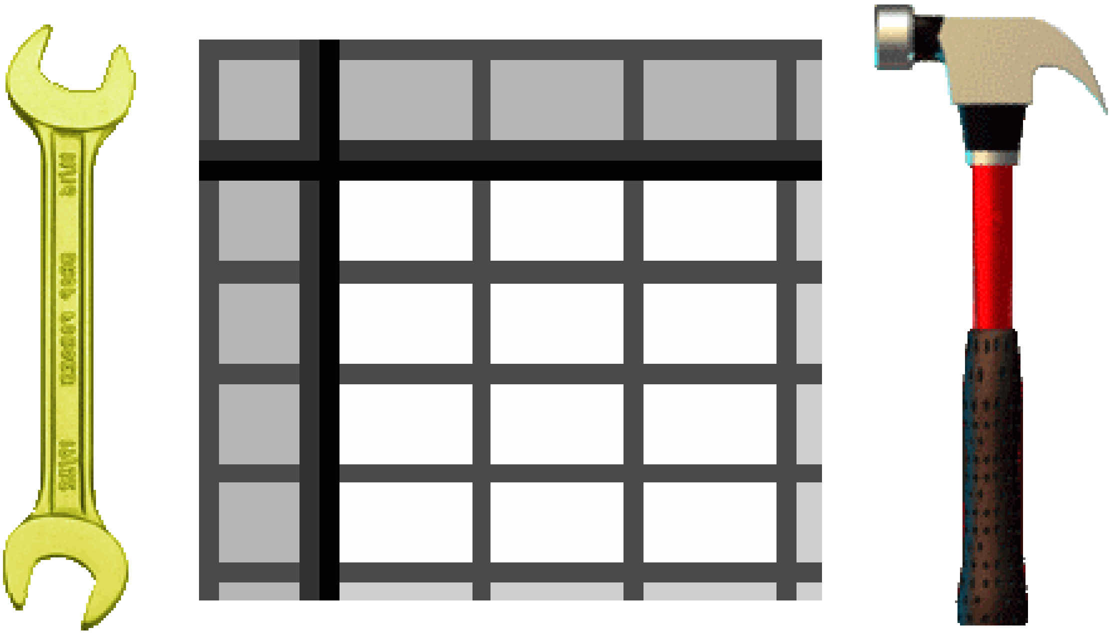
STILTS - Starlink Tables Infrastructure Library Tool Set

Version 1.0b



Starlink User Note 256
Mark Taylor
30 September 2005

$Id: sun256.xml,v 1.26 2005/09/30 15:42:19 mbt Exp $

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 format conversion, format validation, column calculation and rearrangement, row selection, sorting, crossmatching, 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.

STILTS is available under the GNU General Public Licence.

Contents

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 the following commands for generic table manipulation:

- `tcat` (Appendix A.2): Table Concatenater
- `tcopy` (Appendix A.3): Table Format Converter
- `tmatch2` (Appendix A.4): Pair Crossmatcher
- `tpipe` (Appendix A.5): Generic Table Pipeline Utility

the following commands specifically for operating on VOTable files:

- `votcopy` (Appendix A.6): VOTable Encoding Translator
- `votlint` (Appendix A.7): VOTable Validity Checker

and the following general purpose utility:

- `calc` (Appendix A.1): Calculator

More tools may be introduced in the future.

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.

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 A) 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.

`-disk`
Encourages the command to use temporary files on disk for caching large amounts of data rather than doing it in memory. This is a good flag to try if you are running out of memory. This flag is in most cases equivalent to specifying the system property `-Dstartable.storage=disk`.

`-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.

`-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.

`-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.

