plot3d` in use:

```
stilts plot3d in=cat.xml xdata=RMAG ydata=BMAG zdata=VEL zlog=true
```

Plots a 3-d scatter plot of red magnitude vs. blue magnitude vs. velocity; the velocity is plotted on a logarithmic scale. Since no `omode` or `out` value has been specified, the plot is posted directly to the graphics display for inspection. By adding the parameter `out=xyplot.eps` the plot could be written to an Encapsulated Postscript file instead.

The generated plot is here.

```
stilts plot3d in=sim1.fits xdata=x ydata=y zdata=z
               cmd='addcol vel "sqrt(velx*velx+vely*vely+velz*velz)"' auxdata=vel auxlog=true
               xpix=500 ypix=400 phi=50 theta=10 out=cube.jpeg
```

Plots the x, y, z positions of particles from a file containing the result of a simulation run. Here an auxiliary axis is used to colour-code the points according their velocity. This is done by introducing a new `vel` column to the table using the `addcol` filter command, so that the `vel` column can be used as the value for the `auxdata` parameter. Alternatively, the given expression

for the velocity could have been used directly as the value of the `auxdata` parameter. Additionally, the `phi` and `theta` parameters are given to adjust the orientation of the cube. The generated plot is here.

B.14 `plothist`: Old-style Histogram

This section describes a deprecated command. It still works, but you are advised to use the more capable `plot2plane` instead.

`plothist` performs histogram plots, sending the output to a graphical display or writing it to a file in some vector or bitmapped graphics format. You need to supply it with values for one or more sets of X values, in terms of table columns, and it will bin the data and draw bars appropriately. Plot bounds, bin widths etc may be supplied explicitly, but will be calculated from the data and set from defaults as appropriate otherwise. There are many options available to configure the detailed appearance of the plot, but in its simplest form invocation is quite straightforward. See Section 9 for more discussion on use of the plotting commands.

B.14.1 Usage

The usage of `plothist` is

```
stilts <stilts-flags> plothist xpix=<int-value> ypix=<int-value>
                                font=dialog|serif|... fontsize=<int-value>
                                fontstyle=plain|bold|italic|bold-italic
                                legend=true|false title=<value>
                                omode=swing|out|cgi|discard|auto
                                out=<out-file>
                                ofmt=png|png-transp|gif|jpeg|pdf|eps|eps-gzip
                                inN=<table> ifmtN=<in-format>
                                istreamN=true|false cmdN=<cmds>
                                xdataN=<expr> xlo=<float-value>
                                xhi=<float-value> xlog=true|false
                                xflip=true|false xlabel=<value>
                                subsetNS=<expr> nameNS=<value>
                                colourNS=<rrggb>|red|blue|...
                                barstyleNS=fill|open|...
                                linewidthNS=<int-value>
                                dashNS=dot|dash|...|<a,b,...>
                                grid=true|false antialias=true|false
                                sequence=<suffix>,<suffix>,...
                                ylo=<float-value> yhi=<float-value>
                                ylog=true|false ylabel=<value>
                                weightN=<value> binwidth=<float-value>
                                norm=true|false cumulative=true|false
                                binbase=<float-value>
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TableHistogram`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`antialias = true|false` (Boolean)

Controls whether lines are drawn using antialiasing, where applicable. If lines are drawn to a bitmapped-type graphics output format setting this parameter to true smooths the lines out by using gradations of colour for diagonal lines, and setting it false simply sets each pixel in the line to on or off. For vector-type graphics output formats, or for cases in which no diagonal lines are drawn, the setting of this parameter has no effect. Setting it true may slow the plot down slightly.

[Default: true]

barstyleNS = fill|open|... (BarShape)

Defines how histogram bars will be drawn for dataset NS. The options are:

- fill
- open
- tops
- semi
- semitops
- spikes
- fillover
- openover

[Default: fill]

binbase = <float-value> (Double)

Adjusts the offset of the bins. By default zero (or one for logarithmic X axis) is a boundary between bins; other boundaries are defined by this and the bin width. If this value is adjusted, the lower bound of one of the bins will be set to this value, so all the bins move along by the corresponding distance.

[Default: 0.0]

binwidth = <float-value> (Double)

Defines the width on the X axis of histogram bins. If the X axis is logarithmic, then this is a multiplicative value.

cmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on the table. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file filename to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

colourNS = <rrggbb>|red|blue|... (Color)

Defines the colour of bars plotted for data set NS. The value may be a 6-digit hexadecimal number giving red, green and blue intensities, e.g. "ff00ff" for magenta. Alternatively it may be the name of one of the pre-defined colours. These are currently red, blue, green, grey, magenta, cyan, orange, pink, yellow, black and white.

For most purposes, either the American or the British spelling is accepted for this parameter name.

cumulative = true|false (Boolean)

Determines whether histograms are cumulative. When false (the default), the height of each bar is determined by counting the number of points which fall into the range on the X axis that it covers. When true, the height is determined by counting all the points between negative infinity and the upper bound of the range on the X axis that it covers.

[Default: false]

dashNS = dot|dash|...|<a,b,...> (float[])

Defines the dashing pattern for lines drawn for dataset NS. To generate a dashed line the value may be one of the named dash types:

- dot

- dash
- longdash
- dotdash

or may be a comma-separated string of on/off length values such as "4,2,8,2". A `null` value indicates a solid line. Only certain bar styles are affected by the dash pattern.

font = `dialog|serif|...` (*String*)

Determines the font that will be used for textual annotation of the plot, including axes etc. At least the following fonts will be available:

- serif
- sansserif
- monospaced
- dialog
- dialoginput

as well as a range of system-dependent fonts, possibly including

- abyssinica_sil
- ar_pl_uming_cn
- ar_pl_uming_hk
- ar_pl_uming_tw
- ar_pl_uming_tw_mbe
- bitstream_charter
- caladea
- carlito
- century_schoolbook_l
- cm_roman
- cm_roman_asian
- cm_roman_ce
- cm_roman_cyrillic
- cm_roman_greek
- cm_sans
- cm_sans_asian
- cm_sans_ce
- cm_sans_cyrillic
- cm_sans_greek
- cm_typewriter
- cm_typewriter_asian
- cm_typewriter_ce
- cm_typewriter_cyrillic
- cm_typewriter_greek
- ...

[Default: dialog]

fontsize = `<int-value>` (*Integer*)

Sets the font size used for plot annotations.

[Default: 12]

fontstyle = `plain|bold|italic|bold-italic` (*Integer*)

Gives a style in which the font is to be applied for plot annotations. Options are `plain`, `bold`, `italic` and `bold-italic`.

[Default: plain]

grid = `true|false` (*Boolean*)

If `true`, grid lines are drawn on the plot. If `false`, they are absent.

[Default: `true`]

ifmtN = <in-format> (String)

Specifies the format of the input table as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <table> (StarTable)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istreamN = true|false (Boolean)

If set true, the input table specified by the `inN` parameter will be read as a stream. It is necessary to give the `ifmtN` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: `false`]

legend = true|false (Boolean)

Determines whether a legend showing which plotting style is used for each data set. Defaults to true if there is more than one set, false otherwise.

linewidthNS = <int-value> (Integer)

Defines the line width for lines drawn as part of the bars for dataset NS. Only certain bar styles are affected by the line width.

[Default: `2`]

nameNS = <value> (String)

Provides a name to use for a subset with the symbolic label NS. This name will be used for display in the legend, if one is displayed.

norm = true|false (Boolean)

Determines whether bin counts are normalised. If true, histogram bars are scaled such that summed height of all bars over the whole dataset is equal to one. Otherwise (the default), no scaling is done.

[Default: `false`]

ofmt = png|png-transp|gif|jpeg|pdf|eps|eps-gzip (GraphicExporter)

Graphics format in which the plot is written to the output file, see Section 8.6. One of:

- `png`: PNG
- `png-transp`: PNG with transparent background
- `gif`: GIF

- jpeg: JPEG
- pdf: Portable Document Format
- eps: Encapsulated PostScript
- eps-gzip: Gzipped Encapsulated PostScript

May default to a sensible value depending on the filename given by `out`.

omode = `swing|out|cgi|discard|auto` (*PaintMode*)

Determines how the drawn plot will be output, see Section 8.5.

- swing: Plot will be displayed in a window on the screen. This plot is "live"; it can be resized and (except for old-style plots) navigated around with mouse actions in the same way as plots in TOPCAT.
- out: Plot will be written to a file given by `out` using the graphics format given by `ofmt`.
- cgi: Plot will be written in a way suitable for CGI use direct from a web server. The output is in the graphics format given by `ofmt`, preceded by a suitable "Content-type" declaration.
- discard: Plot is drawn, but discarded. There is no output.
- auto: Behaves as swing or out mode depending on presence of `out` parameter

[Default: auto]

out = `<out-file>` (*uk.ac.starlink.util.Destination*)

The location of the output file. This is usually a filename to write to. If it is equal to the special value "-" the output will be written to standard output.

sequence = `<suffix>,<suffix>,...` (*String[]*)

Can be used to control the sequence in which different datasets and subsets are plotted. This will affect which symbols are plotted on top of, and so potentially obscure, which other ones. The value of this parameter is a comma-separated list of the "NS" suffixes which appear on the parameters which apply to subsets. The sets which are named will be plotted in order, so the first-named one will be at the bottom (most likely to be obscured). Note that if this parameter is supplied, then only those sets which are named will be plotted, so this parameter may also be used to restrict which plots appear (though it may not be the most efficient way of doing this). If no explicit value is supplied for this parameter, sets will be plotted in some sequence decided by STILTS (probably alphabetic by suffix).

subsetNS = `<expr>` (*String*)

Gives the selection criterion for the subset labelled "NS". This is a boolean expression which may be the name of a boolean-valued column or any other boolean-valued expression. Rows for which the expression evaluates true will be included in the subset, and those for which it evaluates false will not.

title = `<value>` (*String*)

A one-line title to display at the top of the plot.

weightN = `<value>` (*String*)

Defines a weighting for each point accumulated to determine the height of plotted bars. If this parameter has a value other than 1 (the default) then instead of simply accumulating the number of points per bin to determine bar height, the bar height will be the sum over the weighting expression for the points in each bin. Note that with weighting, the figure drawn is no longer strictly speaking a histogram.

When weighted, bars can be of negative height. An anomaly of the plot as currently implemented is that the Y axis never descends below zero, so any such bars are currently invisible. This may be amended in a future release (contact the author to lobby for such an amendment).

[Default: 1]

xdataN = `<expr>` (*String*)

Gives a column name or expression for the x axis data for table N. The expression is a numeric algebraic expression based on column names as described in Section 10

xflip = `true|false` (*Boolean*)

If set true, the scale on the x axis will increase in the opposite sense from usual (e.g. right to left rather than left to right).

[Default: `false`]

xhi = `<float-value>` (*Double*)

The upper limit for the plotted x axis. If not set, a value will be chosen which is high enough to accommodate all the data.

xlabel = `<value>` (*String*)

Specifies a label to be used for annotating axis x. A default values based on the plotted data will be used if no value is supplied for this parameter.

xlo = `<float-value>` (*Double*)

The lower limit for the plotted x axis. If not set, a value will be chosen which is low enough to accommodate all the data.

xlog = `true|false` (*Boolean*)

If false (the default), the scale on the x axis is linear; if true it is logarithmic.

[Default: `false`]

xpix = `<int-value>` (*Integer*)

The width of the output graphic in pixels.

[Default: 400]

yhi = `<float-value>` (*Double*)

Upper bound for Y axis. Autogenerated from the data if not supplied.

ylabel = `<value>` (*String*)

Specifies a label for annotating the vertical axis. A default value based on the type of histogram will be used if no value is supplied for this parameter.

[Default: `Count`]

ylo = `<float-value>` (*Double*)

Lower bound for Y axis.

[Default: `0.0`]

ylog = `true|false` (*Boolean*)

Whether to use a logarithmic scale for the Y axis.

[Default: `false`]

ypix = `<int-value>` (*Integer*)

The height of the output graphic in pixels.

[Default: 300]

B.14.2 Examples

Here are some examples of `plothist` in use:

```
stilts plothist in=cat.xml xdata=RMAG-BMAG
```

Plots a histogram of the R-B colour. The plot is displayed directly on the screen.

The generated plot is here.

```
stilts plothist in=cat.xml xdata=RMAG-BMAG ofmt=eps-gzip out=hist.eps.gz
```

Makes the same plot as the previous example, but writes it to a gzipped encapsulated postscript file instead of displaying it on the screen.

The generated plot is here.

```
stilts plothist inJ=2mass_xsc.fits xdataJ=j_m_k20fe barstyleJ=tops
inH=2mass_xsc.fits xdataH=h_m_k20fe barstyleH=tops
inK=2mass_xsc.fits xdataK=k_m_k20fe barstyleK=tops
binwidth=0.1 xlo=12 xhi=16 xflip=true xlabel=Magnitude xpix=500
out=2mass.gif
```

Overplots histograms of three different columns from the same input table. These are treated as three separate datasets which all happen to use the same input file. The different datasets are labelled "J", "H" and "K" so these suffixes appear on all the dataset-dependent parameters which are supplied. The binwidth and X range are specified explicitly rather than leaving them to be chosen automatically by examining the data.

The generated plot is here.

B.15 regquery: Queries the VO registry

`regquery` submits a query to the Virtual Observatory **registry** and returns the result as a table containing all the records which match the condition specified. The resulting table can be written out in any of the supported formats or otherwise processed in the usual ways. Making use of this command requires an understanding of the VOResource schema.

It is important to note that the results of this command give a very much flattened and incomplete view of the results of a full registry query. That is because the contents of an IVOA Registry (see the IVOA Resource Metadata and VOResource documents for more detail) are hierarchical and cannot be faithfully represented in a simple tabular structure. Other superior registry search clients exist; this command is just useful for viewing the results in a rather simplified way which can be represented as a table.

B.15.1 Usage

The usage of `regquery` is

```
stilts <stilts-flags> regquery query=<value> regurl=<url-value>
soapout=<out-file> ocmd=<cmds>
omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|t
out=<out-table> ofmt=<out-format>
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.RegQuery`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

ocmd = <cmds> (*ProcessingStep*[])

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline

which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (String)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui (ProcessingMode)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (TableConsumer)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

query = <value> (String)

Text of an ADQL WHERE clause targeted at the VOResource 1.0 schema defining which resource records you wish to retrieve from the registry. Some examples are:

- @xsi:type like '%Organisation%'
- capability/@standardID = 'ivo:///ivoa.net/std/ConeSearch' and title like '%SDSS%'
- curation/publisher like 'CDS%' and title like '%galax%'

A full description of ADQL syntax and of the VOResource schema is well beyond the scope of

this documentation, but in general you want to use `<field-name>` like `'<value>'` where `'%'` is a wildcard character. Logical operators `and` and `or` and parentheses can be used to group and combine expressions. To work out the various `<field-name>`s you need to look at the VOResource 1.0 schema; you can find some more discussion in the documentation of the NVO IVOARegistry package.

regurl = `<url-value>` (URL)

The URL of a SOAP endpoint which provides a VOResource1.0 IVOA registry service. Some known suitable registry endpoints at time of writing are

- `http://registry.astrogrid.org/astrogrid-registry/services/RegistryQueryv1_0`
- `http://registry.euro-vo.org/services/RegistrySearch`
- `http://vao.stsci.edu/directory/ristandardservice.asmx`

[Default:

`http://registry.astrogrid.org/astrogrid-registry/services/RegistryQueryv1_0]`

soapout = `<out-file>` (*uk.ac.starlink.util.Destination*)

If set to a non-null value, this gives the destination for the text of the request and response SOAP messages. The special value `"-"` indicates standard output.

B.15.2 Examples

Here are some examples of `regquery`:

```
stilts regquery query="title like '%IRAS%' ofmt=ascii out=iras.txt
```

Retrieves all the records in the registry whose `title` field contain the string "IRAS". The `'%'` characters function as wildcards for the ADQL `like` operator. The output is written to a local ASCII table which can be examined later.

```
stilts regquery query="capability/@standardID = 'ivo://ivoa.net/std/ConeSearch'
                  and curation/@publisher like '%astrogrid%'
                  omode=count
```

Searches for all resources which offer a cone search service and are published by AstroGrid. In this case the records are not stored, but the `omode=count` output mode counts the rows. This therefore tells you how many AstroGrid cone search services are in the registry.

```
stilts regquery query="capability/@standardID = 'ivo://ivoa.net/std/SSA'
                  ocmd='keepcols identifier accessUrl'
                  ofmt=ascii out=-
```

Queries the registry for all Simple Spectral Access services. The `keepcols` filter takes the result and throws away all the columns except for `identifier` and `accessUrl`, and these are written to the terminal in ASCII format.

B.16 server: Runs an HTTP server to perform STILTS commands

`server` runs an HTTP server which makes some or all of the various STILTS tasks available to local or remote clients making HTTP requests rather than using the more usual command line interface.

When you run `server` it will start up a server which runs until it is interrupted, and write to the screen the *base URL* at which it can be accessed, for instance `"http://localhost:2112/stilts/"`. If you point your browser here you will see some examples (hyperlinks to server requests) of how to use the server. Currently there are two main sets of capabilities:

Tasks (*baseURL* /*task*/ *task-name*)

There is a URL as above associated with each STILTS task provided by the server. The task parameters are passed in the usual way for HTTP queries, using application/x-www-form-urlencoded (see e.g. the HTML FORM specification). Some examples are given in the Client Examples subsection below. Either HTTP GET or POST methods may be used; since the task invocations will normally be idempotent, GET is more respectable, but long URLs can cause trouble in some circumstances (MS IE apparently imposes a limit of about 2000 characters) so POST may be preferable for lengthy invocations.

Forms (*baseURL* /*form*/)

There are a couple of example HTML Forms which can be used to access the server tasks. These by no means show all the capabilities of the tasks that they use, they are just intended to be examples of how forms can be used in this way.

In general if you request a URL which contains no useful information, an attempt will be made to return an HTML page directing you to a more useful starting point.

You might want to run STILTS in server mode if you are providing a web service to external users which is able to access files residing on the server, for instance generating table plots or row selections on the fly. This can be done without the server mode, for instance by invoking the `stilts` script or `java` from a CGI script to serve each request, but using server mode has two advantages: first it provides correct HTTP headers such as Content-Types, and secondly it avoids the Java startup overheads for each invocation. Note however that in its current form no great attention has been paid to security, so it may be possible for clients to read and write files and expend significant system resources by making certain requests to the server. Anyone exposing the STILTS HTTP server directly to external clients should bear this in mind.

For more flexibility you can run STILTS in servlet mode. See the javadocs and sources of the `uk.ac.starlink.ttools.server.TaskServlet` class. The `server` command is a fairly thin wrapper around this, which simply deploys the servlet in an embedded web application container (Jetty). By using the servlet class in your own custom web application instead you can customise the way it is accessed, for instance providing improved security.

Note: The `server` command and associated servlet code are at time of writing (v2.0) experimental, and probably buggy and missing some features which ought to be present. If you have requirements which are not currently provided, please contact the author for discussion.

B.16.1 Usage

The usage of `server` is

```
stilts <stilts-flags> server port=<int-value> basepath=<value>
                             tasks=<task-name> ...
                             tablefactory=file|dirs:...|locator:...
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.StiltsServer`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

basepath = <value> (*String*)

Base path on the server at which request URLs are rooted. The default is `/stilts`, which means that for instance requests to execute task `plot2d` should be directed to the URL `http://host:portnum/stilts/task/plot2d?name=value&name=value...`

[Default: /stilts]

port = <int-value> (*Integer*)

Port number on which the server should run.

[Default: 2112]

tablefactory = file|dirs:...|locator:... (*StarTableFactory*)

This parameter determines how input table names (typically the `in` parameter of table processing commands) are used to acquire references to actual table data. The default behaviour is for input table names to be treated as filenames, in conjunction with some file type parameter. While this is usually sensible for local use, in server situations it may be inappropriate, since you don't want external users to have read access to your entire filesystem.

This parameter gives options for alternative ways of mapping table names to table data items. The currently available options are:

- `file`: default behaviour - names are treated as filenames
- `dirs:<dir>:<dir>:...:` following the "dirs:" prefix a list of directories is specified which will be searched for the file named. Note that the directory separator character differs between operating systems; it is a colon (":") for Unix-like OSs and a semi-colon (";") for MS Windows. If a given name is identical to the path-less filename in one of the <dir> directories, that file is used as the referenced table. File type information is ignored in this case, so the files must be one of the types which STILTS can autodetect, currently FITS or VOTable (FITS is more efficient). By using this option, clients can be restricted to using a fixed set of tables in a restricted part of the server's file system.
- `locator:<class-name>:` the <class-name> must be the name of a Java class on the classpath which implements the interface `uk.ac.starlink.ttools.task.TableLocator` and which has a no-arg constructor. An instance of this class will be used to resolve names to tables.

The usage and functionality of this parameter is experimental, and may change significantly in future releases.

[Default: file]

tasks = <task-name> ... (*String*)

Gives a space-separated list of tasks which will be provided by the running server. If the value is `null` then all tasks will be available. However, some tasks don't make a lot of sense to run from the server, so the default value is a somewhat restricted list. If the server is being exposed to external users, you might also want to reduce the list for security reasons.

[Default: calc cdsskymatch coneskymatch pixfoot pixsample plot2d plot3d plothist regquery sqlclient sqlskymatch sqlupdate taplint tapquery tapresume tapskymatch tcat tcatn tcopy tcube tjoin tloop tmatch1 tmatch2 tmatchn tmulti tmultin tpipe tskymatch2 votcopy votlint plot2plane plot2sky plot2cube plot2sphere plot2time]

B.16.2 Examples

Here are some examples of running the `server` command:

stilts server

Starts a server on the default port until it is interrupted. Most tasks are available in server mode. A message will be printed on standard output indicating the base URL at which it may be accessed, for instance "http://localhost:2112/stilts/".

stilts server port=2100 basepath=tableserv

Starts a server running on port 2100 with a given URL. The URL at which, for instance, the `plot2d` task can be executed will be "`http:// host :2100/tableserv/task/plot2d`"

```
stilts server tasks="plot2d plothist"
```

Starts a server with a restricted list of tasks available. Only the plotting tasks `plot2d` and `plothist` will be available for execution by clients.

B.16.3 Client Examples

Here are some examples of URLs which can be retrieved from a server which is running at the base URL `http://localhost:2112/stilts/`. All these use the HTTP GET form of request; the POST form could be used instead with the same effect.

```
http://localhost:2112/stilts/
```

Returns an HTML page giving version information and some links to example usages of the server.

```
http://localhost:2112/stilts/task/tpipe
```

Returns an HTML page giving usage instructions for the `tpipe` task.

```
http://localhost:2112/stilts/task/calc?expression=21%2b2
```

Invokes the `calc` task to return a document containing the text "23". Note that the plus ("+" sign in the expression has to be encoded using the sequence "%2b" since "+" has a special significance in query URLs - see for instance sec 2.2 of RFC 1738.

```
http://localhost:2112/stilts/task/plot2d?in=/data/table1.vot&xdata=RMAG&ydata=BMAG
```

Invokes the `plot2d` task to return a magnitude-magnitude diagram of the named local file as an image (probably an `image/png`).

```
http://localhost:2112/stilts/task/tcopy?in=/data/cat.fits&ofmt=votable
```

Invokes the `tcopy` task to return a translation of the named local FITS file to VOTable format.

B.17 `sqlclient`: Executes SQL statements

`sqlclient` is a simple command-line client for use with SQL databases. One or more SQL statements can be supplied using the `sql` parameter. The result of each statement may be one or more update counts (for update-type statements) or tables (for query-type statements). Tables will be written to standard output in a format given by the `ofmt` parameter. Update results and timing information will be written to standard error.

In most cases, you will find life easier if you use either the database's own command-line or GUI client, or, if you require STILTS-type format conversion or post-processing, a `jdbc:-format` URL for the `in` parameter of the `tpipe` or `tcopy` commands (see Section 3.4 for more explanation of the latter). However, this command enables you to submit multiple queries over the same JDBC connection, including ones which do not generate a tabular result. It may be useful if a command-line client is not available to you for the database you are using.

This command can only be used if you have access to an SQL database via JDBC. The details of

how to configure a JDBC connection to a database are discussed in Section 3.4 - obviously you will need a database to connect to and appropriate permissions on it as well as the relevant drivers.

This command is experimental, and it may be enhanced, renamed or withdrawn in future releases.

B.17.1 Usage

The usage of `sqlclient` is

```
stilts <stilts-flags> sqlclient db=<jdbc-url> user=<value> password=<value>
                                sql=<sql> ofmt=<out-format>
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.SqlClient`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

db = <jdbc-url> (*Connection*)

URL which defines a connection to a database. This has the form `jdbc:<subprotocol>:<subname>` - the details are database- and driver-dependent. Consult Sun's JDBC documentation and that for the particular JDBC driver you are using for details. Note that the relevant driver class will need to be on your classpath and referenced in the `jdbc.drivers` system property as well for the connection to be made.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value `"(auto)"` (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

[Default: `text`]

password = <value> (*String*)

Password for logging in to SQL database.

sql = <sql> (*String*)

Text of an SQL statement for execution. This parameter may be repeated, or statements may be separated by semicolon `";"` characters.

user = <value> (*String*)

User name for logging in to SQL database. Defaults to the current username.

[Default: `mbt`]

B.17.2 Examples

Here are some examples of `sqlclient`:

```
stilts -classpath lib/drivers.jtds-1.1.jar \
-Djdbc.drivers=net.sourceforge.jtds.jdbc.Driver \
-Djava.net.preferIPv4Stack=true \
sqlclient \
    db='jdbc:jtds:sqlserver://amenhotep:1433/twomass' \
    user='guest1' \
    ofmt=csv-nohead \
    sql='SET SHOWPLAN_TEXT ON' \
```

```
sql='SELECT ra,dec FROM twomass_psc WHERE ra BETWEEN 21.7 AND 21.8 \
      AND dec BETWEEN 9.1 AND 9.12'
```

This sends two commands to a SQL Server database; the first one (SET SHOWPLAN...) sets a flag which causes the DB to return an execution plan rather than the result for subsequent queries, and the second makes the query itself. Since the password is not provided on the command line, a prompt for it will be issued before execution. The result is SQL Server's execution plan for the SELECT statement expressed as a headerless comma-separated value table sent to the terminal. CSV is chosen for the output format since it does not truncate wide columns.

B.18 sqlskymatch: Crossmatches table on sky position against SQL table

sqlskymatch resembles coneskymatch (Appendix B.3), but instead of sending an HTTP query to a remote cone search service for each match (i.e. each row of the input table), it executes an SQL query directly. The query is a SELECT statement with a WHERE clause which makes restrictions on Right Ascension and Declination columns; the names of these columns must be given as parameters. The effect is that of a spatial join between a client-side table and a table stored in the database.

This command can only be used if you have access to an SQL database via JDBC. The details of how to configure a JDBC connection to a database are discussed in Section 3.4 - obviously you will need a database to connect to and appropriate read permissions on it as well as the relevant drivers.

Note: this task was known as sqlcone in its experimental form in STILTS v1.3.

B.18.1 Usage

The usage of sqlskymatch is

```
stilts <stilts-flags> sqlskymatch ifmt=<in-format> istream=true|false
                                icmd=<cmds> ocmd=<cmds>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plast
                                out=<out-table> ofmt=<out-format>
                                ra=<expr> dec=<expr> sr=<expr/deg>
                                find=best|all|each usefoot=true|false
                                footnside=<int-value>
                                copycols=<colid-list> scorecol=<col-name>
                                erract=abort|ignore|retry|retry<n>
                                ostream=true|false fixcols=none|dups|all
                                suffix0=<label> suffix1=<label>
                                db=<jdbc-url> user=<value>
                                password=<value> dbtable=<table-name>
                                dbra=<sql-col> dbdec=<sql-col>
                                dbunit=deg|rad
                                tiling=htm<level>|healpixnest<nside>|healpixring<nside>
                                dbtile=<sql-col> selectcols=<sql-cols>
                                where=<sql-condition>
                                preparesql=true|false
                                [in=<table>]
```

If you don't have the stilts script installed, write "java -jar stilts.jar" instead of "stilts" - see Section 3. The available <stilts-flags> are listed in Section 2.1. For programmatic invocation, the Task class for this command is uk.ac.starlink.ttools.task.SqlCone.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

copycols = <colid-list> (*String*)

List of columns from the input table which are to be copied to the output table. Each column

identified here will be prepended to the columns of the combined output table, and its value for each row taken from the input table row which provided the parameters of the query which produced it. See Section 6.3 for list syntax. The default setting is "*", which means that all columns from the input table are included in the output.

[Default: *]

db = <jdbc-url> (*Connection*)

URL which defines a connection to a database. This has the form jdbc:<subprotocol>:<subname> - the details are database- and driver-dependent. Consult Sun's JDBC documentation and that for the particular JDBC driver you are using for details. Note that the relevant driver class will need to be on your classpath and referenced in the jdbc.drivers system property as well for the connection to be made.

dbdec = <sql-col> (*String*)

The name of a column in the SQL database table dbtable which gives the declination. Units are given by dbunit.

dbra = <sql-col> (*String*)

The name of a column in the SQL database table dbtable which gives the right ascension. Units are given by dbunit.

dbtable = <table-name> (*String*)

The name of the table in the SQL database which provides the remote data.

dbtile = <sql-col> (*String*)

The name of a column in the SQL database table dbtable which contains a sky tiling pixel index. The tiling scheme is given by the tiling parameter. Use of a tiling column is optional, but if present (and if the column is indexed in the database table) it may serve to speed up searches. Set to null if the database table contains no tiling column or if you do not wish to use one.

dbunit = deg|rad (*AngleUnits*)

Units of the right ascension and declination columns identified in the database table. May be either deg[rees] (the default) or rad[ians].

[Default: deg]

dec = <expr> (*String*)

Declination in degrees in the coordinate system for the position of each row of the input table. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

erract = abort|ignore|retry|retry<n> (*ConeErrorPolicy*)

Determines what will happen if any of the individual cone search requests fails. By default the task aborts. That may be the best thing to do, but for unreliable or poorly implemented services you may find that some searches fail and others succeed so it can be best to continue operation in the face of a few failures. The options are:

- abort: failure of any query terminates the task
- ignore: failure of a query is treated the same as a query which returns no rows
- retry: failed queries are retried until they succeed; use with care - if the failure is for some good, or at least reproducible reason this could prevent the task from ever completing
- retry<n>: failed queries are retried at most a fixed number <n> of times. If they still fail the task terminates.

[Default: abort]

find = best|all|each (*String*)

Determines which matches are retained.

- **best**: Only the matching query table row closest to the input table row will be output. Input table rows with no matches will be omitted. (Note this corresponds to the `best1` option in the pair matching commands, and `best1` is a permitted alias).
- **all**: All query table rows which match the input table row will be output. Input table rows with no matches will be omitted.
- **each**: There will be one output table row for each input table row. If matches are found, the closest one from the query table will be output, and in the case of no matches, the query table columns will be blank.

[Default: `all`]

fixcols = none|dups|all (*Fixer*)

Determines how input columns are renamed before use in the output table. The choices are:

- **none**: columns are not renamed
- **dups**: columns which would otherwise have duplicate names in the output will be renamed to indicate which table they came from
- **all**: all columns will be renamed to indicate which table they came from

If columns are renamed, the new ones are determined by `suffix*` parameters.

[Default: `dups`]

footnside = <int-value> (*Integer*)

Determines the HEALPix Nside parameter for use with the MOC footprint service. This tuning parameter determines the resolution of the footprint if available. Larger values give better resolution, hence a better chance of avoiding unnecessary queries, but processing them takes longer and retrieving and storing them is more expensive.

The value must be a power of 2, and at the time of writing, the MOC service will not supply footprints at resolutions greater than `nside=512`, so it should be `<=512`.

Only used if `usefoot=true`.

icmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (`;`). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character `'@'`. Thus a value of `"@filename"` causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given

explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.

- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istream = true|false (*Boolean*)

If set true, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

ocmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(*ProcessingMode*)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp

- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

ostream = `true|false` (*Boolean*)

If set true, this will cause the operation to stream on output, so that the output table is built up as the results are obtained from the cone search service. The disadvantage of this is that some output modes and formats need multiple passes through the data to work, so depending on the output destination, the operation may fail if this is set. Use with care (or be prepared for the operation to fail).

[Default: false]

out = `<out-table>` (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

password = `<value>` (*String*)

Password for logging in to SQL database.

preparesql = `true|false` (*Boolean*)

If true, the JDBC connection will use `PreparedStatement`s for the SQL `SELECT`s otherwise it will use simple `Statements`. This is a tuning parameter and affects only performance. On some database/driver combinations it's a lot faster set false (the default); on others it may be faster, who knows?

[Default: false]

ra = `<expr>` (*String*)

Right ascension in degrees in the coordinate system for the position of each row of the input table. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

scorecol = `<col-name>` (*String*)

Gives the name of a column in the output table to contain the distance between the requested central position and the actual position of the returned row. The distance returned is an angular distance in degrees. If a null value is chosen, no distance column will appear in the output table.

[Default: Separation]

selectcols = `<sql-cols>` (*String*)

An SQL expression for the list of columns to be selected from the table in the database. A value of "*" retrieves all columns.

[Default: *]

sr = `<expr/deg>` (*String*)

Expression which evaluates to the search radius in degrees for the request at each row of the input table. This will often be a constant numerical value, but may be the name or ID of a column in the input table, or a function involving one.

suffix0 = `<label>` (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended

to all renamed columns from the input table.

[Default: `_0`]

suffix1 = <label> (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from the cone result table.

[Default: `_1`]

tiling = `htm<level>|healpixnest<nside>|healpixring<nside>` (*SkyTiling*)

Describes the sky tiling scheme that is in use. One of the following values may be used:

- `htm<level>`: Hierarchical Triangular Mesh with a level value of `level`.
- `healpixnest<nside>`: HEALPix using the Nest scheme with an `nside` value of `nside`.
- `healpixring<nside>`: HEALPix using the Ring scheme with an `nside` value of `nside`.

usefoot = `true|false` (*Boolean*)

Determines whether an attempt will be made to restrict searches in accordance with available footprint information. If this is set true, then before any of the per-row queries are performed, an attempt may be made to acquire footprint information about the service. If such information can be obtained, then queries which fall outside the footprint, and hence which are known to yield no results, are skipped. This can speed up the search considerably.

Currently, the only footprints available are those provided by the CDS MOC (Multi-Order Coverage map) service, which covers VizieR and a few other cone search services.

[Default: `true`]

user = <value> (*String*)

User name for logging in to SQL database. Defaults to the current username.

[Default: `mbt`]

where = <sql-condition> (*String*)

An SQL expression further limiting the rows to be selected from the database. This will be combined with the constraints on position implied by the cone search centres and radii. The value of this parameter should just be a condition, it should not contain the `WHERE` keyword. A null value indicates no additional criteria.

B.18.2 Examples

Here are some examples of `sqlskymatch`:

```
stilts -classpath lib/drivers/mysql-connector-java.jar \
       -Djdbc.drivers=com.mysql.jdbc.Driver
       sqlskymatch in=messier.xml ra=RA dec=DEC sr=0.05 \
                   db='jdbc:mysql://localhost/ASTRO1' user=mbt \
                   dbtable=FIRST dbra=_RA2000 dbdec=_DE2000 \
                   out=matches.xml
```

This performs a series of `SELECT` statements on the table `FIRST` in the local MySQL database `ASTRO1` to identify database objects in the region of each object represented in the VOTable `messier.xml`. The result, a join between the `Messier` and `FIRST` tables, is output as a VOTable called `matches.xml`. In this case a password has not been supplied on the command line, so if one is required it will be prompted for on the console.

B.19 `sqlupdate`: Updates values in an SQL table

`sqlupdate` updates values in an existing table in an SQL database. The rows to update are specified, as a normal `SELECT` statement, using the `select` parameter. Each column to update, and the value to write to it, are given using the `assign` parameter.

Why not just use the database's own `UPDATE` statement? In most cases, that would be a much better idea. However, using `sqlupdate` you can write values using STILTS's expression language (Section 10), and hence take advantage of its various functions, without having to embed them into the database. SQL column names can be used as variables in these expressions, in the same way that table column names are used as variables in other commands such as `tpipe`.

This command can only be used if you have access to an SQL database via JDBC. The details of how to configure a JDBC connection to a database are discussed in Section 3.4 - obviously you will need a database to connect to and appropriate write permissions on it as well as the relevant drivers.

This is a somewhat specialised command, and several (database/driver-specific) things can go wrong with it. If you do not have a fairly good understanding of the database with which you are using it then you may run into problems (but then you'd be unlikely to have the permissions to do the updates in any case).

B.19.1 Usage

The usage of `sqlupdate` is

```
stilts <stilts-flags> sqlupdate db=<jdbc-url> user=<value> password=<value>
                                select=<select-stmt> assign=<col>=<expr>
                                progress=true|false
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.SqlUpdate`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`assign = <col>=<expr>` (*Assignment[]*)

Assigns new values for a given column. The assignment is made in the form `<colname>=<expr>` where `<colname>` is the name of a column in the SQL table and `<expr>` is the text of an expression using STILTS's expression language, as described in Section 10. SQL table column names or `$ID` identifiers may be used as variables in the usual way.

This parameter may be supplied more than once to effect multiple assignments, or multiple assignments may be made by separating them with semicolons in the value of this parameter.

`db = <jdbc-url>` (*Connection*)

URL which defines a connection to a database. This has the form `jdbc:<subprotocol>:<subname>` - the details are database- and driver-dependent. Consult Sun's JDBC documentation and that for the particular JDBC driver you are using for details. Note that the relevant driver class will need to be on your classpath and referenced in the `jdbc.drivers` system property as well for the connection to be made.

`password = <value>` (*String*)

Password for logging in to SQL database.

`progress = true|false` (*Boolean*)

If true, a spinner will be drawn on standard error which shows how many rows have been updated so far.

[Default: `true`]

select = <select-stmt> (*String*)

Gives the full text (including "SELECT") of the SELECT statement to identify which rows undergo updates.

user = <value> (*String*)

User name for logging in to SQL database. Defaults to the current username.

[Default: mbt]

B.19.2 Examples

Here are some examples of `sqlupdate`:

```
stilts -classpath lib/drivers/mysql-connector-java.jar \
-Djdbc.drivers=com.mysql.jdbc.Driver \
sqlupdate db='jdbc:mysql://localhost/RADIO' user=root
select='SELECT * from FIRST' \
assign='HTMID=htmIndex(20,POS_EQ_RA,POS_EQ_DEC)'
```

Fills in the HTMID column of a table called FIRST in the local MySQL database RADIO, using HTM pixel indices based on the existing right ascension and declination columns in that table. The HTMID column must exist prior to executing this command.

B.20 `taplint`: Tests TAP services

`taplint` runs a series of tests on a Table Access Protocol (TAP) service and reports the results. Unlike most of the other tools in this package it is not likely to be of use to normal users; its intended use is for people developing or operating TAP services to assess their services, perhaps with a view to improving compliance.

Testing takes place in a number of stages; it is possible to choose which stages are run in by using the `stages` parameter. At present output is line-based text to standard output, and each report line is of the (fairly greppable) form:

```
T-SSS-MMMMxN aaaaa...
```

where the parts have the following meanings:

- **T**: Report type, one of E(rror), W(arning), I(nfo), S(ummary), F(ailure). See the documentation of the `report` parameter for further description of what these mean. The `report` parameter can be used to suppress some of these; only **E** indicates actual service compliance errors, but including the others may make it easier to see what's going on.
- **SSS**: Stage abbreviation, as used in the `stages` parameter. The `stages` parameter can be used to select which stages are run.
- **MMMM**: Message label, which is always the same for messages generated by the same test, is usually different for messages generated by different tests, and may be somewhat mnemonic.
- **x**: Continuation indicator, either "-" or "+". In most cases it is "-", indicating the first line of a message, but multi-line messages (rare) use "-" for the first line and "+" for any continuation lines.
- **N**: Sequence number, which is 1 for the first time message T-SSS-MMMM is reported, and increases by one for each subsequent appearance. After a certain maximum (determined by the `maxrepeat` parameter) additional reports with the same code are no longer output individually, but a summary of the number of reports so discarded is written at the end of the section with the character "x" instead of the sequence number. This behaviour prevents the output being swamped by multiple reports of the same issue. If the `maxrepeat` parameter is increased above 9, more than one digit will be used here (so e.g. for `maxrepeat=999`, the format would be NNN not N).

- `aaaaa...:` Message text, a free text description of what is being reported.

TAP is a complicated beast, referencing many standards (including TAP, UWS, VODataService, ADQL, VOResource, VOSI, TAPRegExt, DALI, ObsCore, VOTable, HTTP, RDFa Lite), and it is hard to write a validator which is comprehensive, especially one which can provide useful output for services with a range of compliance levels. This tool tries to make a wide range of tests, but does not claim to be comprehensive. An idea of what tests it does perform can be gained from the stages listed in the description of the `stages` parameter. It does make a fairly good job of checking that declared metadata is consistent and matches the data actually returned from queries, and it tests job submission in most of the various ways permitted by the TAP standard. Things it does not test much include complex ADQL queries, coordinate/STC-related data types, queries in non-ADQL languages, and service registration.

B.20.1 Usage

The usage of `taplint` is

```
stilts <stilts-flags> taplint
                                stages=TMV|TME|TMS|TMC|CPV|CAP|AVV|QGE|QPO|QAS|UWS|MDQ|OBS|UWS
                                report=[EWISF]+ maxrepeat=<int-value>
                                truncate=<int-value> maxtable=<int-value>
                                debug=true|false
                                [tapurl=<url-value>
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TapLint`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`debug = true|false` (Boolean)

If true, debugging output including stack traces will be output along with the normal validation messages.

[Default: false]

`maxrepeat = <int-value>` (Integer)

Puts a limit on the number of times that a single message will be repeated. By setting this to some reasonably small number, you can ensure that the output does not get cluttered up by millions of repetitions of essentially the same error.

[Default: 9]

`maxtable = <int-value>` (Integer)

Limits the number of tables from the service that will be tested. Currently, this only affects stage `MDQ`. If the value is left blank (the default), or if it is larger than the number of tables actually present in the service, it will have no effect.

`report = [EWISF]+` (String)

Letters indicating which message types should be listed. Each character of the string is one of the letters `E, W, I, S, F`, with the following meanings:

- `E`: Error in operation or standard compliance of the service.
- `W`: Warning that service behaviour is questionable, or contravenes a standard recommendation, but is not in actual violation of the standard.
- `I`: Information about progress, for instance details of queries made.
- `S`: Summary of previous successful/unsuccessful reports.
- `F`: Failure of the validator to perform some testing. The cause is either some error internal to the validator, or some error or missing functionality in the service which has already

been reported.

[Default: EWISF]

stages = TMV|TME|TMS|TMC|CPV|CAP|AVV|QGE|QPO|QAS|UWS|MDQ|OBS|UPL|EXA[...]
(*String[]*)

Lists the validation stages which the validator will perform. Each stage is represented by a short code, as follows:

- TMV: Validate table metadata against XML schema (on)
- TME: Check content of tables metadata from /tables (on)
- TMS: Check content of tables metadata from TAP_SCHEMA (on)
- TMC: Compare table metadata from /tables and TAP_SCHEMA (on)
- CPV: Validate capabilities against XML schema (on)
- CAP: Check content of TAPRegExt capabilities record (on)
- AVV: Validate availability against XML schema (on)
- QGE: Make ADQL queries in sync GET mode (on)
- QPO: Make ADQL queries in sync POST mode (on)
- QAS: Make ADQL queries in async mode (on)
- UWS: Test asynchronous UWS/TAP behaviour (on)
- MDQ: Check table query result columns against declared metadata (on)
- OBS: Test implementation of ObsCore Data Model (on)
- UPL: Make queries with table uploads (on)
- EXA: Check content of examples document (on)

You can specify a list of stage codes, separated by spaces. Order is not significant.

Note that removing some stages may affect the operation of others; for instance table metadata is acquired from the metadata stages, and avoiding those will mean that later stages that use the table metadata to pose queries will not be able to do so with knowledge of the database schema.

[Default: TMV TME TMS TMC CPV CAP AVV QGE QPO QAS UWS MDQ OBS UPL EXA]

tapurl = <url-value> (*URL*)

The base URL of a Table Access Protocol service. This is the bare URL without a trailing "/[a]sync".

truncate = <int-value> (*Integer*)

Limits the line length written to the output.

[Default: 640]

B.20.2 Examples

Here are some examples of `taplint`:

```
stilts taplint http://dc.zah.uni-heidelberg.de/__system__/tap/run/tap
```

Performs a default validation run against the TAP service based at the given URL.

```
stilts taplint tapurl=http://example.com/tap
report=EW stages='TMS UWS' truncate=80 maxrepeat=4
```

A validation run is done against the named TAP service. Only Error and Warning type messages are output, only two validation stages are performed, lines are truncated to a maximum of 80 characters, and each message is repeated a maximum of 4 times. An invocation like this may be suitable if you find the default operation too verbose.

The output of this invocation might look like this:

```

Section TMS: Check content of tables metadata from TAP_SCHEMA
E-TMS-CINT-1 Column principal in TAP_SCHEMA.columns has wrong type char not int
E-TMS-CINT-2 Column std in TAP_SCHEMA.columns has wrong type char not int
W-TMS-CLUN-1 Unused entry in TAP_SCHEMA.columns table: ivoa.obscore

Section UWS: Test asynchronous UWS/TAP behaviour
E-UWS-GMIM-1 Incorrect Content-Type text/xml != text/plain for http://exampl....
E-UWS-GMIM-2 Incorrect Content-Type text/xml != text/plain for http://exampl....
E-UWS-GMIM-3 Incorrect Content-Type text/xml != text/plain for http://exampl....
E-UWS-GMIM-4 Incorrect Content-Type text/xml != text/plain for http://exampl....
E-UWS-GMIM-x (3 more)

Totals: Errors: 9; Warnings: 1

```

B.21 tapquery: Queries a Table Access Protocol server

`tapquery` can query remote databases using the Table Access Protocol (TAP) services by submitting Astronomical Data Query Language queries to them and retrieving the results. TAP and ADQL are Virtual Observatory protocols.

Queries can be submitted in either synchronous or asynchronous mode, as determined by the `sync` parameter. In asynchronous mode, if the query has not been deleted by the time the command exits (see the `delete` parameter), the result can be picked up at a later stage using the `tapresume` command. Table uploads are supported, so it is possible (if the service supports this functionality), to upload a local table to the remote database, perform a query involving it, such as a join with a remote table of some sort, and receive the result. This powerful facility gives you crossmatches between local and remote tables.

This command does not provide any facility for querying the service for either table or capability metadata, so you will need to know about the service capabilities and database structure from some other source (possibly TOPCAT).

Note: this command has been introduced at STILTS version 2.3, at which time most available TAP services are quite new and may not fully conform to the standards, and usage patterns are still settling down. For this reason you may find that some TAP services do not behave quite as expected; it is also possible that in future versions the command behaviour or parameters will change in line with changing service profiles or in the light of user experience.

B.21.1 Usage

The usage of `tapquery` is

```

stilts <stilts-flags> tapquery nupload=<count> ufmtN=<in-format>
                                uploadN=<tableN> ucmdN=<cmds> ocmd=<cmds>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|t
                                out=<out-table> ofmt=<out-format>
                                upnameN=<label> tapurl=<url-value>
                                adql=<value> parse=true|false
                                sync=true|false maxrec=<longint-value>
                                destruction=<value>
                                executionduration=<longint-value>
                                compress=true|false
                                upvotformat=TABLEDATA|BINARY|BINARY2
                                language=<value> poll=<int-value>
                                progress=true|false
                                delete=finished|never|always

```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` -

see Section 3. The available <stiltts-flags> are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TapQuerier`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

adql = <value> (*String*)

Astronomical Data Query Language string specifying the TAP query to execute. ADQL/S resembles SQL, so this string will likely start with "SELECT".

compress = true|false (*Boolean*)

If true, the service is requested to provide HTTP-level compression for the response stream (Accept-Encoding header is set to "gzip", see RFC 2616). This does not guarantee that compression will happen but if the service honours this request it may result in a smaller amount of network traffic at the expense of more processing on the server and client.

[Default: true]

delete = finished|never|always (*DeleteMode*)

Determines under what circumstances the UWS job is to be deleted from the server when its data is no longer required. If it is not deleted, then the job is left on the TAP server and it can be accessed via the normal UWS REST endpoints until it is destroyed by the server.

Possible values:

- `finished`: delete only if the job finished, successfully or not
- `never`: do not delete
- `always`: delete in any case

[Default: finished]

destruction = <value> (*String*)

Posts an updated value of the UWS DESTRUCTION parameter to the query job before it starts. This only makes sense for asynchronous jobs (`sync=false`).

The supplied value should be an ISO-8601-like string, giving the new requested job destruction time. The service is not obliged to honour this request. See UWS v1.0, sec 2.2.3.3.

executionduration = <longint-value> (*Long*)

Posts an updated value of the UWS EXECUTIONDURATION parameter to the query job before it starts. This only makes sense for asynchronous jobs (`sync=false`).

The supplied value is an integer giving the maximum number of wall-clock seconds for which the job is permitted to execute before being forcibly terminated. A value of zero indicates unlimited duration. The service is not obliged to honour this request. See UWS v1.0, sec 2.2.3.4.

language = <value> (*String*)

Language to use for the ADQL-like query. This will usually be "ADQL" (the default), but may be set to some other value supported by the service, for instance a variant indicating a different ADQL version. Note that at present, setting it to "PQL" is not sufficient to submit a PQL query.

[Default: ADQL]

maxrec = <longint-value> (*Long*)

Sets the requested maximum row count for the result of the query. The service is not obliged to respect this, but in the case that it has a default maximum record count, setting this value may raise the limit. If no value is set, the service's default policy will be used.

nupload = <count> (*Integer*)

The number of upload tables for this task. For each of the upload tables N there will be

associated parameters `ufmtN`, `uploadN` and `ucmdN`.

[Default: 0]

ocmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

**omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(*ProcessingMode*)**

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

parse = true|false (*Boolean*)

Determines whether an attempt will be made to check the syntax of the ADQL prior to submitting the query. If this is set true, and if a syntax error is found, the task will fail with an error before any attempt is made to submit the query.

[Default: false]

poll = <int-value> (*Integer*)

Interval to wait between polling attempts, in milliseconds. Asynchronous TAP queries can only find out when they are complete by repeatedly polling the server to find out the job's status. This parameter allows you to set how often that happens. Attempts to set it too low (<50) will be rejected on the assumption that you're thinking in seconds.

[Default: 5000]

progress = true|false (*Boolean*)

If this parameter is set true, progress of the job is reported to standard output as it happens.

[Default: true]

sync = true|false (*Boolean*)

Determines whether the TAP query is submitted in synchronous or asynchronous mode. Synchronous (*true*) means that the result is retrieved over the same HTTP connection that the query is submitted from. This is uncomplicated, but means if the query takes a long time it may time out and the results will be lost. Asynchronous (*false*) means that the job is queued and results may be retrieved later. Normally this command does the necessary waiting around and recovery of the result, though with appropriate settings you can get *tapresume* to pick it up for you later instead. In most cases *false* (the default) is preferred.

[Default: false]

tapurl = <url-value> (*URL*)

The base URL of a Table Access Protocol service. This is the bare URL without a trailing *"/[a]sync"*.

ucmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on upload table #N as specified by parameter *uploadN*, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (*;*). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character *'@'*. Thus a value of *"@filename"* causes the file *filename* to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ufmtN = <in-format> (*String*)

Specifies the format of upload table #N as specified by parameter *uploadN*. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (*auto*) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (*auto*)]

uploadN = <tableN> (*StarTable*)

The location of upload table #N. This may take one of the following forms:

- A filename.
- A URL.
- The special value *"-"*, meaning standard input. In this case the input format must be given

explicitly using the `ufmtN` parameter. Note that not all formats can be streamed in this way.

- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

upnameN = <label> (*String*)

Identifier to use in server-side expressions for uploaded table #N. In ADQL expressions, the table should be referred to as "`TAP_UPLOAD.<label>`".

[Default: `upN`]

upvotformat = TABLEDATA|BINARY|BINARY2 (*uk.ac.starlink.votable.VOTableWriter*)

Determines how any uploaded tables will be serialized for transmission to the TAP server. The supplied string is the name of one of the defined VOTable serialization formats. The choice shouldn't affect any results, though it may affect required bandwidth, and some services may (though should not) have non-standard requirements for serialization format.

[Default: `TABLEDATA`]

B.21.2 Examples

Here are some examples of `tapquery`:

```
stilts tapquery tapurl='http://dc.zah.uni-heidelberg.de/__system__/tap/run/tap'
               adql='SELECT TOP 1000 * FROM ppmxl.main'
               out=ppmxl.fits
```

Executes the given ADQL query on the service referenced by the URL and writes the result to a FITS file.

```
stilts tapquery
tapurl='http://dc.zah.uni-heidelberg.de/__system__/tap/run/tap'
adql="SELECT *
      FROM twomass.data AS t
      JOIN TAP_UPLOAD.up1 AS s
      ON 1=CONTAINS(POINT('ICRS', t.RAJ2000, t.DEJ2000),
                    CIRCLE('ICRS', s.ra2000, s.dec2000, 5./3600.))"
nupload=1 upload1=6dfgs_E7.fits ucmd1='select BMAG-RMAG<0'
maxrec=20000
ocmd='tablename 2mass_x_6df' omode=topcat
```

The local table `6dfgs_E7` is filtered to contain only rather blue objects, and the resulting selection is uploaded to the TAP server. A positional crossmatch with 5 arcsec tolerance is then performed on the server between this uploaded table and the `twomass.data` table held by the service. The adjusted `maxrec` parameter ensures that the result will not be artificially truncated to shorter than 20000 rows (assuming the service limits permit this). When the result is received, it is loaded directly into TOPCAT with the name "`2mass_x_6df`".

B.22 `tapresume`: Resumes a previous query to a Table Access Protocol server

`tapresume` can resume monitoring and data retrieval from an asynchronous Table Access Protocol query which has already been submitted. TAP is a Virtual Observatory protocol. Such a pre-existing query may have been submitted by the `tapquery` command or by some completely different mechanism. It essentially does the same job as `tapquery` but without the job submission stage. It waits until the query has completed, and then retrieves the table result and processes it in

accordance with the supplied parameters. The query may or may not be deleted from the server as part of the operation.

B.22.1 Usage

The usage of `tapresume` is

```
stilts <stilts-flags> tapresume joburl=<url-value> compress=true|false
                                poll=<int-value> progress=true|false
                                delete=finished|never|always ocmd=<cmds>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic
                                out=<out-table> ofmt=<out-format>
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TapResume`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

compress = true|false (*Boolean*)

If true, the service is requested to provide HTTP-level compression for the response stream (Accept-Encoding header is set to "gzip", see RFC 2616). This does not guarantee that compression will happen but if the service honours this request it may result in a smaller amount of network traffic at the expense of more processing on the server and client.

[Default: true]

delete = finished|never|always (*DeleteMode*)

Determines under what circumstances the UWS job is to be deleted from the server when its data is no longer required. If it is not deleted, then the job is left on the TAP server and it can be accessed via the normal UWS REST endpoints until it is destroyed by the server.

Possible values:

- `finished`: delete only if the job finished, successfully or not
- `never`: do not delete
- `always`: delete in any case

[Default: finished]

joburl = <url-value> (*URL*)

The URL of a job created by submission of a TAP query which was created earlier and has not yet been deleted (by the client) or destroyed (by the server). This will usually be of the form `<tap-url>/async/<job-id>`. You can also find out, and possibly retrieve results from the job by pointing a web browser at this URL.

ocmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "`@filename`" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(ProcessingMode)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (`<mode-args>`) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> **(TableConsumer)**

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

poll = <int-value> **(Integer)**

Interval to wait between polling attempts, in milliseconds. Asynchronous TAP queries can only find out when they are complete by repeatedly polling the server to find out the job's status. This parameter allows you to set how often that happens. Attempts to set it too low (<50) will be rejected on the assumption that you're thinking in seconds.

[Default: 5000]

progress = true|false **(Boolean)**

If this parameter is set true, progress of the job is reported to standard output as it happens.

[Default: true]

B.22.2 Examples

Here are some examples of `tapresume`:

```
stilts tapresume joburl='http://dc.zah.uni-heidelberg.de/__system__/tap/run/tap/async/d4ENGR
out=result.csv ofmt=csv
```

Resumes waiting for the output of a query on a job with ID d4ENGR which was previously started on the GAVO TAP server. When it has completed the output table will be written as a comma-separated value file.

B.23 tapskymatch: Crossmatches table on sky position against TAP table

tapskymatch allows you to perform a positional crossmatch of a local table with one held in a remote TAP service, as long as that TAP supports upload queries. This task does three main jobs. First, it prepares the ADQL queries and TAP negotiations for you so that you don't need to remember the syntax for performing positional crossmatches against a TAP service. Second, it organises data transfer so that only those columns required (basically the positional ones) are transmitted to and from the service, to save on bandwidth. And third it divides the job up into chunks, so that the TAP service only has to perform a manageable-sized query at a time. If the job is large this chunking can be useful to monitor progress of the job, and it also allows you to perform a match which would otherwise hit the upload or output limits imposed by the service.

The positional match may be done in any spherical coordinate system, it's up to the user to ensure that the same coordinates are provided for the local and remote tables.

Note that `cdsskymatch` provides similar functionality by accessing a different external service, which is usually much faster; if the table you wish to match is part of the VizieR database, you may wish to use that command instead.

B.23.1 Usage

The usage of `tapskymatch` is

```
stilts <stilts-flags> tapskymatch ifmt=<in-format> istream=true|false
                                icmd=<cmds> ocmd=<cmds>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plast
                                out=<out-table> ofmt=<out-format>
                                inlon=<expr/deg> inlat=<expr/deg>
                                tapurl=<url-value> taptable=<name>
                                taplon=<column> taplat=<column>
                                tapcols=<colname,...> sr=<expr/deg>
                                find=all|best|each|each-dist
                                blocksize=<int-value> maxrec=<int-value>
                                sync=true|false compress=true|false
                                fixcols=none|dups|all suffixin=<label>
                                suffixremote=<label>
                                [in=<table>
```

If you don't have the `stilts` script installed, write "java -jar stilts.jar" instead of "stilts" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TapUploadSkyMatch`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

blocksize = <int-value> (Integer)

The number of rows uploaded in each TAP query. TAP services may have limits on the number of rows in a table uploaded for matching. This command can therefore break up input tables into blocks and make a number of individual TAP queries to generate the result. This parameter controls the maximum number of rows uploaded in each individual request. For an

input table with fewer rows than this value, the whole thing is done as a single query.

[Default: 5000]

compress = **true|false** (*Boolean*)

If true, the service is requested to provide HTTP-level compression for the response stream (Accept-Encoding header is set to "gzip", see RFC 2616). This does not guarantee that compression will happen but if the service honours this request it may result in a smaller amount of network traffic at the expense of more processing on the server and client.

[Default: true]

find = **all|best|each|each-dist** (*UserFindMode*)

Determines which pair matches are included in the result.

- **all**: All matches
- **best**: Matched rows, best remote row for each input row
- **each**: One row per input row, contains best remote match or blank
- **each-dist**: One row per input row, column giving distance only for best match

Note only the **all** mode is symmetric between the two tables.

[Default: all]

fixcols = **none|dups|all** (*Fixer*)

Determines how input columns are renamed before use in the output table. The choices are:

- **none**: columns are not renamed
- **dups**: columns which would otherwise have duplicate names in the output will be renamed to indicate which table they came from
- **all**: all columns will be renamed to indicate which table they came from

If columns are renamed, the new ones are determined by **suffix*** parameters.

[Default: dups]

icmd = **<cmds>** (*ProcessingStep[]*)

Specifies processing to be performed on the input table as specified by parameter **in**, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file **filename** to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = **<in-format>** (*String*)

Specifies the format of the input table as specified by parameter **in**. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (**auto**) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (auto)]

in = **<table>** (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given

explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.

- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

inlat = <expr/deg> (*String*)

Longitude in degrees for the position of each row in the input table. This may simply be a column name, or it may be an algebraic expression as explained in Section 10. The coordinate system must match that used for the coordinates in the remote table.

inlon = <expr/deg> (*String*)

Longitude in degrees for the position of each row in the input table. This may simply be a column name, or it may be an algebraic expression as explained in Section 10. The coordinate system must match that used for the coordinates in the remote table.

istream = true|false (*Boolean*)

If set true, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

maxrec = <int-value> (*Integer*)

Limit to the number of rows resulting from this operation. If the value is negative (the default) no limit is imposed. Note however that there can be truncation of the result if the number of records returned from a single chunk exceeds limits imposed by the service.

[Default: -1]

ocmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(*ProcessingMode*)

The mode in which the result table will be output. The default mode is `out`, which means that

the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (`<mode-args>`) are required to determine the exact behaviour.

Possible values are

- `out`
- `meta`
- `stats`
- `count`
- `cgi`
- `discard`
- `topcat`
- `samp`
- `plastic`
- `tosql`
- `gui`

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: `out`]

out = `<out-table>` (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value `"-"` (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of `"out"`.

[Default: `-`]

sr = `<expr/deg>` (*String*)

Maximum distance in degrees from the local table (lat,lon) position at which counterparts from the remote table will be identified. This is an ADQL expression interpreted within the TAP service, so it may be a constant value or may involve columns in the remote table.

suffixin = `<label>` (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from the input table.

[Default: `_in`]

suffixremote = `<label>` (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from the TAP result table.

[Default: `_tap`]

sync = `true|false` (*Boolean*)

Determines whether the TAP queries are submitted in synchronous or asynchronous mode. Since this command uses chunking to keep requests to a reasonable size, hopefully requests will not take too long to execute, therefore the default is synchronous (`true`).

[Default: `true`]

tapcols = `<colname,...>` (*String[]*)

Comma-separated list of column names to retrieve from the remote table. If no value is supplied (the default), all columns from the remote table will be returned.

taplat = `<column>` (*String*)

Latitude in degrees for the position of each row in the remote table. This is an ADQL

expression interpreted within the TAP service, typically just a column name. The coordinate system must match that used for the input table.

taplon = <column> (*String*)

Longitude in degrees for the position of each row in the remote table. This is an ADQL expression interpreted within the TAP service, typically just a column name. The coordinate system must match that used for the input table.

taptable = <name> (*String*)

Name of the table in the given TAP service against which the matching will be performed.

tapurl = <url-value> (*URL*)

The base URL of a Table Access Protocol service. This is the bare URL without a trailing `"/[a]sync"`.

B.23.2 Examples

Here are some examples of `tapskymatch`:

```
stilts tapskymatch tapurl=http://dc.g-vo.org/tap
                  taptable=twomass.data taplon=raj2000 taplat=dej2000
                  in=dr5qso.fits inlon=RA inlat=DEC sr=0.00027 find=all
                  out=qso_2mass.fits
```

Matches a local catalogue `dr5qso.fits` against the table named `twomass.data` in the GAVO TAP service. The search radius is 1/3600 degrees (1 arcsecond) and all 2MASS sources within the radius of each input source are returned.

If you run the command with `"stilts -verbose ..."` the text of the ADQL query submitted to the TAP service will (amongst other things) be logged on the console, and you will also see the number of rows uploaded and matched in each chunk.

```
stilts tapskymatch tapurl=http://dc.g-vo.org/tap
                  taptable=rave.dr3 taplon=raj2000 taplat=dej2000
                  tapcols=name,raj2000,dej2000,pmra,pmde
                  in=hip_main.fits inlon=RAdeg inlat=DEdeg
                  icmd='keepcols "HIP RAdeg DEdeg pmra pmde"'
                  sr=0.00027
                  icmd='select nearMoc("\'III/265/ravedr3\'",RAdeg,DEdeg,.00027)'
                  icmd=cache icmd=progress
                  blocksize=5000
                  fixcols=all suffixin=_hip suffixremote=_rave
                  find=best
                  omode=topcat
```

This matches a local copy of the Hipparcos survey against a remote copy of the RAVE survey with a 1-arcsecond radius. The output table contains only the identifier, position and proper motion columns from both the input table (by using the `keepcols` filter) and the remote table (by specifying `tapcols`); the other columns are discarded. The `fixcols` and `suffix*` parameters ensure that a suffix is added to all the output column names, `_hip` for the input (Hipparcos) columns and `_rave` for the remote (RAVE) ones.

Before uploading, the input table is preprocessed by selecting only those rows that fall within the actual footprint of the RAVE survey, by filtering with a MOC giving RAVE coverage (the RAVE dr3 MOC is also available at this URL). This step reduces the amount of data that needs to be uploaded, since only those rows in the given coverage region stand a chance of having a match in the remote table. Note use of the `nearMoc` function with the value of the match radius as the fourth parameter; this includes those objects which may be outside the actual MOC region but close enough that a match could still result.

The `blocksize` parameter determines the number of rows uploaded at a time. If you receive warnings that the output has been truncated, you should decrease this number.

Progress is displayed as the match continues. The `cache` filter must be applied upstream of (before) the `progress` filter itself for this to work, since otherwise the match processing reads all the input rows before the actual work is done, and the progress monitor completes before the match actually starts.

B.24 `tcat`: Concatenates multiple similar tables

`tcat` is a tool for concatenating any number of similar tables one after the other. The tables must be of similar form to each other (same number and types of columns). Preprocessing of the tables may be done using the `icmd` parameter, which will operate in the same way on all the input tables. Table parameters of the output table will be taken from the first of the input tables.

Subject to some constraints on the details of the input and output formats and processing, `tcat` is capable of joining an unlimited number of tables together to produce an output table of unlimited length, without large memory requirements.

If you have heterogeneous tables, in different formats or requiring different preprocessing steps from each other before they can be concatenated, use `tcatn` instead.

B.24.1 Usage

The usage of `tcat` is

```
stilts <stilts-flags> tcat in=<table> [<table> ...] ifmt=<in-format>
                                multi=true|false istream=true|false icmd=<cmds>
                                ocmd=<cmds>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql
                                out=<out-table> ofmt=<out-format>
                                seqcol=<colname> loccol=<colname>
                                uloccol=<colname> lazy=true|false
                                countrows=true|false
```

If you don't have the `stilts` script installed, write "java -jar `stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TableCat`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

countrows = true|false (*Boolean*)

Whether to count the rows in the table before starting the output. This is essentially a tuning parameter - if writing to an output format which requires the number of rows up front (such as normal FITS) it may result in skipping the number of passes through the input files required for processing. Unless you have a good understanding of the internals of the software, your best bet for working out whether to set this true or false is to try it both ways

[Default: false]

icmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on each input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter

commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = <in-format> (String)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

The same format parameter applies to all the tables specified by `in`.

[Default: `(auto)`]

in = <table> [<table> ...] (TableProducer[])

Locations of the input tables. Either specify the parameter multiple times, or supply the input tables as a space-separated list within a single use.

The following table location forms are allowed:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

Compression in any of the supported compression formats (Unix compress, gzip or bzip2) is expanded automatically.

A list of input table locations may be given in an external file by using the indirection character '@'. Thus "`in=@filename`" causes the file `filename` to be read for a list of input table locations. The locations in the file should each be on a separate line.

istream = true|false (Boolean)

If set true, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

The same streaming flag applies to all the tables specified by `in`.

[Default: `false`]

lazy = true|false (Boolean)

Whether to perform table resolution lazily. If true, each table is only accessed when the time comes to add its rows to the output; if false, then all the tables are accessed up front. This is mostly a tuning parameter, and on the whole it doesn't matter much how it is set, but for joining an enormous number of tables setting it true may avoid running out of resources.

[Default: `false`]

loccol = <colname> (String)

Name of a column to be added to the output table which will contain the location (as specified in the input parameter(s)) of the input table from which each row originated.

multi = true|false (Boolean)

Determines whether all tables, or just the first one, from input table files will be used. If set `false`, then just the first table from each file named by `in` will be used. If `true`, then all tables present in those input files will be used. This only has an effect for file formats which are

capable of containing more than one table, which effectively means FITS and VOTable and their variants.

[Default: false]

ocmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

**omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(*ProcessingMode*)**

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

seqcol = <colname> (*String*)

Name of a column to be added to the output table which will contain the sequence number of the input table from which each row originated. This column will contain 1 for the rows from the first concatenated table, 2 for the second, and so on.

uloccol = <colname> (*String*)

Name of a column to be added to the output table which will contain the unique part of the location (as specified in the input parameter(s)) of the input table from which each row originated. If not null, parameters will also be added to the output table giving the pre- and post-fix string common to all the locations. For example, if the input tables are "/data/cat_a1.fits" and "/data/cat_b2.fits" then the output table will contain a new column <colname> which takes the value "a1" for rows from the first table and "b2" for rows from the second, and new parameters "<colname>_prefix" and "<colname>_postfix" with the values "/data/cat_" and ".fits" respectively.

B.24.2 Examples

Here are some examples of `tcat`:

```
stilts tcat ifmt=ascii in=t1.txt in=t2.txt in=t3.txt out=table.txt
```

Concatenates the three named ASCII format tables to produce an output table. All three must have compatible numbers and types of columns.

```
stilts tcat ifmt=ascii in="t1.txt t2.txt t3.txt" out=table.txt
```

Has exactly the same effect as the previous example.

```
stilts tcat ifmt=ascii in=@inlist.lis out=table.txt
```

This will have the same effect as the previous two examples if a file name "inlist.lis" in the current directory contains three lines, "t1.txt", "t2.txt" and "t3.txt".

```
stilts tcat in=r368776.fits#1 in=r368776#2 in=r368776.fits#3 in=r368776.fits#4
out=r368776_all.fits
```

Concatenates the contents of four tables (the first four extension HDUs) from a multi-extension FITS file to produce a single FITS table. Many Unix shells (csh, bash) will allow you to list the input files using the following shorthand: "in=r368776.fits#{1,2,3,4}".

```
stilts tcat in=r368776.fits multi=true out=r368776_all.fits
```

Concatenates all the tables in the named file together. Setting `multi=true` means that instead of picking the first table from each named `in` table, all tables will be selected. So, if the input FITS file in this example has just four table HDUs, then this example does exactly the same as the previous one, but with less typing. The same thing works with multi-TABLE VOTable documents, but most other file formats (CSV etc) do not have the facility for storing multiple tables in a single file.

```
stilts tcat in=r368776.fits multi=true out=r368776_all.fits
icmd=progress seqcol=ID
```

Does the same as the previous example with a couple of additions. Firstly, progress through each of the input files will be reported to the console. Secondly, an additional column "ID" will be appended to the output which contains 1 for all the rows from the first input table, 2 for the rows from the second one and so on.

```
stilts tcat in='rA.csv rB.csv rC.csv' ifmt=csv \
  icmd='keepcols "RA DEC FLUX"' icmd='sorthead 10 FLUX' \
  ocmd='sort FLUX'
```

Takes the 10 rows with highest FLUX values from each of three input tables (in comma-separated value format) and joins them together to produce a 30-row output table. This is then sorted in FLUX order, and the resulting table is output to the console in text format. Only the columns RA, DEC and FLUX are output; any other columns are discarded. The input tables don't need to have identical forms to each other, but each must have at least an RA, DEC and FLUX column.

```
stilts tcat in=vizier.xml multi=true
  icmd='keepcols "ucd$RECORD ucd$POS_EQ_RA_MAIN ucd$POS_EQ_DEC_MAIN"'
  uloccol=TID out=all.csv
```

This processes a VOTable file which may have multiple TABLEs in it, but for which each of the tables is known to have columns with the UCDs RECORD, POS_EQ_RA_MAIN and POS_EQ_DEC_MAIN (this is typical of VOTables retrieved from CDS's Vizier service). It retains only those columns from each table and writes the result as a single concatenated table to a CSV file.

B.25 tcatn: Concatenates multiple tables

tcatn is a tool for concatenating a number of tables one after the other. Each table can be manipulated separately prior to the concatenation. If you have two tables T1 and T2 which contain similar columns, and you want to treat them as a single table, you can use tcatn to produce a new table whose metadata (row headings etc) comes from T1 and whose data consists of all the rows of T1 followed by all the rows of T2.

For this concatenation to make sense, each column of T1 must be compatible with the corresponding column of T2 - they must have compatible types and, presumably, meanings. If this is not the case for the tables that you wish to concatenate, for instance the columns are in different orders, or the units differ between a column in T1 and its opposite number in T2, you can use the icmd1 and/or icmd2 parameters to manipulate the input tables so that the column sequences are compatible. See Appendix B.25.2 for some examples.

If the tables are similar to each other (same format, same columns, same preprocessing stages required if any), you may find it easier to use tcat instead.

B.25.1 Usage

The usage of tcatn is

```
stilts <stilts-flags> tcatn nin=<count> ifmtN=<in-format> inN=<tableN>
  icmdN=<cmds> ocmd=<cmds>
  omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|tos
  out=<out-table> ofmt=<out-format>
  seqcol=<colname> loccol=<colname>
  uloccol=<colname> countrows=true|false
```

If you don't have the stilts script installed, write "java -jar stilts.jar" instead of "stilts" - see Section 3. The available <stilts-flags> are listed in Section 2.1. For programmatic invocation, the Task class for this command is uk.ac.starlink.ttools.task.TableCatN.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

countrows = true|false (*Boolean*)

Whether to count the rows in the table before starting the output. This is essentially a tuning parameter - if writing to an output format which requires the number of rows up front (such as normal FITS) it may result in skipping the number of passes through the input files required for processing. Unless you have a good understanding of the internals of the software, your best bet for working out whether to set this true or false is to try it both ways

[Default: false]

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on input table #N as specified by parameter *inN*, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file *filename* to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of input table #N as specified by parameter *inN*. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value (auto) (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: (auto)]

inN = <tableN> (*StarTable*)

The location of input table #N. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the *ifmtN* parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("<syscmd" or "syscmd|"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

loccol = <colname> (*String*)

Name of a column to be added to the output table which will contain the location (as specified in the input parameter(s)) of the input table from which each row originated.

nin = <count> (*Integer*)

The number of input tables for this task. For each of the input tables N there will be associated parameters *ifmtN*, *inN* and *icmdN*.

ocmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline

which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (String)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

**omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(ProcessingMode)**

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (TableConsumer)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

seqcol = <colname> (String)

Name of a column to be added to the output table which will contain the sequence number of the input table from which each row originated. This column will contain 1 for the rows from the first concatenated table, 2 for the second, and so on.

uloccol = <colname> (String)

Name of a column to be added to the output table which will contain the unique part of the location (as specified in the input parameter(s)) of the input table from which each row originated. If not null, parameters will also be added to the output table giving the pre- and post-fix string common to all the locations. For example, if the input tables are

"/data/cat_a1.fits" and "/data/cat_b2.fits" then the output table will contain a new column `<colname>` which takes the value "a1" for rows from the first table and "b2" for rows from the second, and new parameters "`<colname>_prefix`" and "`<colname>_postfix`" with the values "/data/cat_" and ".fits" respectively.

B.25.2 Examples

Here are some examples of `tcatn`:

```
stilts tcatn nin=2 in1=obs1.fits in2=obs2.fits out=combined.fits
```

Concatenates two similar observation catalogues to form a combined one. In this case, both input and output tables are FITS files.

```
stilts tcatn nin=3 omode=stats in1=obs1.txt ifmt1=ascii
                                in2=obs2.xml ifmt2=votable
                                in3=obs3.fit ifmt3=fits
```

Three catalogues with similar forms but in different data formats are joined. Instead of writing the result to an output file, the resulting joined catalogue is examined to calculate its statistics, which are written to standard output.

```
stilts tcatn nin=2 in1=survey.vot.gz ifmt2=csv in2=more_data.csv
            icmd1='addskycoords fk5 galactic RA2000 DEC2000 GLON GLAT' \
            icmd1='keepcols "OBJ_ID GLON GLAT"' \
            icmd2='keepcols "ident gal_long gal_lat"' \
            loccol=FILENAME
            omode=topcat
```

In this case we are trying to concatenate results from two tables which are quite dissimilar to each other. In the first place, one is a VOTable (no `ifmt1` parameter is required since VOTables can be detected automatically), and the other is a comma-separated-values file (for which the `ifmt2=csv` parameter must be given). In the second place, the column structure of the two tables may be quite different. By pre-processing the two tables using the `icmd1` & `icmd2` parameters, we produce in each case an input table which consists of three columns of compatible types and meanings: an integer identifier and floating point galactic longitude and latitude coordinates. The second table contains such columns to start with, but the first table requires an initial step to convert FK5 J2000.0 coordinates to galactic ones. `tcatn` joins the two doctored tables together, to produce a table which contains only these three columns, with all the rows from both input tables, and sends the result directly to a new or running instance of TOPCAT. An additional column named `FILENAME` is appended to the table before sending it; this contains "survey.vot.gz" for all the columns from the first table and "more_data.csv" for all the columns from the second one.

B.26 `tcopy`: Converts between table formats

`tcopy` is a table copying tool. It simply copies a table from one place to another, but since you can specify the input and output formats as desired, it works as a converter from any of the supported input formats (Section 5.2.1) to any of the supported output formats (Section 5.2.2).

`tcopy` is just a stripped-down version of `tpipe` - it doesn't do anything that `tpipe` can't, but the usage is slightly simplified. It is provided as a drop-in replacement for the old `tablecopy` (`uk.ac.starlink.table.TableCopy`) tool which was supplied with earlier versions of STIL and TOPCAT - it has the same arguments and behaviour as `tablecopy`, but is implemented somewhat differently and will in some cases be more efficient.

B.26.1 Usage

The usage of `tcopy` is

```
stilts <stilts-flags> tcopy ifmt=<in-format> ofmt=<out-format>
[in=]<table> [out=]<out-table>
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TableCopy`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

ifmt = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value `"(auto)"` (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

[Default: `(auto)`]

out = <out-table> (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value `"-"` (the default) the output table will be written to standard output.

[Default: `-`]

B.26.2 Examples

Here are some examples of `tcopy` in use:

```
stilts tcopy stars.fits stars.xml
```

Copies a FITS table to a VOTable. Since no input format is specified, the format is automatically detected (FITS is one of the formats for which this is possible). Since no output format is specified, the `stars.xml` filename is examined to make a guess at the kind of output to write: the `.xml` ending is taken to mean a TABLEDATA-encoded VOTable.

```
stilts tcopy stars.fits stars.xml ifmt=fits ofmt=votable
```

Does the same as the previous example, but the input and output formats have been specified explicitly.

```
stilts tcopy ofmt=text http://remote.host/data/vizer.xml.gz#4 -
```

Prints the contents of a remote, compressed VOTable to the terminal in a human-readable form. The #4 at the end of the URL indicates that the data from the fifth TABLE element in the remote document are to be used. The gzip compression of the table is taken care of automatically.

```
stilts tcopy ifmt=csv ofmt=latex spec.csv
```

Converts a comma-separated values file to a LaTeX table environment, writing the result to standard output.

```
stilts -classpath /usr/local/jars/pg73jdbc3.jar \
-Djdbc.drivers=org.postgresql.Driver \
tcopy in="jdbc:postgresql://localhost/imsim#SELECT ra, dec, Imag FROM dqc" \
ofmt=fits wfslist.cat
```

Makes an SQL query on a PostgreSQL database and writes the results to a FITS file. The whole command is shown here, to show that the classpath is augmented to include the PostgreSQL driver class, and the driver class is named using the `jdbc.drivers` system property. As you can see, using SQL from Java is a bit fiddly, and there are other ways to perform this setup than on the command line - see Section 3.4 and `tpipe`'s `omode=toSQL` output mode.

B.27 `tcube`: Calculates N-dimensional histograms

`tcube` constructs an N-dimensional histogram, or density map, from N columns of an input table, and writes it out as an N-dimensional data cube. The parameters you supply define which N numeric columns of the input table you want to use and the dimensions (bounds and pixel sizes) of the output grid. Each table row then defines a point in N-dimensional space. The program goes through each row, and if the point that row defines falls within the bounds of the output grid you have defined, increments the value associated with the corresponding pixel. The resulting N-dimensional array, whose pixel values represent a count of the rows associated with that region of the N-dimensional space, is then written out as a FITS file. In one dimension, this gives you a normal histogram of a given variable. In two dimensions it might typically be used to plot the density on the sky of objects from a catalogue.

As with some of the other generic table commands, you can perform extensive pre-processing on the input table by use of the `icmd` parameter before the actual cube counts are calculated.

B.27.1 Usage

The usage of `tcube` is

```
stilts <stilts-flags> tcube cols=<col-id> ... ifmt=<in-format>
istream=true|false icmd=<cmds>
```

```

bounds=[<lo>]:[<hi>] ... binsizes=<size> ...
nbins=<num> ... out=<out-file>
otype=byte|short|int|long|float|double
scale=<col-id>
[in=]<table>

```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TableCube`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

binsizes = <size> ... (*String[]*)

Gives the extent of the data bins (cube pixels) in each dimension in data coordinates. The form of the value is a space-separated list of values, giving a list of extents for the first, second, ... dimension. Either this parameter or the `nbins` parameter must be supplied.

bounds = [<lo>]:[<hi>] ... (*String[]*)

Gives the bounds for each dimension of the cube in data coordinates. The form of the value is a space-separated list of words, each giving an optional lower bound, then a colon, then an optional upper bound, for instance `"1:100 0:20"` to represent a range for two-dimensional output between 1 and 100 of the first coordinate (table column) and between 0 and 20 for the second. Either or both numbers may be omitted to indicate that the bounds should be determined automatically by assessing the range of the data in the table. A null value for the parameter indicates that all bounds should be determined automatically for all the dimensions.

If any of the bounds need to be determined automatically in this way, two passes through the data will be required, the first to determine bounds and the second to populate the cube.

cols = <col-id> ... (*String[]*)

Columns to use for this task. One or more `<col-id>` elements, separated by spaces, should be given. Each one represents a column in the table, using either its name or index.

The number of columns listed in the value of this parameter defines the dimensionality of the output data cube.

icmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (`;`). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character `'@'`. Thus a value of `"@filename"` causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istream = true|false (*Boolean*)

If set true, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

nbins = <num> ... (*String[]*)

Gives the number of bins (cube pixels) in each dimension. The form of the value is a space-separated list of integers, giving the number of pixels for the output cube in the first, second, ... dimension. Either this parameter or the `binsizes` parameter must be supplied.

otype = byte|short|int|long|float|double (*Class*)

The type of numeric value which will fill the output array. If no selection is made, the output type will be determined automatically as the shortest type required to hold all the values in the array. Currently, integers are always signed (no BSCALE/BZERO), so for instance the largest value that can be recorded in 8 bits is 127.

out = <out-file> (*uk.ac.starlink.util.Destination*)

The location of the output file. This is usually a filename to write to. If it is equal to the special value "-" the output will be written to standard output.

The output cube is currently written as a single-HDU FITS file.

[Default: -]

scale = <col-id> (*String*)

Optionally gives a value by which the count in each bin is scaled. If this value is null (the default) then for each row that falls within the bounds of a pixel, the pixel value will be incremented by 1. If a column ID is given, then instead of 1 being added, the value of that column for the row in question is added. The effect of this is that the output image contains the mean of the given column for the rows corresponding to each pixel rather than just a count of them.

B.27.2 Examples

```
stilts tcube in=2QZ_6QZ_pubcat.fits out=ccm.fits \
  cols='Bj_R U_Bj Bj' binsizes='0.05 0.05 0.5' bounds='-2:1 -3:2 :'
```

Calculates a 3-dimensional colour-colour-magnitude grid from three existing columns in a table. The bin (pixel) sizes are specified. The data bounds are specified explicitly for the (first two) colour dimensions, but for the (third) magnitude dimension it is determined from the minimum and maximum values the data in that column of the table. The output is a three-dimensional FITS cube.

```
stilts tcube in=iras_psc.vot out=iras_psc_map.fits \
            icmd='addskycoords fk5 galactic ra dec glat glon' \
            cols='glat glon' nbins='400 200'
```

Calculates a map of object densities in galactic coordinates from a catalogue of IRAS point sources. The output is a two-dimensional FITS image representing the sky in galactic coordinates. Bounds are determined automatically from the data, and the number of pixels in each dimension (400 in latitude and 200 in longitude) are specified, which means that the pixel sizes don't have to be. Since the input table contains sky positions in equatorial coordinates rather than galactic ones, the `addskycoords` filter is used to preprocess the data before the cube generation step (see Section 6.1).

B.28 `tloop`: Generates a single-column table from a loop variable

`tloop` generates a one-column table where the values in the column are effectively populated from a for loop (start, end, step). This may be useful as it is, or it can be postprocessed with `ocmd` parameters to add more columns etc.

B.28.1 Usage

The usage of `tloop` is

```
stilts <stilts-flags> tloop ocmd=<cmds>
                             omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosc
                             out=<out-table> ofmt=<out-format>
                             forcefloat=true|false
                             [colname=]<value> [start=]<float-value>
                             [end=]<float-value> [step=]<float-value>
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TableLoop`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

colname = <value> (*String*)

Gives the name of the single column produced by this command.

[Default: `i`]

end = <float-value> (*Double*)

Gives the value which the loop variable will not exceed. Exceeding is in the positive or negative sense according to the sense of the `step` parameter, as usual for a `for`-type loop.

forcefloat = true|false (*Boolean*)

Affects the data type of the loop variable column. If true, the column is always floating point. If false, and if the other parameters are all of integer type, the column will be an integer column.

[Default: `false`]

ocmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (`;`). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

**omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(*ProcessingMode*)**

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

start = <float-value> (*Double*)

Gives the starting value of the loop variable. This will be the value in the first row of the table.

[Default: 0.0]

step = <float-value> (*Double*)

Amount by which the loop variable will be incremented at each iteration, i.e. each table row.

[Default: 1.0]

B.28.2 Examples

```
stilts tloop COUNTER 0 1000
```

Generates a table with a single column, named `COUNTER`, and a thousand rows. The value in the first row is 0 and in the last row is 999. The table is written to standard output.

```
stilts tloop time 0 10 0.25 out=times.csv
```

Generates a table with one column `time` counting from 0 to 9.75 in steps of 0.25. Output is to a CSV file. The parameters here are specified in order, but could equivalently be given by name: `"stilts tloop var=time start=0 end=10 step=0.26"`.

```
stilts tloop x start=1 end=11 ocmd='addcol x2 x*x' ocmd='addcol x3 x*x*x'
      ocmd='stats name sum'
```

Generates a table with a column `x` running from 1 to 10 inclusive. The `addcol` filters then append two further columns, giving the squares and cubes of these values respectively, giving a table of 10 rows and 3 columns. Finally this table is piped through a `stats` filter to calculate the sums of the values, squares and cubes in this range.

B.29 tjoin: Joins multiple tables side-to-side

`tjoin` performs a trivial side-by-side join of multiple tables. The N'th row of the output table consists of the N'th row of the first input table, followed by the N'th row of the second input table, ... and so on. It is suitable if you want to amalgamate two or more tables whose row orderings correspond exactly to each other.

For the (more usual) case in which the rows of the tables to be joined are not already in the right order, use one of the crossmatching commands (Section 7).

B.29.1 Usage

The usage of `tjoin` is

```
stilts <stilts-flags> tjoin nin=<count> ifmtN=<in-format> inN=<tableN>
      icmdN=<cmds> ocmd=<cmds>
      omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|tose
      out=<out-table> ofmt=<out-format>
      fixcols=none|dups|all suffixN=<label>
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TableJoinN`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

fixcols = none|dups|all (*Fixer*)

Determines how input columns are renamed before use in the output table. The choices are:

- `none`: columns are not renamed
- `dups`: columns which would otherwise have duplicate names in the output will be renamed to indicate which table they came from
- `all`: all columns will be renamed to indicate which table they came from

If columns are renamed, the new ones are determined by `suffix*` parameters.

[Default: `dups`]

icmdN = <cmds> (ProcessingStep[])

Specifies processing to be performed on input table #N as specified by parameter `inN`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (String)

Specifies the format of input table #N as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <tableN> (StarTable)

The location of input table #N. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

nin = <count> (Integer)

The number of input tables for this task. For each of the input tables N there will be associated parameters `ifmtN`, `inN` and `icmdN`.

ocmd = <cmds> (ProcessingStep[])

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (String)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value `"(auto)"` (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

`omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui`
(ProcessingMode)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (`<mode-args>`) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

`out = <out-table>` *(TableConsumer)*

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

`suffixN = <label>` *(String)*

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from table N.

[Default: _N]

B.29.2 Examples

Here are some examples of using `tjoin`

```
stilts tjoin nin=2 in1=positions.fit in2=fluxes.fits out=combined.fits
```

Takes two input FITS files and sticks them together side by side, writing the result as a third FITS file. The output will have the same number of rows as each of the input catalogues, and a number of columns equal to the sum of those in the two input catalogues.

```
stilts tjoin nin=3 fixcols=all \
    ifmt1=ascii in1=t1.txt suffix1=_T1 \
    ifmt2=ascii in2=t2.txt suffix2=_T2 \
    ifmt3=ascii in3=t3.txt suffix3=_T3 \
    ocmd='select FLAG_T1==0' \
```

omode=stats

This joins three ascii tables together. Each column of the output table is renamed by appending a string to it ("_T1" for the first table, "_T2" for the second...). Only those rows of the output for which the FLAG column in the first input table, and hence the FLAG_T1 column in the output table, have the value zero are selected. Statistics are calculated for all the columns of these selected rows, and written to the output.

B.30 tmatch1: Performs a crossmatch internal to a single table

tmatch1 performs efficient and flexible crossmatching between the rows of a single table. It can match rows on the basis of their relative position in the sky, or alternatively using many other criteria such as separation in in some isotropic or anisotropic Cartesian space, identity of a key value, or some combination of these; the full range of match criteria is discussed in Section 7.1.

The basic task performed by the intra-table matcher is to identify groups of rows within the table which match each other. See Section 7.2 for an explanation of exactly what constitutes a match group. The result of identifying these groups is expressed as an output table in one of a variety of ways, specified by the `action` parameter. These options include marking group membership in added columns and eliminating some or all rows which form part of a match group.

B.30.1 Usage

The usage of tmatch1 is

```
stilts <stilts-flags> tmatch1 matcher=<matcher-name> params=<match-params>
                                tuning=<tuning-params> values=<expr-list>
                                action=identify|keep0|keep1|wide2|wideN
                                progress=none|log|profile ifmt=<in-format>
                                istream=true|false icmd=<cmds> ocmd=<cmds>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|t
                                out=<out-table> ofmt=<out-format>
                                [in=<table>]
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TableMatch1`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

action = identify|keep0|keep1|wide2|wideN (*Match1Type*)

Determines the form of the table which will be output as a result of the internal match.

- **identify**: The output table is the same as the input table except that it contains two additional columns, `GroupID` and `GroupSize`, following the input columns. Each group of rows which matched is assigned a unique integer, recorded in the `GroupID` column, and the size of each group is recorded in the `GroupSize` column. Rows which don't match any others (singles) have null values in both these columns.
- **keep0**: The result is a new table containing only "single" rows, that is ones which don't match any other rows in the table. Any other rows are thrown out.
- **keep1**: The result is a new table in which only one row (the first in the input table order) from each group of matching ones is retained. A subsequent intra-table match with the same criteria would therefore show no matches.
- **wideN**: The result is a new "wide" table consisting of matched rows in the input table stacked next to each other. Only groups of exactly N rows in the input table are used to form the output table; each row of the output table consists of the columns of the first

group member, followed by the columns of the second group member and so on. The output table therefore has N times as many columns as the input table. The column names in the new table have `_1`, `_2`, ... appended to them to avoid duplication.

[Default: `identify`]

icmd = `<cmds>` (*ProcessingStep[]*)

Specifies processing to be performed on the input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (`;`). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character `'@'`. Thus a value of `"@filename"` causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = `<in-format>` (*String*)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in = `<table>` (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istream = `true|false` (*Boolean*)

If set `true`, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as `VOTable`).

[Default: `false`]

matcher = `<matcher-name>` (*MatchEngine*)

Defines the nature of the matching that will be performed. Depending on the name supplied, this may be positional matching using celestial or Cartesian coordinates, exact matching on the value of a string column, or other things. A list and explanation of the available matching algorithms is given in Section 7.1. The value supplied for this parameter determines the meanings of the values required by the `params`, `values*` and `tuning` parameter(s).

[Default: `sky`]

ocmd = `<cmds>` (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(*ProcessingMode*)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

params = <match-params> (*String[]*)

Determines the parameters of this match. This is typically one or more tolerances such as error radii. It may contain zero or more values; the values that are required depend on the match type selected by the `matcher` parameter. If it contains multiple values, they must be separated

by spaces; values which contain a space can be 'quoted' or "quoted".

progress = none|log|profile (*String*)

Determines whether information on progress of the match should be output to the standard error stream as it progresses. For lengthy matches this is a useful reassurance and can give guidance about how much longer it will take. It can also be useful as a performance diagnostic.

The options are:

- none: no progress is shown
- log: progress information is shown
- profile: progress information and limited time/memory profiling information are shown

[Default: log]

tuning = <tuning-params> (*String[]*)

Tuning values for the matching process, if appropriate. It may contain zero or more values; the values that are permitted depend on the match type selected by the `matcher` parameter. If it contains multiple values, they must be separated by spaces; values which contain a space can be 'quoted' or "quoted". If this optional parameter is not supplied, sensible defaults will be chosen.

values = <expr-list> (*String[]*)

Defines the values from the input table which are used to determine whether a match has occurred. These will typically be coordinate values such as RA and Dec and perhaps some per-row error values as well, though exactly what values are required is determined by the kind of match as determined by `matcher`. Depending on the kind of match, the number and type of the values required will be different. Multiple values should be separated by whitespace; if whitespace occurs within a single value it must be 'quoted' or "quoted". Elements of the expression list are commonly just column names, but may be algebraic expressions calculated from zero or more columns as explained in Section 10.

B.30.2 Examples

Here are some examples of using `tmatch1`

```
stilts tmatch1 matcher=sky values="RA2000 DE2000" params=20 \
      action=keep0 in=crowded.vot out=sparse.vot
```

Copies an input catalogue "crowded.vot" to an output catalogue "sparse.vot", but omitting any objects (rows) which are within 20 arcsec of other objects. The output catalogue will contain no near neighbours.

```
stilts tmatch1 matcher=skyerr values="RA2000 DE2000 RADIUS*4" params=40 \
      action=keep0 in=crowded.vot out=sparse.vot
```

This is similar to the previous example, but uses the `skyerr` matcher which determines the proximity threshold on a row-by-row basis from values in the table - in this case 4 times the value of the `RADIUS` column (this value must be in arc seconds). The `params=40` value does not affect the result, but it gives the algorithm an idea of the rough scale of object separation.

```
stilts tmatch1 matcher=3d values="XPIX YPIX ZPIX" params=10 action=identify \
      in=state.fit ocmd='select GroupSize>3' out=groups3+.fit
```

Uses the "3d" matcher to identify groups of objects in terms of their proximity in a 3-dimensional Cartesian space, with positions given by the `XPIX`, `YPIX` and `ZPIX` columns in the input table. The `action=identify` parameter means that the input table is written out with the same rows, but with additional columns indicating which rows are associated with each

other. One of these columns, "GroupSize" gives the number of objects in each group. The postprocessing filter `ocmd='select GroupSize>3'` selects only those rows which are part of groups of three objects or larger; singletons and pairs are discarded before writing the output file.

```
stilts tmatch1 matcher=sky values="ra dec" params=3 action=wide2 \
      ocmd='keepcols "id_1 ra_1 dec_1 id_2 ra_2 dec_2"'
      in=galaxy.fits out=binaries.txt ofmt=ascii
```

Identifies pairs of objects within 3 arcsec of each other from an input catalogue. Singles, and groups of three or more, will be discarded. The output table generated is a double-width version of the input table with pairs of objects next to each other on the same row. Here, the `ocmd` post-processing filter discards all of the columns except the identifiers and sky positions for each object. The output is to a text file.

B.31 tmatch2: Crossmatches 2 tables using flexible criteria

`tmatch2` is an efficient and highly configurable tool for crossmatching pairs of tables. It can match rows between tables on the basis of their relative position in the sky, or alternatively using many other criteria such as separation in some isotropic or anisotropic Cartesian space, identity of a key value, or some combination of these; the full range of match criteria is discussed in Section 7.1. You can choose whether you want to identify all the matches or only the closest, and what form the output table takes, for instance matched rows only, or all rows from one or both tables, or only the unmatched rows.

If you simply want to match two tables based on sky position with a fixed maximum separation, you may find the `tskymatch2` command easier to use.

Note: the `duptag1` and `duptag2` parameters have been replaced at version 1.4 by `suffix1` and `suffix2` for consistency with other table join tasks.

B.31.1 Usage

The usage of `tmatch2` is

```
stilts <stilts-flags> tmatch2 ifmt1=<in-format> ifmt2=<in-format>
                             icmd1=<cmds> icmd2=<cmds> ocmd=<cmds>
                             omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|t
                             out=<out-table> ofmt=<out-format>
                             matcher=<matcher-name> values1=<expr-list>
                             values2=<expr-list> params=<match-params>
                             tuning=<tuning-params>
                             join=1and2|1or2|all1|all2|1not2|2not1|1xor2
                             find=all|best|best1|best2
                             fixcols=none|dups|all suffix1=<label>
                             suffix2=<label> scorecol=<col-name>
                             progress=none|log|profile
                             [in1=]<table1> [in2=]<table2>
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TableMatch2`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

find = all|best|best1|best2 (*PairMode*)

Determines what happens when a row in one table can be matched by more than one row in

the other table. The options are:

- **all**: All matches. Every match between the two tables is included in the result. Rows from both of the input tables may appear multiple times in the result.
- **best**: Best match, symmetric. The best pairs are selected in a way which treats the two tables symmetrically. Any input row which appears in one result pair is disqualified from appearing in any other result pair, so each row from both input tables will appear in at most one row in the result.
- **best1**: Best match for each Table 1 row. For each row in table 1, only the best match from table 2 will appear in the result. Each row from table 1 will appear a maximum of once in the result, but rows from table 2 may appear multiple times.
- **best2**: Best match for each Table 2 row. For each row in table 2, only the best match from table 1 will appear in the result. Each row from table 2 will appear a maximum of once in the result, but rows from table 1 may appear multiple times.

The differences between **best**, **best1** and **best2** are a bit subtle. In cases where it's obvious which object in each table is the best match for which object in the other, choosing between these options will not affect the result. However, in crowded fields (where the distance between objects within one or both tables is typically similar to or smaller than the specified match radius) it will make a difference. In this case one of the asymmetric options (**best1** or **best2**) is usually more appropriate than **best**, but you'll have to think about which of them suits your requirements. The performance (time and memory usage) of the match may also differ between these options, especially if one table is much bigger than the other.

[Default: **best**]

fixcols = **none|dups|all** (*Fixer*)

Determines how input columns are renamed before use in the output table. The choices are:

- **none**: columns are not renamed
- **dups**: columns which would otherwise have duplicate names in the output will be renamed to indicate which table they came from
- **all**: all columns will be renamed to indicate which table they came from

If columns are renamed, the new ones are determined by **suffix*** parameters.

[Default: **dups**]

icmd1 = **<cmds>** (*ProcessingStep[]*)

Specifies processing to be performed on the first input table as specified by parameter **in1**, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file **filename** to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

icmd2 = **<cmds>** (*ProcessingStep[]*)

Specifies processing to be performed on the second input table as specified by parameter **in2**, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file **filename** to be read for a list of filter

commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt1 = <in-format> (String)

Specifies the format of the first input table as specified by parameter `in1`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

ifmt2 = <in-format> (String)

Specifies the format of the second input table as specified by parameter `in2`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in1 = <table1> (StarTable)

The location of the first input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmt1` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

in2 = <table2> (StarTable)

The location of the second input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmt2` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

join = 1and2|1or2|all1|all2|1not2|2not1|1xor2 (JoinType)

Determines which rows are included in the output table. The matching algorithm determines which of the rows from the first table correspond to which rows from the second. This parameter determines what to do with that information. Perhaps the most obvious thing is to write out a table containing only rows which correspond to a row in both of the two input tables. However, you may also want to see the unmatched rows from one or both input tables, or rows present in one table but unmatched in the other, or other possibilities. The options are:

- `1and2`: An output row for each row represented in both input tables (INNER JOIN)
- `1or2`: An output row for each row represented in either or both of the input tables (FULL OUTER JOIN)

- `all1`: An output row for each matched or unmatched row in table 1 (LEFT OUTER JOIN)
- `all2`: An output row for each matched or unmatched row in table 2 (RIGHT OUTER JOIN)
- `1not2`: An output row only for rows which appear in the first table but are not matched in the second table
- `2not1`: An output row only for rows which appear in the second table but are not matched in the first table
- `1xor2`: An output row only for rows represented in one of the input tables but not the other one

[Default: `1and2`]

matcher = <matcher-name> (*MatchEngine*)

Defines the nature of the matching that will be performed. Depending on the name supplied, this may be positional matching using celestial or Cartesian coordinates, exact matching on the value of a string column, or other things. A list and explanation of the available matching algorithms is given in Section 7.1. The value supplied for this parameter determines the meanings of the values required by the `params`, `values*` and `tuning` parameter(s).

[Default: `sky`]

ocmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(*ProcessingMode*)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- `out`
- `meta`
- `stats`
- `count`

- `cgi`
- `discard`
- `topcat`
- `samp`
- `plastic`
- `tosql`
- `gui`

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: `out`]

out = `<out-table>` (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value `"-"` (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of `"out"`.

[Default: `-`]

params = `<match-params>` (*String[]*)

Determines the parameters of this match. This is typically one or more tolerances such as error radii. It may contain zero or more values; the values that are required depend on the match type selected by the `matcher` parameter. If it contains multiple values, they must be separated by spaces; values which contain a space can be 'quoted' or "quoted".

progress = `none|log|profile` (*String*)

Determines whether information on progress of the match should be output to the standard error stream as it progresses. For lengthy matches this is a useful reassurance and can give guidance about how much longer it will take. It can also be useful as a performance diagnostic.

The options are:

- `none`: no progress is shown
- `log`: progress information is shown
- `profile`: progress information and limited time/memory profiling information are shown

[Default: `log`]

scorecol = `<col-name>` (*String*)

Gives the name of a column in the output table to contain the "match score" for each pairwise match. The meaning of this column is dependent on the chosen `matcher`, but it typically represents a distance of some kind between the two matching points. If a null value is chosen, no score column will be inserted in the output table. The default value of this parameter depends on `matcher`.

[Default: `Score`]

suffix1 = `<label>` (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from table 1.

[Default: `_1`]

suffix2 = `<label>` (*String*)

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from table 2.

[Default: `_2`]

tuning = `<tuning-params>` (*String[]*)

Tuning values for the matching process, if appropriate. It may contain zero or more values; the

values that are permitted depend on the match type selected by the `matcher` parameter. If it contains multiple values, they must be separated by spaces; values which contain a space can be 'quoted' or "quoted". If this optional parameter is not supplied, sensible defaults will be chosen.

values1 = <expr-list> (String[])

Defines the values from table 1 which are used to determine whether a match has occurred. These will typically be coordinate values such as RA and Dec and perhaps some per-row error values as well, though exactly what values are required is determined by the kind of match as determined by `matcher`. Depending on the kind of match, the number and type of the values required will be different. Multiple values should be separated by whitespace; if whitespace occurs within a single value it must be 'quoted' or "quoted". Elements of the expression list are commonly just column names, but may be algebraic expressions calculated from zero or more columns as explained in Section 10.

values2 = <expr-list> (String[])

Defines the values from table 2 which are used to determine whether a match has occurred. These will typically be coordinate values such as RA and Dec and perhaps some per-row error values as well, though exactly what values are required is determined by the kind of match as determined by `matcher`. Depending on the kind of match, the number and type of the values required will be different. Multiple values should be separated by whitespace; if whitespace occurs within a single value it must be 'quoted' or "quoted". Elements of the expression list are commonly just column names, but may be algebraic expressions calculated from zero or more columns as explained in Section 10.

B.31.2 Examples

Here are some examples of using `tmatch2`

```
stilts tmatch2 in1=obs_v.xml in2=obs_i.xml out=obs_iv.xml \
    matcher=sky values1="ra dec" values2="ra dec" params="2"
```

Takes two input catalogues (VOTables), one with observations in the V band and the other in the I band, and performs a match to find objects within 2 arcseconds of each other. The result is a new table containing only rows where a match was found.

```
stilts tmatch2 survey.fits ifmt2=csv mycat.csv \
    icmdl1='addskycoords fk4 fk5 RA1950 DEC1950 RA2000 DEC2000' \
    matcher=skyerr \
    params=10 values1="RA2000 DEC2000 POS_ERR" values2="RA DEC 0" \
    join=2not1 omode=count
```

Here a comma-separated-values file is being compared with a FITS catalogue representing some survey results. Positions in the survey catalogue use the FK4 B1950.0 system, and so a preprocessing step is inserted to create new position columns in the first input table using the FK5 J2000.0 system, which is what the other input table uses. The survey catalogue contains a `POS_ERR` column which gives the positional uncertainty of its entries, so the `skyerr` matcher is used, which takes account of this; the third entry in the `values1` parameter is the `POS_ERR` column (in arcsec). Since the second input table has no positional uncertainty information, 0 is used as the third entry in `values2`. The `params` gives a rough idea of the scale of the object separations, but its value does not affect the result. The join type is `2not1`, which means the output table will only contain those entries which are in the second input table but not in the first one. The output table is not stored, but the number of rows it contains (the number of objects represented in the CSV file but not the survey) is written to the screen.

```
stilts tmatch2 ifmt1=ascii ifmt2=ascii in1=cat-a.txt in2=cat-b.txt \
    matcher=2d values1='X Y' values2='X Y' params=5 join=1and2 \
```

```
suffix1=_a suffix2=_b \
ocmd='addcol XDIFF X_a-X_b; addcol YDIFF Y_a-Y_b' \
ocmd='keepcols "XDIFF YDIFF"' omode=stats
```

Two ASCII-format catalogues are matched, where rows are considered to match if their X,Y positions are within 5 units of each other in some Cartesian space. The result of the matching operation is a table of all the matched rows, containing columns named X_a, Y_a, X_b and Y_b (along with any others in the input tables) - the `suffix*` parameters describe how the input X and Y columns are to be renamed to avoid duplicate column names in the output table. To this result are added two new columns, representing the X and Y positional difference between the rows from one input table and those from the other. The `keepcols` filter then throws all the other columns away, retaining only these difference columns. The final two-column table is not stored anywhere, but (`omode=stats`) statistics including mean and standard deviation are calculated on its columns and displayed to the screen. Having done all this, you can examine the average X and Y differences between the two input tables for matched rows, and if they differ significantly from zero, you can conclude that there is a systematic error between the positions in the two input files.

```
stilts tmatch2 in1=mgc.fits in2=6dfgs.xml join=1and2 find=all \
matcher=sky+1d params='3 0.5' \
values1='ra dec bmag' values2='RA2000 DEC2000 B_MAG' \
out=pairs.fits
```

This performs a match with a matcher that combines `sky` and `1d` match criteria. This means that the only rows which match are those which are *both* within 3 arcsec of each other on the sky *and* within 0.5 blue magnitudes. Note that for both the `params` and the `values1` and `values2` parameters, the items for the `sky` matcher (RA and DEC) are listed first, followed by those for the `1d` matcher (in this case, blue magnitude).

B.32 tmatchn: Crossmatches multiple tables using flexible criteria

`tmatchn` performs efficient and flexible crossmatching between multiple tables. It can match rows on the basis of their relative position in the sky, or alternatively using many other criteria such as separation in in some isotropic or anisotropic Cartesian space, identity of a key value, or some combination of these; the full range of match criteria is discussed in Section 7.1.

Since the match criteria define what counts as a match between two objects, it is not immediately obvious what is meant by a multi-table match. In fact the command can work in one of two distinct modes, controlled by the `multimode` parameter. In `pairs` mode, one table (by default the first input table) is designated the reference table, and pair matches between each of the other tables and that one are identified. In `group` mode groups of objects from all the input tables are identified, as discussed in Section 7.2. Currently, in both cases an output matched row cannot contain more than one object from each input table. Options for output of multiple rows per input table per match may be forthcoming in future releases if there is demand.

`tmatchn` is intended for use with more than two input tables - see `tmatch1` and `tmatch2` for 1- and 2-table crossmatching respectively.

B.32.1 Usage

The usage of `tmatchn` is

```
stilts <stilts-flags> tmatchn nin=<count> ifmtN=<in-format> inN=<tableN>
icmdN=<cmds> ocmd=<cmds>
omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|t
out=<out-table> ofmt=<out-format>
multimode=pairs|group iref=<table-index>
```

```

matcher=<matcher-name> params=<match-params>
tuning=<tuning-params> valuesN=<expr-list>
joinN=default|match|nomatch|always
fixcols=none|dups|all suffixN=<label>
progress=none|log|profile

```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TableMatchN`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

fixcols = none|dups|all (*Fixer*)

Determines how input columns are renamed before use in the output table. The choices are:

- `none`: columns are not renamed
- `dups`: columns which would otherwise have duplicate names in the output will be renamed to indicate which table they came from
- `all`: all columns will be renamed to indicate which table they came from

If columns are renamed, the new ones are determined by `suffix*` parameters.

[Default: `dups`]

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on input table #N as specified by parameter `inN`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (`;`). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character `'@'`. Thus a value of `"@filename"` causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of input table #N as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <tableN> (*StarTable*)

The location of input table #N. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

iref = <table-index> (Integer)

If `multimode=pairs` this parameter gives the index of the table in the input table list which is to serve as the reference table (the one which must be matched by other tables). Ignored in other modes.

Row ordering in the output table is usually tidiest if the default setting of 1 is used (i.e. if the first input table is used as the reference table).

[Default: 1]

joinN = default|match|nomatch|always (MultiJoinType)

Determines which rows from input table N are included in the output table. The matching algorithm determines which of the rows in each of the input tables correspond to which rows in the other input tables, and this parameter determines what to do with that information.

The default behaviour is that a row will appear in the output table if it represents a match of rows from two or more of the input tables. This can be altered on a per-input-table basis however by choosing one of the non-default options below:

- `match`: Rows are included only if they contain an entry from input table N.
- `nomatch`: Rows are included only if they do not contain an entry from input table N.
- `always`: Rows are included if they contain an entry from input table N (overrides any `match` and `nomatch` settings of other tables).
- `default`: Input table N has no special effect on whether rows are included.

[Default: default]

matcher = <matcher-name> (MatchEngine)

Defines the nature of the matching that will be performed. Depending on the name supplied, this may be positional matching using celestial or Cartesian coordinates, exact matching on the value of a string column, or other things. A list and explanation of the available matching algorithms is given in Section 7.1. The value supplied for this parameter determines the meanings of the values required by the `params`, `values*` and `tuning` parameter(s).

[Default: sky]

multimode = pairs|group (String)

Defines what is meant by a multi-table match. There are two possibilities:

- `pairs`: Each output row corresponds to a single row of the *reference table* (see parameter `iref`) and contains entries from other tables which are pair matches to that. If a reference table row matches multiple rows from one of the other tables, only the best one is included.
- `group`: Each output row corresponds to a group of entries from the input tables which are mutually linked by pair matches between them. This means that although you can get from any entry to any other entry via one or more pair matches, there is no guarantee that any entry is a pair match with any other entry. No table has privileged status in this case. If there are multiple entries from a given table in the match group, an arbitrary one is chosen for inclusion (there is no unique way to select the best). See Section 7.2 for more discussion.

In the case of well-separated objects these modes will give the same results. For crowded fields however it will make a difference which is chosen.

Note that which rows actually appear in the output is also influenced by the `joinN` parameter.

[Default: pairs]

nin = <count> (Integer)

The number of input tables for this task. For each of the input tables N there will be associated parameters `ifmtN`, `inN` and `icmdN`.

ocmd = <cmds> (ProcessingStep[])

Specifies processing to be performed on the output table, after all other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ofmt = <out-format> (String)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "(auto)" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "out".

[Default: (auto)]

omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui (ProcessingMode)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (<mode-args>) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (TableConsumer)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "-" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "out".

[Default: -]

params = <match-params> (String[])

Determines the parameters of this match. This is typically one or more tolerances such as error radii. It may contain zero or more values; the values that are required depend on the match type selected by the `matcher` parameter. If it contains multiple values, they must be separated

by spaces; values which contain a space can be 'quoted' or "quoted".

progress = none|log|profile *(String)*

Determines whether information on progress of the match should be output to the standard error stream as it progresses. For lengthy matches this is a useful reassurance and can give guidance about how much longer it will take. It can also be useful as a performance diagnostic.

The options are:

- none: no progress is shown
- log: progress information is shown
- profile: progress information and limited time/memory profiling information are shown

[Default: log]

suffixN = <label> *(String)*

If the `fixcols` parameter is set so that input columns are renamed for insertion into the output table, this parameter determines how the renaming is done. It gives a suffix which is appended to all renamed columns from table N.

[Default: _N]

tuning = <tuning-params> *(String[])*

Tuning values for the matching process, if appropriate. It may contain zero or more values; the values that are permitted depend on the match type selected by the `matcher` parameter. If it contains multiple values, they must be separated by spaces; values which contain a space can be 'quoted' or "quoted". If this optional parameter is not supplied, sensible defaults will be chosen.

valuesN = <expr-list> *(String[])*

Defines the values from table N which are used to determine whether a match has occurred. These will typically be coordinate values such as RA and Dec and perhaps some per-row error values as well, though exactly what values are required is determined by the kind of match as determined by `matcher`. Depending on the kind of match, the number and type of the values required will be different. Multiple values should be separated by whitespace; if whitespace occurs within a single value it must be 'quoted' or "quoted". Elements of the expression list are commonly just column names, but may be algebraic expressions calculated from zero or more columns as explained in Section 10.

B.32.2 Examples

Here are some examples of using `tmatchn`

```
stilts tmatchn multimode=pairs nin=4 matcher=sky params=5 \
  in1=transients.txt ifmt1=ascii values1='alpha delta' \
  in2=2mass_virgo.fits values2='ra2000 dec2000' \
  in3=sdss_virgo.fits values3='ra dec' \
  in4=first_virgo.fits values4='pos_eq_ra pos_eq_dec' \
  out=matches.xml ofmt=votable-binary
```

Compares a text-format table "transients.txt" against each of three other catalogues covering the same region of sky, and outputs a table which contains a row for each row of "transients.txt" which matches (is within 5 arcsec) of an object in any of the other tables.

```
stilts tmatchn multimode=pairs nin=4 matcher=sky params=5 \
  in1=transients.txt ifmt1=ascii suffix1='_t' values1='alpha delta' \
  in2=2mass_virgo.fits suffix2='_2mass' values2='ra2000 dec2000' \
  in3=sdss_virgo.fits suffix3='_sdss' values3='ra dec' \
  in4=first_virgo.fits suffix4='_first' values4='pos_eq_ra pos_eq_dec' \
  fixcols=all join1=all \
  ocmd='keepcols "*_t designation_2mass SDSSName_sdss id_field_first"' \
```

```
out=matches.xml ofmt=votable-binary
```

Similar to the previous example but with some doctoring of what the output table will look like. The `fixcols=all` and `suffixN` assignments mean that all the columns from the input tables will be renamed for output by adding the given suffixes. The `keepcols` filter applied to the output table throws out all the columns except the ones from the reference table (`*_t`) and one column from each of the other table giving object identifiers. This output table will probably be easier to read (though contain less information) than that from the previous example). Additionally, the `join1=all` assignment means that the output table will have one row for each row of the reference table (`transients.txt`), even if no matches are found for it.

```
stilts tmatchn multimode=group nin=3 matcher=skyerr params=8 \
  in1=Hband.fits values='RA DEC SEEING*2' \
  in2=Jband.fits values='RA DEC SEEING*2' \
  in3=Kband.fits values='RA DEC SEEING*2' \
  omode=topcat
```

Performs a group-mode match. There is no reference table, so that an output row will result for each object which is represented in any two of the input catalogues. The match takes account of per-object errors equivalent to twice the recorded seeing, which is in the region of 8 arcsec. Note that this may not operate as expected if the catalogues contain multiple distinct objects too close (in comparison to the declared separation) to each other. The resulting matched table is sent directly to TOPCAT (if available).

B.33 `tmulti`: Writes multiple tables to a single container file

`tmulti` takes multiple input tables and writes them as separate tables to a single output container file. The container file must be of some format which can contain more than one table, for instance a FITS file (which can contain multiple extensions) or a VOTable document (which can contain multiple TABLE elements). Filtering may be performed on the tables prior to writing them. It is not necessary that all the tables are similar (e.g. that they all have the same type and number of columns), but the same processing commands will be applied to all of them. For more individual control, use the `tmultin` task.

B.33.1 Usage

The usage of `tmulti` is

```
stilts <stilts-flags> tmulti in=<table> [<table> ...] ifmt=<in-format>
                                multi=true|false istream=true|false
                                icmd=<cmds> out=<out-file> ofmt=<out-format>
```

If you don't have the `stilts` script installed, write "`java -jar stilts.jar`" instead of "`stilts`" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.MultiCopy`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

icmd = <cmds> (*ProcessingStep*[])

Specifies processing to be performed on each input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters ("`;`"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = <in-format> (String)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

The same format parameter applies to all the tables specified by `in`.

[Default: `(auto)`]

in = <table> [<table> ...] (TableProducer[])

Locations of the input tables. Either specify the parameter multiple times, or supply the input tables as a space-separated list within a single use.

The following table location forms are allowed:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

Compression in any of the supported compression formats (Unix `compress`, `gzip` or `bzip2`) is expanded automatically.

A list of input table locations may be given in an external file by using the indirection character '@'. Thus "`in=@filename`" causes the file `filename` to be read for a list of input table locations. The locations in the file should each be on a separate line.

istream = true|false (Boolean)

If set `true`, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as `VOTable`).

The same streaming flag applies to all the tables specified by `in`.

[Default: `false`]

multi = true|false (Boolean)

Determines whether all tables, or just the first one, from input table files will be used. If set `false`, then just the first table from each file named by `in` will be used. If `true`, then all tables present in those input files will be used. This only has an effect for file formats which are capable of containing more than one table, which effectively means `FITS` and `VOTable` and their variants.

[Default: `false`]

ofmt = <out-format> (String)

Specifies the format in which the output tables will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value `"(auto)"` (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename

what output format is intended, an error will result.

Not all output formats are capable of writing multiple tables; if you choose one that is not, an error will result.

[Default: (auto)]

`out = <out-file>` (*uk.ac.starlink.util.Destination*)

The location of the output file. This is usually a filename to write to. If it is equal to the special value "-" the output will be written to standard output.

[Default: -]

B.33.2 Examples

Here are some examples of using `tmulti`:

```
stilts tmulti ifmt=ascii in=t1.txt in=t2.txt in=t3.txt
          ofmt=fits out=tables.fits
```

Takes the three named ASCII format tables and writes them into a multi-extension FITS file, as three separate BINTABLE HDUs. These tables do not need to be of the same shape or otherwise similar.

```
stilts tmulti ifmt=ascii in="t1.txt t2.txt t3.txt"
          ofmt=fits out=tables.fits
```

Does exactly the same as the previous example.

```
stilts tmulti ifmt=ascii in=@inlist.lis
          ofmt=fits out=tables.fits
```

This will have the same effect as the previous two examples if a file name "inlist.lis" in the current directory contains three lines, "t1.txt", "t2.txt" and "t3.txt".

```
stilts tmulti in=extract.fits multi=true out=extract.vot
```

This takes the table extensions from a multi-extension FITS file and writes them out as a multi-TABLE VOTable document. The `multi=true` setting is required, since this means that all the tables from the input file are used as input; if it was set false, only the first table HDU from the input file would be used.

```
stilts tmulti in=extract.fits multi=true out=extract.vot
          icmd='badval -999 *MAG'
```

Does the same as the previous example, but additionally replaces with a blank value occurrences of the value "-999" in columns whose name ends with "MAG" in any of the input tables before copying them.

B.34 `tmultin`: Writes multiple processed tables to single container file

`tmultin` takes multiple input tables and writes them to a single output container file. The container file must be of some format which can contain more than one table, for instance a FITS file (which can contain multiple extensions) or a VOTable document (which can contain multiple TABLE elements). Individual filtering may be performed on the tables prior to writing them, and their formats may be specified individually. If you want to apply the same pre-processing to all the input tables, you may find the `tmulti` command more convenient.

B.34.1 Usage

The usage of `tmultin` is

```
stilts <stilts-flags> tmultin nin=<count> ifmtN=<in-format> inN=<tableN>
                                icmdN=<cmds> out=<out-file> ofmt=<out-format>
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.MultiCopyN`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

icmdN = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on input table #N as specified by parameter `inN`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of "@filename" causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmtN = <in-format> (*String*)

Specifies the format of input table #N as specified by parameter `inN`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

inN = <tableN> (*StarTable*)

The location of input table #N. This may take one of the following forms:

- A filename.
- A URL.
- The special value "-", meaning standard input. In this case the input format must be given explicitly using the `ifmtN` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a "<" character at the start, or a "|" character at the end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

nin = <count> (*Integer*)

The number of input tables for this task. For each of the input tables N there will be associated parameters `ifmtN`, `inN` and `icmdN`.

ofmt = <out-format> (*String*)

Specifies the format in which the output tables will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value `"(auto)"` (the default), then the output filename will be examined to try to guess what

sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

Not all output formats are capable of writing multiple tables; if you choose one that is not, an error will result.

[Default: (auto)]

out = <out-file> (*uk.ac.starlink.util.Destination*)

The location of the output file. This is usually a filename to write to. If it is equal to the special value "-" the output will be written to standard output.

[Default: -]

B.34.2 Examples

Here are some examples of using `tmultin`:

```
stilts tmultin nin=3 in1=t1.xml ifmt1=votable
                      in2=t2.fits ifmt2=fits
                      in3=t3.txt ifmt3=ascii
                      out=tables.fits
```

Takes three input tables in different formats, and writes them out as a single multi-extension FITS file.

```
stilts tmultin nin=3 in1=data.fits icmd1='every 10; head 100'
                      in2=data.fits icmd2='every 100; head 100'
                      in3=data.fits icmd3='every 1000; head 100'
                      out=samples.xml ofmt=votable
```

Writes three hundred-row tables as separate TABLE elements in a single output VOTable document. Each of the output tables is a sample from the same input table, but sampled differently; the first is every tenth row, the second every hundredth, and the third every thousandth.

B.35 `tpipe`: Performs pipeline processing on a table

`tpipe` performs all kinds of general purpose manipulations which take one table as input. It is extremely flexible, and can do the following things amongst others:

- calculate statistics
- display metadata
- select rows in various ways, including algebraically
- define new columns as algebraic functions of old ones
- delete or rearrange columns
- sort rows
- convert between table formats

and combine these operations. You can think of it as a supercharged table copying tool.

The basic operation of `tpipe` is that it reads an input table, performs zero or more processing steps on it, and then does something with the output. There are therefore three classes of things you need to tell it when it runs:

Input table location

Specified by the `in`, `ifmt` and `istream` parameters.

Processing steps

Either provide a string giving steps as the value of one or more `cmd` parameters, or the name of

a file containing the steps using the `script` parameter. The steps that you can perform are described in Section 6.1.

Output table destination

What happens to the output table is determined by the value of the `omode` parameter. By default, `omode=out`, in which case the table is written to a new table file in a format determined by `ofmt`. However, you can do other things with the result such as calculate the per-column statistics (`omode=stats`), view only the table and column metadata (`omode=meta`), display it directly in TOPCAT (`omode=topcat`) etc.

See Section 6 for a more detailed explanation of these ideas.

The parameters mentioned above are listed in detail in the next section.

B.35.1 Usage

The usage of `tpipe` is

```
stilts <stilts-flags> tpipe ifmt=<in-format> istream=true|false cmd=<cmds>
                             omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosc
                             out=<out-table> ofmt=<out-format>
                             [in=<table>]
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.TablePipe`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

cmd = <cmds> (*ProcessingStep[]*)

Specifies processing to be performed on the input table as specified by parameter `in`, before any other processing has taken place. The value of this parameter is one or more of the filter commands described in Section 6.1. If more than one is given, they must be separated by semicolon characters (";"). This parameter can be repeated multiple times on the same command line to build up a list of processing steps. The sequence of commands given in this way defines the processing pipeline which is performed on the table.

Commands may alternatively be supplied in an external file, by using the indirection character '@'. Thus a value of `"@filename"` causes the file `filename` to be read for a list of filter commands to execute. The commands in the file may be separated by newline characters and/or semicolons.

ifmt = <in-format> (*String*)

Specifies the format of the input table as specified by parameter `in`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in = <table> (*StarTable*)

The location of the input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmt` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the

end ("`<syscmd`" or "`syscmd|`"). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

istream = true|false (*Boolean*)

If set true, the input table specified by the `in` parameter will be read as a stream. It is necessary to give the `ifmt` parameter in this case. Depending on the required operations and processing mode, this may cause the read to fail (sometimes it is necessary to read the table more than once). It is not normally necessary to set this flag; in most cases the data will be streamed automatically if that is the best thing to do. However it can sometimes result in less resource usage when processing large files in certain formats (such as VOTable).

[Default: false]

ofmt = <out-format> (*String*)

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "`(auto)`" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "`out`".

[Default: (auto)]

omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui

(*ProcessingMode*)

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (`<mode-args>`) are required to determine the exact behaviour.

Possible values are

- out
- meta
- stats
- count
- cgi
- discard
- topcat
- samp
- plastic
- tosql
- gui

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: out]

out = <out-table> (*TableConsumer*)

The location of the output table. This is usually a filename to write to. If it is equal to the special value "`-`" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "`out`".

[Default: -]

B.35.2 Examples

Here are some examples of `tpipe` in use with explanations of what's going on. For simplicity these examples assume that you have the `stilts` script installed and are using a Unix-like shell; see Section 3 for an explanation of how to invoke the command if you just have the Java classes.

```
stilts tpipe cat.fits
```

Writes a FITS table to standard output in human-readable form. Since no mode specifier is given, `omode=out` is assumed, and output is to standard output in `text` format.

```
stilts tpipe cmd='head 5' cat.fits.gz
```

Does the same as the last example, but with one processing step: only the first five rows of the table are output. In this case, the input file is compressed using `gzip` - this is automatically detected.

```
stilts tpipe ifmt=csv xxx.csv \
    cmd='keepcols "index ra dec"' \
    omode=out ofmt=fits xxx.fits
```

Reads from a comma-separated values file, writes to a FITS file, and discards all columns in the input table apart from INDEX, RA and DEC. Note the quoting in the `cmd` argument: the outer quotes are so that the argument of the `cmd` parameter itself (`keepcols "index ra dec"`) is not split up by spaces (to protect it from the shell), and the inner quotes are to keep the `colid-list` argument of the `keepcols` command together.

```
stilts tpipe ifmt=votable \
    cmd='addcol IV_SUM "(IMAG+VMAG)"' \
    cmd='addcol IV_DIFF "(IMAG-VMAG)"' \
    cmd='delcols "IMAG VMAG"' \
    omode=out ofmt=votable \
    < tab1.vot \
    > tab2.vot
```

Replaces two columns by their sum and difference in a VOTable. Since neither the `in` nor `out` parameters have been specified, the input and output are actually byte streams on standard input and standard output of the `tpipe` command in this case. The processing steps first add a column representing the sum, then add a column representing the difference, then delete the original columns.

```
stilts tpipe cmd='addskycoords -inunit sex fk5 gal \
    RA2000 DEC2000 GAL_LONG GAL_LAT' \
    6dfgs.fits 6dfgs+gal.fits
```

Adds columns giving galactic coordinates to a table. Both input and output tables are FITS files. The galactic coordinates, stored in new columns named `GAL_LONG` and `GAL_LAT`, are calculated from FK5 J2000.0 coordinates given in the existing columns named `RA2000` and `DEC2000`. The input (FK5) coordinates are represented as sexagesimal strings (hh:mm:ss, dd:mm:ss), and the output ones are numeric degrees.

```
stilts -disk tpipe 2dfgrs_ngp.fits \
    cmd='keepcols "SEQNUM AREA ECCENT"' \
    cmd='sort -down AREA' \
    cmd='head 20'
```

Displays selected columns for the 20 rows with largest values in the `AREA` column of a FITS table. First the columns of interest are selected, then the rows are sorted into descending order by the value of the `AREA` column, then the first 20 rows of the resulting table are selected, and

the result is written to standard output. Since a sort is being performed here, it's not possible to do all the processing a row at a time, since all the AREA values must be available for comparison during the sort. Two things are done here to accommodate this fact: first the column selection is done before the sort, so that it's only a 3-column table which needs to be available for random access, reducing the temporary storage required. Secondly the `-disk` flag is supplied, which means that temporary disk files rather than memory will be used for caching table data.

```
stilts tpipe 2dfgrs_ngp.fits \
  cmd='keepcols "SEQNUM AREA ECCENT"' \
  cmd='sorthead -down 20 AREA'
```

Has exactly the same effect as the previous example. However, the algorithm used by the `sorthead` filter is in most cases faster and cheaper on memory (only 20 rows ever have to be stored in this case), so this is generally a better approach than combining the `sort` and `head` filters.

```
stilts tpipe omode=meta cmd=@commands.lis http://archive.org/data/survey.vot.z
```

Outputs column and table metadata about a table. In this case the table is a compressed VOTable at the end of a URL. Processing is performed according to the commands contained in a file named "commands.lis" in the current directory.

```
stilts tpipe in=survey.fits
  cmd='select "skyDistanceDegrees(hmsToDegrees(RA),dmsToDegrees(DEC), \
                                     hmsToDegrees(2,28,11),dmsToDegrees(-6,49,45)) \
                                     < 5./60."' \
  omode=count
```

Counts the number of rows within a given 5 arcmin cone of sky in a FITS table. The `skyDistanceDegrees` function is an expression which calculates the distance between the position specified in a row (as given by its RA and DEC columns) and a given point on the sky (here, 02:28:11,-06:49:45). Since `skyDistanceDegrees`'s arguments and return value are in decimal degrees, some conversions are required: the RA and DEC columns are sexagesimal strings which are converted using the `hmsToDegrees` and `dmsToDegrees` functions respectively. Different versions of these functions (ones which take numeric arguments) are used to convert the coordinates of the fixed point to degrees. The result is compared to a constant expression representing 5 arcminutes in degrees. Any rows of the input table for which this comparison is true are included in the output. An alternative function, `skyDistanceRadians` which works in radians, is also available. These functions and constants used here are described in detail in Section 10.5.16 and Section 10.5.13.

```
stilts tpipe ifmt=ascii survey.txt \
  cmd='select "OBJTYPE == 3 && Z > 0.15"' \
  cmd='keepcols "IMAG JMAG KMAG"' \
  omode=stats
```

Calculate statistics on the I, J and K magnitudes of selected objects from a catalogue. Only those rows with the given OBJTYPE and in the given Z range are included. The minimum, maximum, mean, standard deviation etc of the IMAG, JMAG and KMAG columns will be written to standard output.

```
stilts -classpath lib/drivers/mysql-connector-java.jar \
  -Djdbc.drivers=com.mysql.jdbc.Driver \
  tpipe in=x.fits cmd="explodeall" omode=toSQL \
  protocol=mysql host=localhost db=ASTRO1 dbtable=TABLEX \
  write=dropcreate user=mbt
```

Writes a FITS table to an SQL table, converting array-valued columns to scalar ones. To make the SQL connection work properly, the classpath is augmented to include the path of the

MySQL JDBC driver and the `jdbc.drivers` system property is set to the JDBC driver class name. The output will be written as a new table named `TABLEX` in the MySQL database named `ASTRO1` on a MySQL server on the local host. The password, if required, will be prompted for, as would any of the other required parameters if they had not been given on the command line. Any existing table in `ASTRO1` with the name `TABLEX` is overwritten. The only processing done here is by the `explodeall` command, which takes any columns which have fixed-size array values and replaces them in the output with multiple scalar columns.

```
java -classpath stilts.jar:lib/drivers/mysql-connector-java.jar
-Djdbc.drivers=com.mysql.jdbc.Driver \
uk.ac.starlink.ttools.Stilts \
ttype in=x.fits \
    cmd=explodeall \
    omode=out \
    out="jdbc:mysql://localhost/ASTRO1?user=mbt#TABLEX"
```

This does exactly the same as the previous example, but achieves it in a slightly different way. In the first place, `java` is invoked directly with the necessary flags rather than getting the `stilts` script to do it. Note that you cannot use `java`'s `-jar` flag in this case, because doing it like that would not permit access to the additional classes that contain the JDBC driver. In the second place we use `omode=out` rather than `omode=toSQL`. For this we need to supply an `out` value which encodes the information about the SQL connection and table in a special URL-like format. As you can see, this is a bit arcane, which is why the `omode=toSQL` mode can be a help.

```
stilts ttype USNOB.FITS cmd='every 1000000' omode=stats
```

Calculates statistics on a selection of the rows in a catalogue, and writes the result to the terminal. In this example, every millionth row is sampled.

B.36 `tskymatch2`: Crossmatches 2 tables on sky position

`tskymatch2` performs a crossmatch of two tables based on the proximity of sky positions. You specify the columns or expressions giving right ascension and declination in degrees for each input table, and a maximum permissible separation in arcseconds, and the resulting joined table is output.

If you omit expressions for the RA and Dec, an attempt is made to identify the columns to use using column Unified Content Descriptors (UCDs) or names. First columns bearing appropriate UCD1 or UCD1+ values (`POS_EQ_RA`, `POS_EQ_RA_MAIN`, `pos.eq.ra` or `pos.eq.ra/meta.main` and their equivalents for declination) are sought. If these cannot be found, columns named something like "RA" or "RA2000" are sought. If either is found, the column units are consulted and radian->degree conversions are performed if necessary (degrees are assumed if no unit value is given). If nothing likely can be found, then the command will fail with an error message. This search logic is intended as a convenience only; it is somewhat ad hoc and subject to change. To make sure that the correct angle values are being used, specify the `ra` and `dec` position parameters explicitly.

`tskymatch2` is simply a cut-down version, provided for convenience, of the more general matching task `tmatch2`. If you want more match options or otherwise more configurability, you can probably find it by using `tmatch2`.

B.36.1 Usage

The usage of `tskymatch2` is

```
stilts <stilts-flags> tskymatch2 ifmt1=<in-format> ifmt2=<in-format>
                                omode=out|meta|stats|count|cgi|discard|topcat|samp|plastic
                                out=<out-table> ofmt=<out-format>
```

```

ra1=<expr> decl1=<expr> ra2=<expr>
dec2=<expr> error=<value/arcsec>
tuning=<healpix-k>
join=1and2|1or2|all1|all2|1not2|2not1|1xor2
find=all|best|best1|best2
[in1=]<table1> [in2=]<table2>

```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.SkyMatch2`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

decl1 = <expr> (*String*)

Declination in degrees for the position of each row of table 1. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

decl2 = <expr> (*String*)

Declination in degrees for the position of each row of table 2. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

error = <value/arcsec> (*Double*)

The maximum separation permitted between two objects for them to count as a match. Units are arc seconds.

find = all|best|best1|best2 (*PairMode*)

Determines what happens when a row in one table can be matched by more than one row in the other table. The options are:

- **all**: All matches. Every match between the two tables is included in the result. Rows from both of the input tables may appear multiple times in the result.
- **best**: Best match, symmetric. The best pairs are selected in a way which treats the two tables symmetrically. Any input row which appears in one result pair is disqualified from appearing in any other result pair, so each row from both input tables will appear in at most one row in the result.
- **best1**: Best match for each Table 1 row. For each row in table 1, only the best match from table 2 will appear in the result. Each row from table 1 will appear a maximum of once in the result, but rows from table 2 may appear multiple times.
- **best2**: Best match for each Table 2 row. For each row in table 2, only the best match from table 1 will appear in the result. Each row from table 2 will appear a maximum of once in the result, but rows from table 1 may appear multiple times.

The differences between `best`, `best1` and `best2` are a bit subtle. In cases where it's obvious which object in each table is the best match for which object in the other, choosing between these options will not affect the result. However, in crowded fields (where the distance between objects within one or both tables is typically similar to or smaller than the specified match radius) it will make a difference. In this case one of the asymmetric options (`best1` or `best2`) is usually more appropriate than `best`, but you'll have to think about which of them suits your requirements. The performance (time and memory usage) of the match may also differ between these options, especially if one table is much bigger than the other.

[Default: `best`]

ifmt1 = <in-format> (*String*)

Specifies the format of the first input table as specified by parameter `in1`. The known formats

are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

ifmt2 = <in-format> (*String*)

Specifies the format of the second input table as specified by parameter `in2`. The known formats are listed in Section 5.2.1. This flag can be used if you know what format your table is in. If it has the special value `(auto)` (the default), then an attempt will be made to detect the format of the table automatically. This cannot always be done correctly however, in which case the program will exit with an error explaining which formats were attempted.

[Default: `(auto)`]

in1 = <table1> (*StarTable*)

The location of the first input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmt1` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

in2 = <table2> (*StarTable*)

The location of the second input table. This may take one of the following forms:

- A filename.
- A URL.
- The special value `"-"`, meaning standard input. In this case the input format must be given explicitly using the `ifmt2` parameter. Note that not all formats can be streamed in this way.
- A system command line with either a `"<"` character at the start, or a `"|"` character at the end (`"<syscmd"` or `"syscmd|"`). This executes the given pipeline and reads from its standard output. This will probably only work on unix-like systems.

In any case, compressed data in one of the supported compression formats (gzip, Unix compress or bzip2) will be decompressed transparently.

join = 1and2|1or2|all1|all2|1not2|2not1|1xor2 (*JoinType*)

Determines which rows are included in the output table. The matching algorithm determines which of the rows from the first table correspond to which rows from the second. This parameter determines what to do with that information. Perhaps the most obvious thing is to write out a table containing only rows which correspond to a row in both of the two input tables. However, you may also want to see the unmatched rows from one or both input tables, or rows present in one table but unmatched in the other, or other possibilities. The options are:

- `1and2`: An output row for each row represented in both input tables (INNER JOIN)
- `1or2`: An output row for each row represented in either or both of the input tables (FULL OUTER JOIN)
- `all1`: An output row for each matched or unmatched row in table 1 (LEFT OUTER JOIN)
- `all2`: An output row for each matched or unmatched row in table 2 (RIGHT OUTER JOIN)

- `1not2`: An output row only for rows which appear in the first table but are not matched in the second table
- `2not1`: An output row only for rows which appear in the second table but are not matched in the first table
- `1xor2`: An output row only for rows represented in one of the input tables but not the other one

[Default: `1and2`]

`ofmt = <out-format> (String)`

Specifies the format in which the output table will be written (one of the ones in Section 5.2.2 - matching is case-insensitive and you can use just the first few letters). If it has the special value "`(auto)`" (the default), then the output filename will be examined to try to guess what sort of file is required usually by looking at the extension. If it's not obvious from the filename what output format is intended, an error will result.

This parameter must only be given if `omode` has its default value of "`out`".

[Default: `(auto)`]

`omode = out|meta|stats|count|cgi|discard|topcat|samp|plastic|tosql|gui
(ProcessingMode)`

The mode in which the result table will be output. The default mode is `out`, which means that the result will be written as a new table to disk or elsewhere, as determined by the `out` and `ofmt` parameters. However, there are other possibilities, which correspond to uses to which a table can be put other than outputting it, such as displaying metadata, calculating statistics, or populating a table in an SQL database. For some values of this parameter, additional parameters (`<mode-args>`) are required to determine the exact behaviour.

Possible values are

- `out`
- `meta`
- `stats`
- `count`
- `cgi`
- `discard`
- `topcat`
- `samp`
- `plastic`
- `tosql`
- `gui`

Use the `help=omode` flag or see Section 6.4 for more information.

[Default: `out`]

`out = <out-table> (TableConsumer)`

The location of the output table. This is usually a filename to write to. If it is equal to the special value "`-`" (the default) the output table will be written to standard output.

This parameter must only be given if `omode` has its default value of "`out`".

[Default: `-`]

`ra1 = <expr> (String)`

Right ascension in degrees for the position of each row of table 1. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

`ra2 = <expr> (String)`

Right ascension in degrees for the position of each row of table 2. This may simply be a column name, or it may be an algebraic expression calculated from columns as explained in Section 10. If left blank, an attempt is made to guess from UCDs, column names and unit annotations what expression to use.

tuning = <healpix-k> (*Integer*)

Tuning parameter that controls the pixel size used when binning the rows. The legal range is from 0 (corresponding to pixel size of about 60 degrees) to 20 (about 0.2 arcsec). The value of this parameter will not affect the result but may affect the performance in terms of CPU and memory resources required. A default value will be chosen based on the size of the `error` parameter, but it may be possible to improve performance by adjusting the default value. The value used can be seen by examining the progress output. If your match is taking a long time or is failing from lack of memory it may be worth trying different values for this parameter.

B.36.2 Examples

Here are some examples of using `tskymatch2`

```
stilts tskymatch2 in1=obs_v.xml in2=obs_i.xml out=obs_iv.xml \
              ral=OBS_RA dec1=OBS_DEC ra2=OBS_RA dec2=OBS_DEC error=2
```

Takes two input catalogues (VOTables), one with observations in the V band and the other in the I band, and performs a match to find objects within 2 arcseconds of each other. The result is a new VOTable containing only rows where a match was found.

```
stilts tskymatch2 in1=obs_v.xml in2=obs_i.xml out=obs_iv.xml \
              error=2
```

This is the same as the previous example but without explicit specification of the sky position columns in either table. It will work only if those columns are identified with appropriate UCDs, for instance `pos.eq.ra:meta.main` and `pos.eq.dec:meta.main`. If no suitable UCDs are in place this invocation will fail with an error.

```
stilts tskymatch2 in1=virgo1.txt ifmt1=ascii in2=mgc.fits \
              ral='radiansToDegrees(raRad)' dec1='radiansToDegrees(deRad)' \
              ra2=MGC_ALPHA_J2000 dec2=MGC_DELTA_J2000 \
              error=10 join=2not1 omode=count
```

Object positions in the text file `virgo1.txt` are compared to those in the FITS file `mgc.fits`. The angles have been recorded in the text file in radians, so they are converted to degrees here before use. Use of the `join=2not1` parameter causes the command to identify all the objects in the first list which do not have counterparts within 10 arcsec in the second list. The number of such objects found is simply output to the terminal.

B.37 votcopy: Transforms between VOTable encodings

The VOTable standard provides for three basic encodings of the actual data within each table: TABLEDATA, BINARY and FITS. TABLEDATA is a pure-XML encoding, which is relatively easy for humans to read and write. However, it is verbose and not very efficient for transmission and processing, for which reason the more compact BINARY format has been defined. FITS format shares the advantages of BINARY, but is more likely to be used where a VOTable is providing metadata 'decoration' for an existing FITS table. In addition, the BINARY and FITS encodings may carry their data either inline (as the base64-encoded text content of a `STREAM` element) or externally (referenced by a `STREAM` element's `href` attribute).

These different formats have their different advantages and disadvantages. Since, to some extent,

programmers are humans too, much existing VOTable software deals in TABLEDATA format even though it may not be the most efficient way to proceed. Conversely, you might wish to examine the contents of a BINARY-encoded table without use of any software more specialised than a text editor. So there are times when it is desirable to convert from one of these encodings to another.

`votcopy` is a tool which translates between these encodings while making a minimum of other changes to the VOTable document. The processing may result in some changes to lexical details such as whitespace in start tags, but the element structure is not modified. Unlike `tpipe` it does not impose STIL's model of what constitutes a table on the data between reading it in and writing it out, so subtleties dependent on the exact structure of the VOTable document will not be mangled. The only important changes should be the contents of DATA elements in the document.

B.37.1 Usage

The usage of `votcopy` is

```
stilts <stilts-flags> votcopy version=1.0|1.1|1.2|1.3
                                charset=<xml-encoding> cache=true|false
                                href=true|false nomagic=true|false
                                base=<location>
                                [in=<location> [out=<location>
                                [format=]TABLEDATA|BINARY|BINARY2|FITS
```

If you don't have the `stilts` script installed, write "java -jar stilts.jar" instead of "stilts" - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.VotCopy`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

base = <location> (*String*)

Determines the name of external output files written when the `href` flag is true. Normally these are given names based on the name of the output file. But if this flag is given, the names will be based on the `<location>` string. This flag is compulsory if `href` is true and `out=-` (output is to standard out), since in this case there is no default base name to use.

cache = true|false (*Boolean*)

Determines whether the input tables are read into a cache prior to being written out. The default is selected automatically depending on the input table; so you should normally leave this flag alone.

charset = <xml-encoding> (*Charset*)

Selects the Unicode encoding used for the output XML. The available options and default are dependent on your JVM, but the default probably corresponds to UTF-8. Use `help=charset` for a full listing.

format = TABLEDATA|BINARY|BINARY2|FITS (*uk.ac.starlink.votable.DataFormat*)

Determines the encoding format of the table data in the output document. If `null` is selected, then the tables will be data-less (will contain no DATA element), leaving only the document structure. Data-less tables are legal VOTable elements.

The BINARY2 format is only available for `version=1.3`

[Default: TABLEDATA]

href = true|false (*Boolean*)

In the case of BINARY or FITS encoding, this determines whether the STREAM elements output will contain their data inline or externally. If set false, the output document will be self-contained, with STREAM data inline as base64-encoded characters. If true, then for each TABLE in the document the binary data will be written to a separate file and referenced by an

href attribute on the corresponding STREAM element. The name of these files is usually determined by the name of the main output file; but see also the `base` flag.

in = <location> (*String*)

Location of the input VOTable. May be a URL, filename, or "-" to indicate standard input. The input table may be compressed using one of the known compression formats (Unix compress, gzip or bzip2).

[Default: -]

nomagic = true|false (*Boolean*)

Eliminate the null attributes of VALUES elements where they are no longer required. In VOTable versions <=1.2, the only way to specify null values for integer-type scalar columns was to use the null attribute of the VALUES element to indicate an in-band magic value representing null. From VOTable v1.3, null values can be represented using empty <TD> elements or flagged specially in BINARY2 streams. In these cases, it is recommended (though not required) not to use the VALUES/null mechanism.

If this parameter is set true, then any VALUES/null attributes will be removed in VOTable 1.3 BINARY2 or TABLEDATA output. If this results in an empty VALUES element, it too will be removed.

This parameter is ignored if the output VOTable version is lower than 1.3 or if `format=BINARY/FITS`.

[Default: true]

out = <location> (*String*)

Location of the output VOTable. May be a filename or "-" to indicate standard output.

[Default: -]

version = 1.0|1.1|1.2|1.3 (*uk.ac.starlink.votable.VOTableVersion*)

Determines the version of the VOTable standard to which the output will conform. If null (the default), the output table will have the same version as the input table.

B.37.2 Examples

Normal use of `votcopy` is pretty straightforward. We give here a couple of examples of its input and output.

Here is an example VOTable document, `cat.vot`:

```
<VOTABLE>
<RESOURCE>

<TABLE name="Authors">
<FIELD name="AuthorName" datatype="char" arraysize="*" />
<DATA>
<TABLEDATA>
<TR><TD>Charles Messier</TD></TR>
<TR><TD>Mark Taylor</TD></TR>
</TABLEDATA>
</DATA>
</TABLE>

<RESOURCE>
<COOSYS equinox="J2000.0" epoch="J2000.0" system="eq_FK4" />
<TABLE name="Messier Objects">
<FIELD name="Identifier" datatype="char" arraysize="10" />
<FIELD name="RA" datatype="double" units="degrees" />
<FIELD name="Dec" datatype="double" units="degrees" />
<DATA>
<TABLEDATA>
<TR> <TD>M51</TD> <TD>202.43</TD> <TD>47.22</TD> </TR>
```

```

<TR> <TD>M97</TD> <TD>168.63</TD> <TD>55.03</TD> </TR>
</TABLEDATA>
</DATA>
</TABLE>
</RESOURCE>

</RESOURCE>
</VOTABLE>

```

Note that it contains more structure than just a flat table: there are two `TABLE` elements, the `RESOURCE` element of the second one being nested in the `RESOURCE` of the first. Processing this document using a generic table tool such as `tpipe` or `tcopy` would lose this structure.

To convert the data encoding to `BINARY` format, we simply execute

```
stilts votcopy format=binary cat.vot
```

and the output is

```

<?xml version="1.0"?>
<VOTABLE>
<RESOURCE>

<TABLE name="Authors">
<FIELD name="AuthorName" datatype="char" arraysize="*" />
<DATA>
<BINARY>
<STREAM encoding='base64'>
AAAD0NoYXJsZXMGTWVzc2llcgAAAAtNYXJrIFRheWxvcg==
</STREAM>
</BINARY>
</DATA>
</TABLE>

<RESOURCE>
<COOSYS equinox="J2000.0" epoch="J2000.0" system="eq_FK4" />
<TABLE name="Messier Objects">
<FIELD name="Identifier" datatype="char" arraysize="10" />
<FIELD name="RA" datatype="double" units="degrees" />
<FIELD name="Dec" datatype="double" units="degrees" />
<DATA>
<BINARY>
<STREAM encoding='base64'>
TTUxAAAAAAAAAEBpTcKPXCj2QEecKPXCj1xNOTcAAAAAAAAAQGUUKPXCj1xAS4PX
Cj1wpA==
</STREAM>
</BINARY>
</DATA>
</TABLE>
</RESOURCE>

</RESOURCE>
</VOTABLE>

```

Note that both tables in the document have been translated to `BINARY` format. The basic structure of the document is unchanged: the only differences are within the `DATA` elements. If we ran

```
stilts votcopy format=tabledata
```

on either this output or the original input then the output would be identical (apart perhaps from whitespace) to the input table, since the data are originally in `TABLEDATA` format.

To generate a `VOTable` document with the data in external files, the `href` parameter is used. We will output in `FITS` format this time. Executing:

```
stilts votcopy format=fits href=true cat.vot fcat.vot
```

writes the following to the file `fcat.vot`:

```
...
```

```

<DATA>
<FITS>
<STREAM href="fcats-1.fits"/>
</FITS>
</DATA>
...
<DATA>
<FITS>
<STREAM href="fcats-2.fits"/>
</FITS>
</DATA>
...

```

(the unchanged parts of the document have been skipped here for brevity). The actual data are written in two additional files in the same directory as the output file, `fcats-1.fits` and `fcats-2.fits`. These filenames are based on the main output filename, but can be altered using the `base` flag if required. Note this has also given you FITS binary table versions of all the tables in the input VOTable document, which can be operated on by normal FITS-aware software quite separately from the VOTable if required.

B.38 `votlint`: Validates VOTable documents

The VOTable standard, while not hugely complicated, has a number of subtleties and it's not difficult to produce VOTable documents which violate it in various ways. In fact it's probably true to say that most VOTable documents out there are not strictly legal. In some cases the errors are small and a parser is likely to process the document without noticing the trouble. In other cases, the errors are so serious that it's hard for any software to make sense of it. In many cases in between, different software will react in different ways, in the worst case appearing to parse a VOTable but in fact understanding the wrong data.

`votlint` is a program which can check a VOTable document and spot places where it does not conform to the VOTable standard, or places which look like they may not mean what the author intended. It is meant for use in two main scenarios:

1. For authors of VOTables and VOTable-producing software, to check that the documents they produce are legal and problem-free.
2. For users of VOTables (including authors of VOTable-processing software) who are having problems with one and want to know whether it is the data or the software at fault.

Validating a VOTable document against the VOTable schema or DTD of course goes a long way towards checking a VOTable document for errors (though it's clear that many VOTable authors don't even go this far), but it by no means does the whole job, simply because the schema/DTD specification languages don't have the facilities to understand the data structure of a VOTable document. For instance the VOTable schema will allow any plain text content in a `TD` element, but whether this makes sense in a VOTable depends on the `datatype` attribute of the corresponding `FIELD` element. There are many other examples. `votlint` tackles this by parsing the VOTable document in a way which understands its structure and assessing the content as critically as it can. For any incorrect or questionable content it finds, it will output a short message describing the problem and giving its location in the document. What you do with this information is then up to you.

Using `votlint` is very straightforward. The `votable` argument gives the location (filename or URL) of a VOTable document. Otherwise, the document will be read from standard input. Error and warning messages will be written on standard error. Each message is prefixed with the location at which the error was found (if possible the line and column are shown, though this is dependent on your JVM's default XML parser). The processing is SAX-based, so arbitrarily long tables can be processed without heavy memory use.

`votlint` can't guarantee to pick up every possible error in a VOTable document, but it ought to pick

up many of the most serious errors that are commonly made in authoring VOTables.

Note: `votlint`'s handling of XML namespaces seems to be somewhat dependent on the XML parser in use. As far as I can see, Crimson (the default in many JREs) works for any namespace arrangements, but Xerces seems to have problems when validating documents which use namespace prefixes. Not sure about other parsers. This probably won't cause you trouble, but if it does you may need to set `validate=false` to work around it. Contact this author if this seems to be a serious issue for you.

B.38.1 Usage

The usage of `votlint` is

```
stilts <stilts-flags> votlint validate=true|false version=1.0|1.1|1.2|1.3
                                out=<location>
                                [votable=]<location>
```

If you don't have the `stilts` script installed, write `"java -jar stilts.jar"` instead of `"stilts"` - see Section 3. The available `<stilts-flags>` are listed in Section 2.1. For programmatic invocation, the Task class for this command is `uk.ac.starlink.ttools.task.VotLint`.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

out = <location> (*uk.ac.starlink.util.Destination*)

Destination file for output messages. May be a filename or "-" to indicate standard output.

[Default: -]

validate = true|false (*Boolean*)

Whether to validate the input document against the VOTable DTD. If true (the default), then as well as `votlint`'s own checks, it is validated against an appropriate version of the VOTable DTD which picks up such things as the presence of unknown elements and attributes, elements in the wrong place, and so on. Sometimes however, particularly when XML namespaces are involved, the validator can get confused and may produce a lot of spurious errors. Setting this flag false prevents this validation step so that only `votlint`'s own checks are performed. In this case many violations of the VOTable standard concerning document structure will go unnoticed.

[Default: true]

version = 1.0|1.1|1.2|1.3 (*uk.ac.starlink.votable.VOTableVersion*)

Selects the version of the VOTable standard which the input table is supposed to exemplify. The version may also be specified within the document using the "version" attribute of the document's VOTABLE element; if it is and it conflicts with the value specified by this flag, a warning is issued.

If no value is provided for this parameter (the default), the version will be determined from the VOTable itself.

votable = <location> (*InputStream*)

Location of the VOTable to be checked. This may be a filename, URL or "-" (the default), to indicate standard input. The input may be compressed using one of the known compression formats (Unix compress, gzip or bzip2).

[Default: -]

B.38.2 Items Checked

Votlint checks that the XML input is well-formed, and, unless the `valid=false` parameter is supplied, that it validates against the 1.0 DTD or 1.1, 1.2 or 1.3 schema as appropriate. Some of the validity checks are also done by `votlint` internally, so that some validity-type errors may give rise to more than one warning. In general, the program errs on the side of verbosity.

In addition to these checks, the following checks are carried out, and lead to ERROR reports if violations are found:

- TD contents incompatible `datatype/arraysize` attributes declared in `FIELD`
- `BINARY/BINARY2` data streams which don't match metadata declared in `FIELD`
- `PARAM` values incompatible with declared `datatype/arraysize`
- Meaningless `arraysize` declarations
- Array-valued TD elements with the wrong number of elements
- Array-valued `PARAM` values with the wrong number of elements
- `nrows` attribute on `TABLE` element different from the number of rows actually in the table
- `VOTABLE` version attribute is unknown
- `ref` attributes without matching `ID` elements elsewhere in the document
- Same `ID` attribute value on multiple elements.

Additionally, the following conditions, which are not actually forbidden by the VOTable standard, will generate WARNING reports. Some of these may result from harmless constructions, but it is wise at least to take a look at the input which caused them:

- Wrong number of TD elements in row of `TABLEDATA` table
- Mismatch between VOTable and FITS column metadata for FITS data encoding
- `TABLE` with no `FIELD` elements
- Use of deprecated attributes
- `FIELD` or `PARAM` elements with `datatype` of either `char` or `unicodeChar` and undeclared `arraysize` - this is a common error which can result in ignoring all but the first character in TD elements from a column
- `ref` attributes which reference other elements by `ID` where the reference makes no, or questionable sense (e.g. `FIELDref` references `FIELD` in a different table)
- Multiple sibling elements (such as `FIELDS`) with the same `name` attributes

B.38.3 Examples

Here is a brief example of running `votlint` against a (very short) imperfect VOTable document. If the document looks like this:

```
<VOTABLE version="1.1">
  <RESOURCE>
    <TABLE nrows="2">
      <FIELD name="Identifier" datatype="char"/>
      <FIELD name="RA" datatype="double"/>
      <FIELD name="Dec" datatype="double"/>
      <DESCRIPTION>A very small table</DESCRIPTION>
      <DATA>
        <TABLEDATA>
          <TR>
            <TD>Fomalhaut</TD>
            <TD>344.48</TD>
            <TD>-29.618</TD>
            <TD>HD 216956</TD>
          </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

then the output of a `votlint` run looks like this:

```

INFO (1.4): No arraysize for character, FIELD implies single character
ERROR (1.7): Element "TABLE" does not allow "DESCRIPTION" here.
WARNING (1.11): Characters after first in char scalar ignored (missing arraysize?)
WARNING (1.15): Wrong number of TDs in row (expecting 3 found 4)
ERROR (1.18): Row count (1) not equal to nrows attribute (2)

```

(Note that the details of the reports will vary according to the XML parser/validator that forms part of your Java installation.)

Note also that the warning at line 11 has resulted from the same error as the one at line 4 - because the `FIELD` element has no `arraysize` attribute, `arraysize="1"` (single character) is assumed, while the author almost certainly intended `arraysize="*"` (unknown length string).

By examining these warnings you can see what needs to be done to fix this table up. Here is what it should look like:

```

<VOTABLE version="1.1">
  <RESOURCE>
    <TABLE nrows="1">
      <DESCRIPTION>A very small table</DESCRIPTION>
      <FIELD name="Identifier" datatype="char"
              arraysize="*" />
      <FIELD name="RA" datatype="double" />
      <FIELD name="Dec" datatype="double" />
    <DATA>
      <TABLEDATA>
        <TR>
          <TD>Fomalhaut</TD>
          <TD>344.48</TD>
          <TD>-29.618</TD>
        </TR>
      </TABLEDATA>
    </DATA>
  </TABLE>
</RESOURCE>
</VOTABLE>

```

<!-- change row count -->
 <!-- move DESCRIPTION -->
 <!-- add arraysize -->
 <!-- remove extra TD -->

When fed this version, `votlint` gives no warnings.

C Release Notes

This is STILTS, Starlink Tables Infrastructure Library Tool Set. It is a collection of non-GUI utilities for general purpose table manipulation.

Author

Mark Taylor (Bristol University)

Email

m.b.taylor@bristol.ac.uk

WWW

<http://www.starlink.ac.uk/stilts/>

User comments, suggestions, requests and bug reports to the above address are welcomed.

C.1 Acknowledgements

The initial development of STILTS was done under the UK's Starlink project (1980-2005, R.I.P.). Since then it has been supported by grant PP/D002486/1 from the UK's Particle Physics and Astronomy Research Council, the VOTech project (from EU FP6), the AstroGrid project (from PPARC/STFC), the AIDA project (from EU FP7), grants ST/H008470/1, ST/I00176X/1, ST/J001414/1 and ST/L002388/1 from the UK's Science and Technology Facilities Council (STFC), the GAVO project (BMBF Bewilligungsnummer 05A08VHA), the European Space Agency, and the EU FP7-2013-Space project GENIUS. All of this support is gratefully acknowledged.

Apart from the excellent Java 2 Standard Edition itself, the following external libraries provide important parts of STILTS's functionality:

- JEL (GNU) for algebraic expression evaluation
- PixTools (Fermilab EAG) for HEALPix-based celestial sphere row matching
- iText (1T3XT BVBA) for PDF output
- EPSGraphics2D (Jibble) for encapsulated postscript output
- MOC (CDS) for Multi-Order Coverage map manipulation
- ADQL (CDS) for ADQL parsing in TAP query preparation
- nom.tam.fits (NASA) for parts of FITS I/O
- Skyview in a Jar (NASA) for sky axis drawing
- JLaTeXMath (Scilab) for LaTeX typesetting in plots
- GifEncoder (Acme) for GIF output
- HTM (Sloan Digital Sky Survey) for HTM-based celestial sphere row matching (now deprecated within STILTS)

Thanks in particular to Nickolai Kouropatkine and Chris Stoughton of Fermilab for writing the PixTools specially for use in STIL.

Many people have contributed ideas and advice to the development of STILTS and its related products. I can't list all of them here, but my thanks are especially due to the following:

- Malcolm Currie (Starlink, RAL)
- Clive Davenhall (Royal Observatory Edinburgh)
- Peter Draper (Starlink, Durham)
- David Giarretta (Starlink, RAL)
- Clive Page (AstroGrid, Leicester)

If you use this software in published work, the following citation would be appreciated:

2006ASPC..351..666T: M. B. Taylor, "STILTS - A Package for Command-Line Processing

of Tabular Data", in *Astronomical Data Analysis Software and Systems XV*, eds. C. Gabriel et al., ASP Conf. Ser. 351, p. 666 (2006)

C.2 Version History

Releases to date have been as follows:

Version 0.1b (29 April 2005)

First public release

Version 0.2b (30 June 2005)

- Added Times func class for MJD-ISO8601 time conversions.
- Fixed bug when doing NULL_ test expressions on first column in table.

Version 1.0b (30 September 2005)

This is the first non-experimental release of STILTS, and it incorporates major changes and backward incompatibilities since version 0.2b.

Parameter system

The parameter system has undergone a complete rewrite; there is now only a single command "stilts", invoked using the `stilts` script or the `stilts.jar` jar file, and the various tasks are named as subsequent arguments on the command line. Command arguments are supplied after that. The new invocation syntax is described in detail elsewhere in this document. As well as invocation features such as improved on-line help, optional prompting, parameter defaulting, and more uniform access to common features, this will make it more straightforward to wrap these tasks for use in non-command-line environments, such as behind a SOAP or CORBA interface, or in a CEA-like execution environment.

Crossmatching

A new command `tmatch2` has been introduced. This provides flexible and efficient crossmatching between two input tables. Future releases will provide commands for intra-table and multi-table matching.

Concatentation

A new command `tcat` has been introduced, which allows two tables to be glued together top-to-bottom. This is currently working but very rudimentary - improvements will be forthcoming in future releases.

Calculator

A new utility command `calc` has been introduced, which performs one-line expression evaluations from the command line.

Pipeline filters

The following new filter commands for use in `tpipe` and other commands have been introduced:

- `addskycoords`: calculates new celestial coordinate pair from existing ones (FK4, FK5, ecliptic, galactic, supergalactic)
- `replacecol`: replaces column data, using existing metadata
- `badval`: replaces given 'magic' value with null
- `replaceval`: replaces given 'magic' value with any specified value
- `tablename`: edits table name
- `explodecols` and `explodecols` commands replace `explode`

The new `stream` parameter of `tpipe` now allows you to write filter commands in an external file, to facilitate more manageable command lines.

Wildarding for column specification is now allowed for some filter commands.

Algebraic functions

- New functions for converting time values between different coordinate systems (Modified Julian Date, ISO-8601, Julian Epoch and Besselian Epoch).
- New RANDOM special function.

Documentation

SUN/256 has undergone many changes. Much of the tool documentation is now automatically generated from the code itself, which goes a long way to ensuring that the documentation is correct with respect to the current state of the code.

Version 1.0-1b (7 October 2005)

Fixed jar file manifest bug which prevented working on Java 1.5

Version 1.1 (10 May 2006)

A number of new features and capabilities have been introduced:

tcube Command

The new `tcube` (Appendix B.27) command calculates N-dimensional histograms (density maps) from N columns of an input table and writes the result to a FITS file.

Processing Filters

The following new filters have been added:

- `stats` filter provides the same information as the old `stats` output mode, but allows much more flexible use of the results. It can also calculate many new quantities, including quantiles, skew and kurtosis.
- `meta` filter provides the same information as the old `meta` output mode, but allows much more flexible use of the results.
- `assert` filter provides in-pipeline logical assertions.
- `uniq` filter collapses multiple adjacent identical or similar rows.
- `sorthead` filter provides a (usually) more efficient method of doing what you could previously do by combining `sort` and `head` filters.
- `colmeta` filter adds/modifies metadata for selected columns.
- `check` filter checks table in stream - for debugging purposes only.

Additionally usage of the `sort` filter has been changed so that it can now do everything that `sortexpr` used to be able to do; `sortexpr` is now withdrawn.

Output Modes

The following new output modes have been introduced:

- `plastic` mode broadcasts the table to one or all registered PLASTIC listeners.
- `cgi` mode writes the table to standard output in a form suitable for output from a CGI script.
- `discard` mode throws away the table.

and usage of the following has been modified:

- `topcat` mode now attempts to use PLASTIC (amongst other methods) to contact TOPCAT.
- `stats` and `meta` modes are mildly deprecated in favour of the corresponding new filters (see above).

Other new features

- New IPAC table format input handler added.
- New `csv-noheader` format variant output handler added.
- `roundDecimal` and `formatDecimal` functions introduced for more control over visual

appearance of numeric values.

- Experimental facilities for automatically generating a CEA application description file.

Bug fixes and minor improvements

- Now copes with 'K'-format FITS binary table columns (64-bit integers).
- Improved, though still imperfect, retention of table-wide metadata in VOTables.
- Distinctions between null and false values in boolean columns are handled more carefully for FITS and VOTable files.
- Efficiency improvement when writing FITS-plus format (now only requires a maximum of two passes rather than three of the input rows).
- Added the `mark.workaround` system property which can optionally work around a bug in some input streams ("Resetting to invalid mark" errors).
- Fixed a bug in Cartesian matching which failed to match if the required error in any dimension was zero.
- Fixed erroneous reports about unknown `ucd` and `utype` attributes of TABLE element in `votlint`.
- When joining tables, column name comparison to determine whether deduplication is required is now case-insensitive.
- Error message improved when no automatic format detection is attempted for streamed tables.
- Setting `istream=true` is now less likely to cause a "Can't re-read stream" error.

Version 1.2 (7 July 2006)

Column-oriented Storage

New features for permitting column-oriented storage (`colfits` format, new `startable.storage` policy "sideways") have been introduced. These can provide considerable efficiency improvements for certain tasks when working with very large (and especially wide) tables.

New VO commands

Added two new commands for querying Virtual Observatory services:

- `multicone` - Makes multiple cone search queries to the same service
- `regquery` - Queries the VO registry

These tasks are experimental and may be modified or renamed in future releases.

Other items

- `transpose` filter added.
- Added flux conversion functions (Jansky<->magnitude).
- ISO-8601 strings now permit times of 24:00:00 as they should.

Version 1.2-1 (3 August 2006)

- Tab-Separated Table (TST) format now supported for reading and writing.
- New `setparam` and `clearparams` filters.
- Added ICRS coordinate system for `addskycoords`.
- TUCDnn header cards now used in FITS files to transmit UCDs (non-standard mechanism).
- Efficiency improvements for column-oriented access.

Version 1.3 (5 October 2006)

Table Concatenation

The old `tcat` command has been replaced by more capable `tcat` and `tcatn` commands.

Between them these provide concatenation of an unlimited number of homogeneous or heterogeneous input tables. Additional columns may be added to indicate which of the input tables given output rows originated from.

Parameter value indirection

Certain parameters (in `in`, `tc`, `cmd` and `friends`) may now be specified in the form `"@filename"`. This indicates that the value for the parameter is to be obtained by reading it from the named file. This is useful if a very long value is required for the parameter in question. The `script` parameter of `tpipe` has therefore been withdrawn, since it did just the same thing.

MySpace access

Direct access to the MySpace virtual file system is now provided by use of `ivo:-` or `myspace:-` type URLs.

Conversion functions

- Time conversion functions between MJD and Decimal Year have been added (Section 10.5.6).
- `toHex` and `fromHex` numeric conversion functions have been added (Section 10.5.3).

Documentation improvements

- The HTML version of SUN/256 now uses CSS to provide better highlighting of commands etc.
- The Output Modes and Processing Filter sections are now split into subsections to make the table of contents clearer.
- The Command Reference section now has only one level of subsection listed in the table of contents to make it clearer.

Other new features and improvements

- Added `-J` flag to `stilts` script for passing flags directly to Java.
- Added new `out` parameter to `votlint`.
- Added `-ifndim` and `-ifshape` flags to `explodeall` filter.
- The `exact match` mode in `tmatch2` now copes with array-valued columns.
- Added `force` parameter to `multicone` task as a workaround for some broken services.
- Added Sample (as opposed to Population) Standard Deviation/Variance calculation options to the `stats` filter.
- Improved CEA description file output - now contains details of all tasks rather than just a few, as well as various improvements in documentation etc.

Bug fixes

- Fixed erroneous complaints from `votlint` about `utype` attribute on RESOURCE elements.
- Fixed a couple of minor crossmatching bugs (which wouldn't have affected results).

Version 1.3-1 (Starlink Hokulei release)

- New command `tjoin` introduced.
- Output to MySpace can now be streamed, if running under J2SE1.5 or later.
- Slight changes to parameters for `votlint` and `votcopy`.
- Fixed bug in handling of single quotes in FITS file metadata.
- Added `-bench` flag to `stilts` command.
- Various scalability improvements for use with very large (Tb?) files.
- Improved efficiency for `text` and `ascii` output formats (now one-pass not two-pass).
- Improved CEA app-description file, including especially option lists for things like input

and output formats.

- Added README.cea file to distribution.
- Fixed problem which could mis-report VOTable out of memory errors as Broken Pipe.
- Added Vega \leftrightarrow AB magnitude conversion constants to Fluxes functions.
- Added `duptag` parameters to `tmatch2` task for customised renaming of columns with duplicated names.
- Added hyperbolic trig functions to Maths class (`sinh`, `cosh`, `tanh` and inverses).
- Added cosmology distance calculations in class Distances.
- Added `funcs` task, a browser for expression language function documentation.
- Added `-checkversion` to list of `stilts` flags.

Version 1.3-2 (6 July 2007)

- Added optional `table` parameter to `calc` command (for access to table parameters).
- Can use table parameter names in expressions using `param$` notation (Section 10.2).
- Can reference columns/parameters by UCD by using `ucd$` notation in expressions (Section 10.1) and as column identifiers (Section 6.2).
- Improved deduplication of column names when joining tables.
- Fix error in output of FITS table `TNULL` *n* header cards - write them as numeric not string values.
- Improve error message for broken CSV files.
- Modified JDBC handling so that MySQL and PostgreSQL do not run out of heap memory when streaming large datasets for input. Think I've done the same for SQL Server, but this is not tested.
- Improve error reporting in the presence of a deficient JVM (such as GNU `gcj`).
- Add locale-specific `formatDecimalLocal` functions in class Formats.
- Add `fluxToLuminosity` and `luminosityToFlux` functions in class Fluxes.
- Fix bug which was causing `NullPointerException`s in the `transpose` filter.

Version 1.3-3 (4 Sep 2007)

- Experimental, and currently undocumented, `sqlcone` task introduced, along with some classes in package `uk.ac.starlink.ttools.cone` designed for library use by AstroGrid DSA code.
- CEA description of `tmatch2` `params` parameter now has `minoccurs=0`, since that can be true for exact matches.

Version 1.3-4 (10 Sep 2007)

- Fixed VotCopy bug.

Version 1.3-5 (30 Oct 2007)

- Added `-stdout` and `-stderr` flags to `stilts` command.
- Some bugs fixed in generation of CEA `app-description.xml` file.
- Documentation provided for `sqlcone` command.
- Fixed error in `fluxToLuminosity` function.

Version 1.4 (6 December 2007)

Table joins

This version provides more cross matching functionality. Added to the existing `tmatch2` command are new tasks:

- `tskymatch2`: stripped down version of `tmatch2` for ease of use when matching with sky coordinates.
- `tmatch1`: internal matcher, finds groups of objects within a table.
- `tmatchn`: finds group or multiple-pair matches between multiple (>2) tables.

Two tasks have been renamed for improved clarity and consistency:

- `multicone` is now named `coneskymatch`
- `sqlcone` is now named `sqlskymatch`

There has also been some enhancement and rationalisation of parameters for all table join tools (`tmatch*` as well as `tjoin`, `coneskymatch` and `sqlskymatch`):

- All table join commands now use similar `fixcols` and `suffix*` parameters to control renaming of duplicated columns in output tables (note this replaces the old `duptag*` parameters in `tmatch2`).
- Crossmatching tasks have a new `progress` parameter which allows you to configure whether progress is reported to the console.
- The `copycols` parameter of `coneskymatch` and `sqlskymatch` now defaults to `"*"` (include all columns from input table in the output).

Section 7 of the manual has been somewhat rearranged and improved.

Other enhancements

- FITS reader now imports table HDU header cards as table parameters.
- CeaWriter can now output CEA service definition XML config file as well as app-description file (experimental - may be withdrawn).

Bug fixes

- Embedded spaces in output ASCII format table column names are now substituted with underscores.
- Fix a bug which caused an infinite number of dots to be printed when attempting a crossmatch with an empty input table.
- Corrected `votlint` handling of TABLEDATA-type multi-dimensional `char/unicodeChar` arrays. These are now split up into strings by counting characters rather than using whitespace delimiters. I *think* it's doing the right thing now.

Version 1.4-1 (28 January 2008)

New RDBMS-related features

- New command `sqlclient`, which is a general JDBC-based SQL command-line client.
- New command `sqlupdate`, which allows updates to existing rows in SQL tables.
- Some changes to `tosql` output mode:
 - choice of options for how to write to the database output table, controlled by new associated parameter `write` (can be `create`, `dropcreate` or `append`)
 - associated parameter `newtable` renamed `dbtable`
 - associated parameter `database` renamed `db` for consistency with other commands

Local and service-based matching command enhancements

- New parameter `scorecol` added to `tmatch2`, `coneskymatch` and `sqlskymatch` commands, which controls adding a new column to match output tables containing a goodness-of-match value.
- New parameter `parallel` added to `coneskymatch` task which allows multiple cone searches to be carried out in parallel.
- New parameter `erract` added to `coneskymatch` which controls response to isolated failures in individual cone search queries.

General improvements

- Improved error reporting (reasons for errors are now reported even without the `-debug` flag).
- Add new help option `help='*'` which prints help for all parameters of a task at once.
- Added (mostly undocumented) `+verbose` flag for reducing verbosity level.
- Minor improvements to CEA app-description.
- Downgraded from WARNING to INFO log messages about the (extremely common) VOTable syntax error of omitting a FIELD/PARAM element's `datatype` attribute.

Version 1.4-2 (26 March 2008)

Minor enhancements:

- Add `progress` parameter to `tmatchn`.
- Add `emptyok` parameter to `conesky` match.

Bugfixes:

- Fixed pair matching performance bug (slower if tables were not given in the right order) introduced at v1.4.
- Fixed null handling error in `calc` task.
- Fixed error in `stats` filter cardinality value calculation.
- Fixed minor bugs in suffix addition for matching commands `fixcols`.
- Removed unformatted XML output in `stats` filter usage message.
- Try to avoid exponential format in cone search URLs (some endpoints seem to require fixed point format).
- Minor CEA fixes.

Version 2.0b (23 October 2008)

This version contains two new major items, plotting and server mode. Both work, but are missing desirable features and have not had extensive testing in the field, so should be considered experimental at this stage.

Plotting

Two table plotting commands are now provided:

- `plot2d`: Old-style 2D Scatter Plot
- `plot3d`: Old-style 3D Scatter Plot
- `plothist`: Old-style Histogram

See also the new Plotting (Section 9) section in the manual.

Server/Servlet Mode

A new command `server` is provided which allows STILTS commands to be executed via HTTP. One purpose of this is to facilitate server-side use of the plotting commands co-located with data to generate on-the-fly graphical summaries of server-held datasets.

Smaller enhancements and bugfixes

- Efficiency improvements (~25%? in both CPU time and memory usage) for HEALPix-based sky crossmatching (thanks to Nikolay Kouropatkine at Fermilab for a new version of the PixTools library).
- New class Arrays added to algebraic functions.
- New Appendix Commands by Category (Appendix A) added to manual.
- Add `minReal` and `maxReal` functions (max/min ignoring blank values) in class Arithmetic.
- Sexagesimal field identification for ASCII input files is less stringent (now permits

minutes or seconds equal to 60).

- Minor CEA fixes.
- HEALPix bug fix (PixTools bug fix update).
- Fix bug in use of `tcats`'s `loccol` parameter.

Version 2.0-1 (23 December 2008)

- Can reference columns/parameters by Utype by using `utype$` notation in expressions (Section 10.1) and as column identifiers (Section 6.2).
- Non-alphanumeric column names may now be used for algebraic expressions in the special case that the expression is just the value of a single column.
- `regquery` command has changed in implementation, data access, and output format. It now queries VOResource1.0 registries rather than the very out of date registry protocol which was used in earlier versions.

Version 2.0-2 (9 January 2009)

- Added new `samp` output mode which passes the generated table to clients using the SAMP protocol.
- Updated the `topcat` output mode to use SAMP as one way of communicating with a running TOPCAT.
- `-version` flag now reports `starjava` subversion revision as well as other items.

Version 2.0-3 (27 March 2009)

- Fits BINTABLE TZERO/TSCAL value reading improvements:
 - Columns with integer TZERO values now read as integers rather than floating point values where possible. This includes unsigned longs ('K'), which were previously represented as doubles with lost precision. Unsigned longs which are too large however ($>2^{63}$) are read as nulls.
 - Byte-valued columns can now be written out by `fits-basic` output handler as signed byte values (TFORM=B,TZERO=-128) rather than signed shorts (TFORM=I).
 - More comprehensive testing.
 - Fixed bug in calculating value scaled double ('D') values.
 - Fixed bug in typing value for scaled float ('E') arrays.
 - Fixed bug which caused registry queries (`regquery`) to fail for Java 1.6.
- Fix minor bugs in detail of `votlint`'s validation tests (VOTABLE element content model, INFO and PARAM and FIELD required attributes).
- Report application name and version in User-Agent header of outgoing HTTP requests.
- The fixed length Substring Array Convention for string arrays (TFORMnn=rAw) is now understood for FITS binary tables.
- Minor SAMP bugs fixed (JSAMP upgraded to 0.3-1).

Version 2.0-4 (17 July 2009)

- Work around J2SE mark/reset bug when loading table direct from URL.
- Produce null rather than nonsense results from sky coordinate conversions with unphysical latitudes (`addskycoords` filter).
- Produce null rather than questionable results from sexagesimal conversions with mins/secs out of range.
- Fix two bugs in `votcopy`: XML processing instructions garbled on output, and pathnames in `base` parameters inappropriately flattened in `hrefs` attribute values.

Version 2.0-5 (2 Oct 2009)

- VOTable 1.2 supported.
- `votlint` can now validate VOTable documents following the (provisional, 2009-09-29 PR) VOTable 1.2 standard.
- Namespacing of VOTable documents made more intelligent, and configurable using the `votable.namespacing` system property.
- `votlint` now checks that the correct XML namespaces are in use.
- Be more careful in XML, including VOTable, output; fix VOTable output encoding to be UTF-8, and ensure no illegal XML characters are written.
- HTML table output is now HTML 4.01 by default (includes THEAD and TBODY tags).
- `parse* string->numeric` conversion functions now cope with leading or trailing whitespace.
- Work around illegally truncated type declarations in IPAC tables.
- Fix a bug which caused the first table in a multi-table file (FITS or VOTable) to be used in streaming mode, even if a subsequent one was requested.
- Bug fixed in crossmatching output: entries which should have been null were sometimes written as non-null (typically large negative numbers) in FITS and in non-TABLEDATA VOTable output. This affected cells in otherwise non-nullable columns where the entire row was absent. The previous behaviour is not likely to have been mistaken for genuine results.

Version 2.1 (6 November 2009)

- `coneskyrmatch` can now match using SIA and SSA services as alternatives to Cone Search ones (see its new `servicetype` parameter).
- Fixed an obscure bug which could under rare circumstances cause truncation of strings with leading/trailing whitespace read from text-format files.
- A new `startable.storage` policy "adaptive" is now the default. This should mean running out of memory less often. The old behaviour can be restored by giving the new `-memory` command line flag.

Note that the STIL API used by this release has changed in some backwardly incompatible ways, and may change further. If you're using STILTS as a library rather than an application you might want to wait for a later release when the API has settled down.

Version 2.1-1 (21 December 2009)

- Plotting commands can now output to PDF as well as existing graphics formats.
- New filter `fixcolnames`.
- Fixed internationalisation bug which could cause `coneskyrmatch` to fail in locales that use "," for a decimal point.
- Significant performance improvements related to the case of VOTable documents containing many tables.

Version 2.1-2 (24 March 2010)

- JyStilts introduced. This is a jython (i.e. Python, though not CPython) interface to the STILTS commands. It is believed to be fully working, but somewhat experimental - feedback is encouraged.
- Considerable performance and scalability improvements to the crossmatching commands (`tmatch1`, `tmatch2`, `tmatchn` and `tskymatch2`). For several common regimes, using default settings, memory use has been decreased by a factor of about 5, and CPU time reduced by a factor of about 3.
- Add optional tuning parameters to crossmatch commands (parameter `tuning` for `tmatch1`, `tmatch2` and `tmatchn`, and parameter `healpixk` for `tskymatch2`). Experimentation with these can lead to significant performance improvements for given matches.
- Fixed a crossmatch bug; it was giving a possibility of suboptimal "find=best" match assignments when pair matching in crowded fields. Crossmatch results thus may differ between earlier versions and this one. Both are reasonable, but the newer behaviour is

- more correct. In non-crowded fields, there should be no change.
- Further performance improvement for VOTable documents with very many TABLEs.
 - Memory management adjusted further - default (Adaptive) storage policy now uses direct allocation (`=malloc()`) for intermediate-sized buffers to avoid running out of java heap space.
 - New option "find=each" for `coneskymatch` and `sqlskymatch` commands. This allows you to get an output table with exactly one row for each row of the input table.
 - New flag `-memgui` to monitor memory usage during runs.
 - Add new filter `rowrange`.
 - Add new functions to Arrays: `array` functions for constructing arrays, and new aggregating functions `median` and `quantile`.
 - Syntax of the crossmatching commands' `progress` parameter has changed; it now has an additional option which will write limited profiling information as well as logging as the match progresses.
 - Add `ylabel` parameter to `plothist` command.
 - The `random` and `sequential` filters have been renamed `randomview` and `seqview` respectively. This provides a better idea of what they do. Since they are only useful for debugging, it is unlikely that this will break anyone's existing code.
 - New filter `random` introduced which converts tables to random-access if necessary.
 - Document previously undocumented `legend` parameter to plotting commands.
 - Matching commands `matcher` parameters can now accept classnames of `MatchEngine` implementation classes as an option.
 - Classes are now distributed as a zip of jars (`stilts_jars.zip`) as an alternative to the monolithic jar file (`stilts.jar`). This may be more appropriate for those using STILTS classes in a framework that contains other third party class libraries.
 - Adjusted the way that data types are read from JDBC databases. Date, Time and Timestamp typed columns will now be converted to Strings which means they can be written to most output formats (previously they were omitted from output tables).
 - STILTS no longer attempts to communicate with TOPCAT using SOAP. TOPCAT's SOAP interface has been deprecated since v2.1 (2006), so this isn't likely to cause trouble, and it permits removal of SOAP (Axis) classes from the application jar file, saving several megabytes and reducing potential version clash problems.
 - Fix bug in code for handling very large mapped FITS files. This was causing fatal read errors in some cases.

Version 2.2 (6 August 2010)

New capabilities for multi-table I/O have been introduced:

- New multi-table output tasks `tmulti` and `tmultin`. These currently just copy multiple input tables to a single multi-table container file (e.g. Multi-Extension FITS or multi-TABLE VOTable). Future releases may generalise the output of multi-table processing.
- New `multi` parameter introduced for `tcat` and `tmulti` tasks to pick up all tables in a multi-table container file.
- New JyStilts functions `treads` and `twrites` for multi-table I/O.

There are some additional enhancements:

- Added experimental name-resolution filter `addresolve`; this currently uses Sesame.
- Added filter `repeat`, which repeats table rows a given number of times.

And a number of bug fixes:

- Recognise unofficial column type "long" in IPAC format tables.
- Better behaviour (warn + failover) when attempting to read large files on 32-bit OS or JVM.
- Efficiency warning now issued for large compressed FITS files.
- Upgraded PixTools HEALPix library to 2010/02/09 version. This fixes a bug that could

theoretically cause deficient crossmatch results, though I haven't managed to produce such errors.

- Fixed bug in TST table output.
- Fixed bug in FITS-plus metadata output (table parameters were getting lost).
- Corrected literature references in Fluxes conversion class documentation (thanks to Mattia Vaccari).
- Fixed bug in CSV file parsing that could ignore header row in absence of non-numeric columns.
- Shape and ElSize metadata items now correctly reported by `meta` filter.
- Fixed JyStilts bug when supplying an empty string for a parameter value.

Finally, from this release STILTS requires version 1.5 (a.k.a. 5.0) of the Java J2SE Runtime Environment; it will no longer run on version 1.4, which is now very old. I don't expect this to cause compatibility issues for anyone, but I'm interested to hear if that's not the case.

Version 2.2-1 (23 December 2010)

- Storage management improvements; removed restriction on large (>2Gb) non-FITS datasets in some circumstances.
- Efficiency improvement in sequential mapped access to large FITS files.
- Fix so FITS tables >2Gb can provide random access in 32-bit mode (though slower than 64-bit).
- FITS files now store table names in EXTNAME (and possibly EXTVAR) header cards.
- Window placement for the few GUI tasks should now behave a bit more like platform norms, rather than sitting in the top left hand corner.
- HTML table output now writes cell contents which look like URLs in HTML `<A>` tags.
- Basic authorization (`http://user:pass@host/path`) on table URLs handled.
- Fixed file pointer int overflow bug in FITS MultiMappedFiles.

Version 2.3 (9 May 2011)

TAP

The new commands `tapquery` and `tapresume` have been introduced. These provide support for the Table Access Protocol (TAP), and allow freeform queries in an SQL-like language to be made to remote databases.

Minor enhancements

- Random Groups HDUs are now tolerated, though not interpreted, within FITS files.
- Added `soapout` parameter to `regquery` command.
- Added `count`, `variance` and `stdev` functions to Arrays.
- Upgrade to JSAMP v1.2.
- Improve text rendering in `funcs` window display.
- Attempt case-sensitive matching before case-insensitive for column names.
- Fix `replaceval` filter to work with Infinities.

Bug fixes and workarounds

- JDBC table input handler now effectively downcasts BigInteger/BigDecimal types to Long/Double. The PostgreSQL JDBC driver seems to use the Big* types routinely for numeric values (which I don't think it used to do).
- Add workaround for J2SE bug #4795134, which could cause errors when reading compressed FITS files.
- Fix FITS character handling bug which could cause corrupted FITS files on output in presence of non-ASCII characters.
- Fix (some) JDBC connection leaks.
- Add missing parameters `dashNS` and `linewidthNS` to `plot2d` task.

Version 2.3-1 (30 June 2011)

- Added new command `taplint`. This is a validator for TAP (Table Access Protocol) services. It is only likely to be useful to people developing or operating TAP services.
- ASCII table parsers now understand python-friendly `nan` and `inf` representations.
- Added new constants to expression language `Infinity` and `NaN`.
- Fixed a significant bug in sky crossmatching. If all points in a table were on one side of the RA=0 line, but the error radius extended across that line, matches on the other side could be missed. Matches could also be missed if different tables used different conventional ranges for RA (e.g. -180..180 in one case and 0..360 in another). This fix may in some, but not most, cases result in slower matching than previously.
- Fixed `conesky` cone search verbosity parameter so that `VERB=3` is not erroneously ignored.

Version 2.4 (27 October 2011)**Crossmatching:**

- Two new asymmetric match options `best1` and `best2` have been added for the `find` parameter in the pair matching commands `tmatch2` and `tskymatch2`. They correspond to finding the best match in table B for each row in table A, and in crowded fields often provide more intuitive semantics than the previous symmetric `best` option (in non-crowded fields there is generally no difference). This replicates the matching performed by some other tools, including Aladin.
- New matchers have been added to permit matching of general elliptical, rather than just circular, regions in both planar and sky coordinates; see `2d_ellipse`, and `skyellipse`.
- Another new matcher is available for dealing with per-object errors in Cartesian coordinates (previously per-object errors could only be handled in sky coords); see `Nd_err`.
- Semantics of the `skyerr` matcher have changed slightly.

Expression language functions:

- Algebraic functions involving angles are now mostly available using degrees as well as radians. The `Coords` class has been replaced by `CoordsDegrees` and `CoordsRadians` classes providing sky coordinate functions, and a new class `TrigDegrees` provides normal degree-based trigonometric functions alongside the radian-based versions in `Maths`. Some of the old function names have changed to make clear that they use radians and not degrees. This change should be much more convenient in most cases; sorry it's taken so long to get round to.
- Add new `join` function is added to the `Arrays` class to combine all the elements of an array into a string.

taplint:

There are several bugfixes and changes related to the TAP validator tool `taplint`, mostly thanks to bug reports etc from the TAP community:

- Improve test logic for record limiting queries.
- Errors no longer reported (e.g. E-Qxx-CNAM) for unexpected TAP_SCHEMA table column ordering (when running query stage but no metadata acquisition stages).
- Add new stage MDQ, which checks query result columns for all tables against declared metadata.
- Add check of versioned and unversioned LANG variants.
- Now uses corrected upload ID (`ivo://ivoa.net/std/TAPRegExt#upload-*`) as per most recent TAPRegExt draft.

Bug fixes and minor enhancements:

- Add parameter `parse` to `tapquery` command, allowing pre-send syntax checking of submitted ADQL.
- Add experimental system properties `star.basicauth.user` and `star.basicauth.password`.
- Improve resilience of `coneskyrmatch` in the presence of unreliable or inconsistent DAL services.
- A `PARAMref` element with no referent in a `VOTable` no longer causes an uncaught `NullPointerException`.

Version 2.5 (28 March 2013)

New coverage-related functionality:

- Add new command `pixsample` which can sample pixel data from HEALPix table files (useful for things like Schlegel dust extinction). Also `addpixsample` filter, which does the same job.
- Add new command `pixfoot` which can generate MOC (Multi-Order Coverage) maps.
- Add MOC-based coverage filter to `coneskyrmatch` when using some Cone Search services (mostly VizieR). This uses the Multi-Order Coverage map service operated by CDS. It can make VizieR multi-cone queries much faster by not doing cone searches that are outside the coverage region of the catalogue in question.
- Add new class `Coverage` to the expression language containing MOC-related functions (currently, just `inMoc`).

Other new capabilities:

- Add IPAC table output format.
- Add new class `KCorrections` to the expression language, containing a method for calculating K-corrections following the method of Chilingarian and Zolotukhin.
- You can now reference tables in multi-extension FITS files by name (`EXTNAME` or `EXTNAME-EXTVER`) as an alternative to by HDU index.

VOTable enhancements:

- VOTable input, output and validation are now supported for version 1.3 of the VOTable standard.
- The version of the VOTable format used for VOTable output can now be selected, by using the system property `votable.version`. Output version is VOTable 1.2 by default.
- `votlint` has been changed so that it handles different VOTable versions more capably. Versions 1.1+ are now validated against a schema (which is how those versions are defined) rather than against a DTD hacked to do the same job as the schema. VOTable 1.3 validation is now provided.
- The `votcopy` command has a new `version` parameter to control output version, and a new `nomagic` parameter to control whether `VALUES/null` attributes are removed where appropriate.
- Infinite floating point values are now correctly encoded in VOTable output ("`+Inf`"/"`-Inf`", not "`Infinity`"/"`-Infinity`" as in previous versions).
- `votlint` is now stricter about floating point `TD` element contents.
- VOTable output no longer writes the `schemaLocation` attribute by default.

Other enhancements:

- Add new function `hypot` (`=sqrt(x*x+y*y)`) to the `Maths` class in expression language.
- Add new `split` functions for string splitting to the `Strings` class in expression language.

- Add `-utype` flags for `addcol`, `replacecol`, `colmeta` and `setparam` filters, and `utype` option for `meta` filter.
- Some changes to the `toString` function: it now works on non-numeric values, gives the right answer for `Long` integers and character values, and returns a blank value rather than the string "null" or "NaN" for blank inputs.
- Sexagesimal to numeric angle conversion functions now permit the seconds part of the sexagesimal string to be missing.
- Changes to the IPAC format definition are accommodated: the "long"/"l" type, which is apparently now official, no longer generates a warning, and headers may now use minus signs instead of whitespace.
- Add OBS stage (ObsTAP validation) to `taplint`.
- Add more checks to CAP stage of `taplint`. Declared languages (including features) and output formats are now checked.
- Tidy up error reporting a bit (fewer duplicate nested messages reported).
- PNG graphics output no longer has transparent background.
- Issue a warning for high values of `conesky match parallel` parameter.
- Upgrade JSAMP library to version 1.3-3.
- Upgrade Grégory Mantelet's ADQL library to version 1.1.

Bug fixes:

- Fix serious and long-standing bug (bad TZERO header, causes subsequent reads to fail) for FITS output of boolean array columns.
- Fix small but genuine sky matching bug. The effect was that near the poles matches near the specified threshold could be missed. The bug was in the PixTools library, fixed at the 2012-07-28 release.
- Fix bug in `tmatchn` group mode which could result in output rows with columns from only a single table, i.e. not representing an inter-table match, even when `join*=default`.
- Fix bug which failed when attempting to read FITS files with complex array columns (`TFORMn=rC/rM`).
- Fix failure when caching very large sequential tables.
- Fix bug in `replacecol` and `replaceval` filters which could cause truncation of strings in FITS and possibly VOTable output when the new value was longer than the previously declared maximum length.
- Fix `tcat`, `tcatn` so that in most cases output column metadata is compatible with all input tables, not just the first one in terms of nullability, array shape etc.
- Adjust SQL writer to avoid a type error for MySQL.
- Fix bug in HMS sexagesimal formatting: minus sign was omitted from negative angles. Now the output is forced positive.
- Cope with 1-column CSV files.
- Use the correct form "rows"/"bytes" rather than "row"/"byte" for TAP capability unit values.
- Fix error bar rendering bug which could result in diagonal lines being offset near the edge of plots.

Version 2.5-1 (1 July 2013)

New functionality

- Add read-only support for CDF (NASA Common Data Format) files.
- Add Median Absolute Deviation calculation (`MedAbsDev` and `ScMedAbsDev`) options to `stats` filter.
- Improved handling of HTTP basic authorization. 401s now generate a useful message about the `star.basicauth.*` system properties if they have not been set up.

Bug fixes and minor enhancements

- Fix CSV regression bug introduced at v2.5 - CSV files now work again with MSDOS-style line breaks.
- Fixed FITS output bug which could result in badly-formed string-valued header cards (no closing quote).
- Source code is now managed by git and not subversion. The format of the "Starjava revision" string reported by the `-version` flag has changed accordingly.
- Output mode `meta` now copes better with array-valued table parameters.
- Implemented fixes to reduce the chance of users inadvertently overloading external Cone/SIA/SSA services with multicone-like queries. First, fix it so that abandoned queries are properly terminated, rather than continuing to hit the server until completion or JVM shutdown. Second, implement a sensible default maximum value for the `parallel` parameter of `skyconematch` (though this may be adjusted with a system property).
- Quoting behaviour has changed when generating SQL to write to RDBMS tables. This ought to reduce problems related to mixed-case identifiers. However, it is possible that it could lead to unforeseen new anomalies.
- More `toString` overloads - now works for byte and boolean values too.

Version 2.5-2 (7 March 2014)

- Add some more colour maps.
- Fix some broken and misdocumented non-table-output JyStilts commands (`tcube`, `pixfoot`).
- Fix bug which prevented access to long integer array elements from expression language.
- The Exact matcher now considers scalar numeric values equal if they have the same numeric value; they are no longer required to have the same type.
- Fixed a registry client bug which means that the `regquery` command can now successfully talk to the NVO/VAO/STSci registry. That has been broken since mid-2010.
- Add new command `tloop` for generating single-column tables from a numeric loop variable.
- `taplint` now checks for the right ObsCore ID, though still recognises the wrong one (got from TAPRegExt), and warns if found.
- Fix TST input handler so TST files with fewer than 3 columns can be read.
- Add `Nd_cuboid` matcher option to match commands.

Version 2.5-3 (4 July 2014)**New and improved functionality:**

- Add new command `cdsskymatch`. In most cases (for querying tables that can be found in VizieR) this can and should be used instead of `coneskymatch` - it's *much* faster.
- Commands `coneskymatch`, `sqlskymatch` and `pixfoot` will now guess RA/Dec columns if relevant parameters are left blank.
- Added new graphics output format `png-transp` to generate PNG files with transparent backgrounds.
- Upgraded Gregory Mantelet's ADQL library to version 1.2. Better ADQL parsing.

Improvements and adjustments to `taplint`:

- Rework `taplint` API to facilitate static acquisition of report codes during programmatic use. A few error codes have changed.
- Add new "duff query" test to `taplint`.
- Avoid `taplint` MDQ stage data type mismatch error report for BOOLEAN/boolean declared/returned data.

- `taplint` now takes steps to ensure that TAP_SCHEMA column list query is not truncated.
- `taplint` now flags absence of ObsCore table with I[nfo] not F[ailure] status.
- Change the implementation of `taplint` stages which perform validation against XSD schemas. Schemas from external namespaces may now be imported and used. The CPV stage, which was previously broken and disabled by default, is now fixed and enabled by default. Known/expected schemas are stored locally, and a warning is reported if external ones are used. Schema validation seems remarkably complicated, so it's possible there are still errors in this implementation - if you suspect so, please report it.
- Add missing geometric reserved words to ADQL reserved word list. This fixes some problems with column names like "DISTANCE" in `taplint` tests.
- Fixed some bugs related to TAP table uploads. In particular these could cause incorrect table upload error reports in `taplint`.

Version 3.0 (3 October 2014)

New plotting commands:

A set of new plotting commands are provided which give comprehensive access to all the new-style visualisation capabilities available in TOPCAT v4. These commands are documented in Section 8. These commands, and the underlying visualisation facilities, are considerably more capable than the, now deprecated, old-style plot commands `plot2d`, `plot3d` and `plothist`.

Programmatic invocation:

Programmatic invocation of STILTS tasks from third-party java code is now officially sanctioned and documented in the new Section 11. To support this changes have been made to the parameter system (`Parameter` class now supports generics) and there are some visible changes to the user documentation as well (parameters now report their data type, and tasks report their classname). Normal (e.g. command-line) usage should not undergo any changes, but a fair bit of UI code has changed, so unexpected problems are possible.

Other items:

- Add new output mode `gui`, which displays the table data in a scrollable window on the screen.
- Add new `-allowunused` flag to the `stilts` command. If this is set, then unused parameter settings on the command line just result in a warning, not failure of the command.
- Attempting to write FITS tables with >999 columns now fails with a more helpful error message.
- Improved Unicode handling in VOTables. Fixed a serious bug in `votcopy` that generated unreadable output to BINARY or BINARY2 serialization if any non-empty column had `datatype="unicodeChar"`. Also improved behaviour when copying between tables with `unicodeChar` columns; these are usually preserved now, rather than being squashed to `datatype char`. Some lurking Unicode-related issues remain.
- The TAP client now tolerates whitespace around UWS status codes.
- `taplint`: downgrade unknown post-table QUERY_STATUS value message from Error to Warning.

Version 3.0-1 (13 November 2014)

New functionality:

- Add (experimental) read-only support for Gaia/DPAC GBIN format.

- Add new task `tapskymatch`.
- Functions in class `Coverage` adjusted: new function `nearMoc`, and MOC can be identified by VizieR table IDs as well as by filename/URL.
- For `repeat` filter, add `-row|-table` flags to control sequence of output rows.
- For `setparam` and `repeat` filters, allow use of an algebraic expression for values, not just a literal value.
- Add special values `$ncol` and `$nrow` to the expression language to refer to the column and row counts in a table. The special variable `index` is also deprecated in favour of `$index` or `$0`.

Bugfixes and minor improvements:

- Add some more colour maps for aux/density shading.
- Fix `stilts` invocation script to pick up classes from `stilts.jar` in script directory in preference to other places (e.g. `topcat-full.jar`).
- Fix `taplint` to permit application/xml not just text/xml content-type where appropriate (UWS stage).
- Fix `taplint` so it doesn't warn (W-TMV-UNSC) about unknown VOSITables schema.
- Fix `taplint` so that `unicodeChar` matches CHAR/VARCHAR in the same way as `char` for column type declaration purposes.
- Fix `taplint` so that capabilities document can have TAPRegExt dataModel ivo-id elements with `xs:anyURI` rather than `vr:IdentifierURI` (only a warning is issued in the latter case), in anticipation of TAPRegExt-1.0 Erratum #1.
- Adjust `taplint` to handle `adql:TIMESTAMP` columns more carefully on upload and retrieval.
- Update JSAMP to v1.3.5.

Version 3.0-2 (6 February 2015)

Plotting enhancements:

- Linear fitting of points is now available using the `linearfit` layer type for `plot2plane`. Points may be weighted.
- You can add titles to plots using the new `title` parameter.
- New plot layer type `sizexy` allows plotting (optionally autoscaled) markers with horizontal and vertical extents independently determined by input data.
- More flexibility when assigning colour maps, in aux and density shading modes, and spectrogram layer. New parameters `*func` allow assignment of different data->ramp mapping functions (`sqrt` and `square` as well as `linear` and `logarithmic`), and new parameters `*quant` allow quantisation of the colour map to discrete levels.
- Replace `maxsizeN` parameter with `autoscaleN` for `size` plot layer type. You can now optionally turn off autoscaling and specify marker size in pixels instead.
- Add `auxcrowd` parameter to `plot2` tasks to influence tick crowding on aux axis colour ramp. Also adjust default to use fewer ticks.
- Add some "dart" options (fixed-base open or filled triangles) for plotting vectors (see `arrowN` parameter in layers like `xyvector`).
- Add some "triangle" options (variable-base open or filled triangles) for plotting ellipses (see `ellipseN` parameter in layers like `xyellipse`).
- Histogram normalisation option adjusted so that total area under bars, rather than total height of bars, is fixed.
- The `PlotDisplay` class that forms the result of `plot2` commands can now have `PointSelectionListeners` registered on it. This lets you determine what point a user has clicked on if you're using the plotting classes from third party java code.

FITS I/O:

- Reworked part of the FITS table input implementation, in particular adjusting the way memory mapping is done to reduce resource requirements on some platforms. If you notice any difference, it should be reduced virtual and perhaps resident memory usage, and some (~10%?) performance improvements, when reading large FITS/colfits files. If you were previously having problems with large memory allocations leading to disk thrashing and system lockup when scanning files larger than RAM (this didn't happen on all OSes), these will hopefully have gone away. However, please report anything that appears to be working worse than before, or continued memory usage issues.
- Colfits files can now be accessed from streams, not just uncompressed disk files (though that's not necessarily a good idea).

Bugfixes and workarounds:

- Fixed a query bug (missing `REQUEST=queryData` parameter) in the multi-SSA mode (`servicetype=ssa`) of `coneskyrmatch`. This long-standing bug would have stopped this command working at all with well-behaved SSA services.
- Fixed error in fits-var output (PCOUNT header card did not include block alignment gap).
- Graphics coordinates are now calculated in floating point rather than as integers. This fixes problems that could cause scaled vectors, ellipses etc to be drawn with shapes or orientations badly wrong due to rounding errors. It also improves plotting of analytic functions, especially to vector contexts (PDF/EPS).
- Fix some problems to do with zooming to very large/small plot axis ranges.
- Hide error bars (etc) that would extend to negative values on logarithmic axes; previously they were being drawn in anomalous places.
- Fix `NullPointerException` bug when null value was supplied to multi-word parameter (e.g. `tcube`).
- Fix Aux axis positioning for 3D plots so that the numeric labels don't get snipped off at top and bottom.
- Add a hack that allows LDAC FITS tables to be treated sensibly in auto-format-detection mode.
- Make `VOTable` handling more robust against unknown (illegal) datatypes.
- Add missing parameters `auxmax`, `auxmin` to plotting task documentation.

Version 3.0-3 (14 April 2015)

- New System Command option for input table syntax; you can now use "`<syscmd`" or "`syscmd|`" to supply input byte streams from `Un*x` pipelines.
- Add new Kernel Density Estimate plot layer types `kde`, `knn` and `densogram` for `plot2plane`.
- More histogram normalisation options provided. Instead of just `true/false`, the `normalisation` parameter of the histogram layer now has the options `none`, `height`, `area` and `maximum`. This allows both the area normalisation introduced in v3.0-2, and the height normalisation used in earlier versions which it replaced.
- More histogram bar style options provided; the histogram `barform` parameter now provides the options `semi_filled` (the new default) and `semi_steps`. These give outlined partially transparent bars, which make it much easier to see what's going on in multi-dataset histograms. Note `semi_steps` does not currently export very nicely to PDF/EPS. Similar options are also available in the new KDE plots.
- Column data read in as unsigned bytes will now be written out as unsigned bytes where the output format permits; previously they were forced to 16-bit signed integers. This affects FITS, `VOTable` and CDF I/O handlers.
- Add `count_rows()` method to `JyStilts` table objects, which for non-random tables may be much more efficient than `len()`.

- Be less strict about recognising colfits files (tolerate implicit TDIMn headers).
- `taplint` is now aware of, and performs some checks related to, schema-level table metadata declared by TAP services.
- Work round FITS read bug that could cause problems for VOTables using inline FITS serialization, and possibly elsewhere.
- Fix bug that caused trouble when auto-ranging a plot with a single sky position.

Version 3.0-4 (17 August 2015)

Bugfixes (some significant):

- Fix a serious bug in processing of FITS bit vector (`TFORMn='rX'`) columns. Values read from these columns are presented as a `boolean[]` array. In all previous versions of STIL the bits have appeared in that array in the wrong sequence (LSB..MSB per byte rather than the other way round). Apologies to anyone who may have got incorrect science results from this error in the past, and thanks to Paul Price for helping to diagnose it.
- Fix a less serious bug with `TFORMn='rX'` processing; attempting to read a single-element bit vector column (`TFORMn=1X` or `X`) previously resulted in an error making the file unreadable. Values read from such columns are now presented as Boolean scalars.
- Fix a VOTable reading bug relating to bit vector data (`datatype="bit"`) appearing in BINARY/BINARY2 serializations. This one was more obvious, it would usually generate an error when attempting to read the file.
- Fix serious bug in time conversion for CDF TIME_TT2000 data types.
- Fix a bug in `votcopy` that converted columns from `datatype unsignedByte` to `short` when transforming. Since v3.0-3 this is no longer necessary. In the case of converting to a binary serialization, since v3.0-3 this was causing it to generate unreadable VOTable output.
- Fix a bug in `votcopy` that failed to handle columns with `datatype bit`. In the case of converting to a binary serialization, these were in all previous versions generating unreadable VOTable output. Now they convert them to columns with `datatype boolean` (not perfect, but better).
- Fix `skyvector` bug: `dlat` and `dlon` values were being used the wrong way round.
- Upgrade JEL to v2.0.2. Fixes problem with evaluating void-typed expressions, and possibly some other obscure bugs.
- Some `taplint` bug fixes.

Behaviour changes:

- Changes to the way that TAP service table/column name reports are interpreted (to conform to original intention of TAP standard). `Taplint` now checks that table/column names from `TAP_SCHEMA` and `/tables` endpoint are regular-or-delimited-identifiers, but no longer submits example queries using supplied column names wrapped in additional quotes.
- Modify the heuristics that determine whether the first row of a CSV file is a header.

Enhancements (mostly minor):

- Added some functions to the `Arrays` class that return array-valued results from array-valued parameters: `add`, `subtract`, `multiply`, `divide`, `reciprocal`, `condition`.
- Improve error reporting in the face of non-VOTable TAP error responses. In many cases this makes it much easier to see what's going wrong with a TAP query.
- As a diagnostic tool, when making TAP queries, a log message giving a roughly equivalent `curl(1)` command is now issued at the CONFIG level (visible using flags `-verbose -verbose`).
- New `taplint` parameter `maxtable` limits the number of tables tested in the stage that queries data from each individual table (`MDQ`). May be useful for very large services.

- New `tapquery` parameter `upvotformat` to determine what VOTable serialization variant is used to transmit uploaded tables to the TAP server. Previously uploads were always BINARY which ought to work, but the parameter now defaults to TABLEDATA, since some services (e.g. CADC) currently fail with binary uploads.
- Minor improvement to version reporting (reports java specification version, no longer issues warning for absent revision string).
- Update JCDF library to v1.1 (minor changes to do with leap seconds).

Version 3.0-5 (22 October 2015)

- Fix error reporting bug when a non-VOTable response is received from a TAP service.
- Upgrade to JCDF v1.2 - fixes a bug when reading large (multi-2Gb) CDF files.
- Added source code for an example basic GUI plot application, `uk.ac.starlink.ttools.example.BasicPlotGui`.
- The expression language has a new way of referring to a column; if you use the form `"Object$ <column-id>"` you get the value as an Object not a primitive. This is a special-interest measure for user-defined functions that need to see null numeric values.
- Adjust GBIN input handler: avoid descending into Class-typed members of gbin list objects, and add logging for object->column translations.

Version 3.0-6 (27 November 2015)

Crossmatching bug fix

Fix a long-standing crossmatch bug relating to range restriction during pre-processing. This could have caused missed associations (but not false positives) near the edge of coverage regions when using per-row errors, if the scale of the errors differed (especially differed significantly) between the matched tables. It affected `matcher` values of `<n>-d_err`, `skyerr`, `2d_ellipse` and `skyellipse` only. Thanks to Grant Kennedy (IoA) for reporting this bug.

Density plots

Some more options for making weighted density plots have been added. Since v3.0 the Density shading mode has let you see the density of plotted points, but this lacked some features. Three new ways to do density plots are added; these all give you the option of weighting by an additional coordinate (like the Aux mode), choosing the combination method (mean, median, sum, max, ...), and displaying the quantitative value-colour mapping on the shared colour ramp (previously aux axis) beside the plot. The new density plots are:

- Weighted shading mode, using shaped marker kernels on the screen pixel grid, available for all plot types
- SkyDensity layer, using HEALPix bins on the celestial sphere, for the Sky plot
- Density layer, using square N*N screen pixel bins, for the Plane plot

The details are somewhat experimental and may undergo some adjustments in future releases (feedback welcome).

Colour maps

There are various changes affecting selection and display of colour maps used for density and aux axis shading:

- The default colour map for Aux mode, and other layers using the shared colour map, is no longer Rainbow! It's Inferno. Rainbow colour maps are much hated by visualisation experts. Of course you can still choose Rainbow if you like.
- Add some new colour maps: *Viridis*, *Inferno*, *Magma* and *Plasma* from Matplotlib 1.5, the *SRON* rainbow variant developed by Paul Tol, some diverging maps (*HotCold*, *RdBu*, *PiYG*, *BrBG*) and a qualitative constant chroma/luminance map *HueCL*.

- The options for Density and Aux shading are now mostly the same as each other except where there's good reason to differ. Previously they were different in haphazard ways.
- An attempt is made to give the default form of each colour map a sensible name, without leading minus signs.
- Fix it so that the whole range of each map is distinguishable from white. This is a good idea when you're plotting symbols on a white background, which is common in stils. Perhaps there are cases it's not such a good idea; if you think so, complain and I may change it back.
- Try to fix it so that all the colour maps go in the same direction (light->dark) where applicable.
- Throw out a couple of particularly useless colour maps.
- Colour map ramp display is now different for non-absolute maps; their effect is shown on a selection of base colours, not just for one base colour.

Minor items

- Try harder to identify epoch columns (suitable for time plot), in particular look for VOTable `xtype` of JD or MJD, and `units` of year.
- Add some functions to the Tilings class to do with solid angles (`healpixSqdeg`, `healpixSteradians`, `steradiansToSqdeg`, `sqdegToSteradians`, `SQDEG`).
- Fix plot bug; titles were painted in white for pixel output formats.
- Rationalise plot report logging. Some more diagnostic information about plots is now logged at the INFO level (visible if `topcat` is run with the `-verbose` flag).

Version 3.0-7 (10 June 2016)

Expression Language

The JEL library underlying the expression language parser has been upgraded to v2.1.1 (thanks to Konstantin Metlov), now supporting variable-length argument lists among other things. This allows the following improvements:

- New functions that support any number of arguments are provided: `array`, `intArray`, `stringArray` in class `Arrays`; `concat`, `join` in class `Strings`; and `sum`, `mean`, `variance`, `stdev`, `min`, `max`, `median`, `countTrue` in new class `Lists`.
- Some old lists of similarly-named functions with fixed numbers of arguments have been replaced by single functions that take an arbitrary number of arguments (e.g. `array(x1)`, `array(x1,x2)`, ... `array(x1,x2,x3,x4,x5,x6,x7,x8)` replaced by `array(values...)`).
- The 2-argument `min/max` functions in class `Arithmetic` have been renamed `minNaN/maxNaN` to avoid confusion, but in most cases existing expressions involving `min/max` will work as before.
- Some functions that used to require string arguments will now auto-convert numeric types (e.g. `concat(toString(RA),";",toString(DEC))` can now be written `concat(RA,";",DEC)`).
- You can now implement user-defined functions with variable numbers of arguments.
- Writing large ($\geq 2^{31}$) literal integers used to fail with an inscrutable error message. Now the message tells you to append the "L" character.

Time plots

The `plot2time` command has been enhanced so that it can make *multi-zone* plots - multiple plots stacked vertically that share the same horizontal (time) axis but have independent vertical axes. The time plot itself and this multi-zone feature are currently experimental; in future versions they may be improved or changed, and the multi-zone feature may be extended to other plot types. Some other changes and fixes have gone along with this:

- A few API changes have been made to support multi-zone plots, including generalising the `NavigationListener` interface and rearranging some `PlotDisplay` and `AbstractPlot2Task` constructor/factory method arguments. For single-zone plots the changes are not very substantial. This only affects you if you are using the STILTS classes as a java plotting library. If that applies to you and you have trouble upgrading, I'm happy to provide assistance.
- `plot2time` now supports shading modes (`shadingN` and associated parameters).
- The spectrogram layer now uses the (per-plot, or more precisely now per-zone) Aux colour map rather than a layer-specific colour map. This means that the colour ramp is displayed alongside the plot, but also that some parameters have been renamed (e.g. `auxmapZ` replaces `spectromapN`, where `Z` is an optional zone suffix and `N` is an optional layer suffix).
- The `function` layer type now works in the time plot, rather than throwing an error. However, it doesn't work very well, since the time coordinate is in unix seconds rather than something more user-friendly.
- Fixed a serious bug in ISO-8601 axis labelling. In some cases axis labels were being drawn at positions badly different from the correct position.

Miscellaneous enhancements and changes

- This and subsequent releases target **Java SE 6**, so will no longer run under the (now very ancient) Java 5 runtime.
- Provide more careful documentation of licensing arrangements. The distributed `LICENSE.txt` file notes that the starjava code is LGPL, and documents licenses for each third-party dependency.
- Add Fill layer type for Plane and Time plots.
- `tapquery` and `tapresume` now use blocking HTTP requests rather than repeated polls to wait for asynchronous TAP job completion from services that declare themselves UWS 1.1 compliant.
- Add new parameters `executionduration` and `destruction` to `tapquery` command. These let you request resource limit adjustments when submitting an asynchronous TAP job.
- Improve sky plot border painting.
- Clean up noisy Cubehelix colour map.
- New function `countTrue` in class `Arrays`.
- New stage `EXA` for `taplint` checks `TAP /examples` endpoint. Note the details of the examples format are still under discussion, (this version targets WD-DALI-1.1-20160415 & WD-TAP-1.1-20160428, somewhat informed by TAPNotes-2013-12-13), so the details may change in future.
- `taplint` now validates `ObsCore 1.1` where declared alongside `ObsCore 1.0`. Currently uses `PR-ObsCore-v1.1-20160330`.
- The `taplint` API has changed slightly: the class that used to be `Reporter` is now called `TextOutputReporter`. If you are using `taplint` programmatically you may need to make small changes. This results from some refactoring that makes it easier to customise `taplint` output.
- Replaced `opaque` config option with `transparency` for plane and sky density plots.
- Changed implementation of GIF exporter for plots, from `Acme` to `ImageIO`. Shouldn't be any noticeable difference. `Acme` encoding dependency removed.

Bug fixes

- Fix bug in cumulative histogram calculation.
- Fix read failure for FITS files with non-blank TDIM for zero-length columns.
- Fix bugs that led to timezone-dependent results when reading ISO-8601 or decimal year time columns.
- Fix numeric field truncation bug in LaTeX table output.

- Fix some parameter handling errors in `conesky`match.
- Fix `NullPointerException` bug for disjoint regions in some cases in `tmatchn`.