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 A - currently the available tasks are:

- `calc`
- `tcat`
- `tcopy`
- `tmatch2`
- `tpipe`
- `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`". 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 comand 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)
```

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 A) you can get help for STILTS usage from the command itself. Typing

```
stilts -help
```

results in this output:

```
Usage:
   stilts [-help] [-version] [-verbose] [-disk] [-debug] [-prompt] [-batch]
          <task-name> <task-args>

   stilts <task-name> help[=<param-name>]

   Known tasks:
      calc
      tcat
      tcopy
      tmatch2
      tpipe
      votcopy
      votlint
```

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=]<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=]<in-table>

   Summary:
      Location of input table

   Description:
      The location of the input table. This is usually a filename or URL,
      and may point to a file compressed in one of the supported compression
      formats (Unix compress, gzip or bzip2). If it is omitted, or equal to
      the special value "-", the input table will be read from standard
      input. In this case the input format must be given explicitly using
      the ifmt parameter.
```

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 A).

3 Invocation

There are a number of ways of invoking the `stilts` command, depending on how you have installed the package. 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
```

to extract it (if you don't have an `unzip` command you can do `jar xvf stilts.jar stilts; chmod +x 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 7.6.3) 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
```

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 7.6.3)
- 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.

Note however that encouraging STILTS to use disk files rather than memory for temporary storage is often a better idea than boosting the heap memory - this is done by specifying the `-disk` flag (`stilts -disk <task-name> ...`), or possibly setting the system property `-Dstartable.storage=disk` (see Section 2.1).

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.

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 called `.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:

`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 7.6.3).

`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).

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

`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.

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 PostreSQL 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 \
       "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.TableCopy \
     "jdbc:mysql://localhost/db1#SELECT id, ra, dec FROM gsc WHERE mag < 9" \
     ofmt=votable 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 Table Formats

The generic table commands in STILTS (currently `tpipe`, `tcopy`, `tcat` and `tmatch2`) 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.

4.1 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. By default the first table HDU in the file will used, but this can be altered by supplying the HDU index after a '#' sign, so "table.fits#3" means the third HDU extension.

`votable`
VOTable format - any legal version 1.0 or 1.1 format VOTable documents, and many illegal ones, can be read. By default the first `TABLE` element 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.

`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.

`wdc`
World Datacentre Format (experimental).

For more details on these formats, see the descriptions in SUN/252.

In some cases (when using VOTable or FITS 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.

4.2 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, first extension contains a FITS binary table (behaves the same as `fits-basic` for most purposes)

`fits-basic`
FITS file; primary HDU is data-less, first extension contains a FITS binary table

`votable-tabledata`
VOTable document with TABLEDATA (pure XML) encoding

`votable-binary-inline`
VOTable document with BINARY-encoded data inline within a `STREAM` element

`votable-binary-href`
VOTable document with BINARY-encoded data in a separate file (only if not writing to a stream)

`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

`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/252.

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

5 Table Pipelines

Several of the tasks available in STILTS take one or more input tables, do something or other with them, and produce an output table. 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 using 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`, `tcat`) have parameters called `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 called `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 called `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 5.1 lists the processing steps available, and explains how to use them, Section 5.2 and Section 5.3 describe the syntax used in some of these filter commands for specifying columns, and Section 5.4 describes the available output modes. See the examples in the command reference, and particularly the `tpipe` examples (Appendix A.5.2), for some examples putting all this together.

5.1 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

```
stilts tpipe cmd='delcols 1; delcols 1; 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 syntax of some of these arguments is described elsewhere in this document:

- `<col-id>`: see Section 5.2
- `<colid-list>`: see Section 5.3
- `<expr>`: see Section 7

```
addcol [-after <col-id> | -before <col-id>]
       [-units <units>] [-ucd <ucd>] [-desc <description>]
       <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` and `-desc` flags can be used to define metadata values for the new column.

```
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

- `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.

`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.

`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.

`delcols <colid-list>`

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

`every <step>`

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

`explodeall`

Replaces any column which is an N-element array with N scalar columns. Only columns with fixed array sizes are affected.

`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.

`head <nrows>`

Include only the first `<nrows>` rows of the table.

`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.

`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.

`random`

Ensures that steps downstream see the table as random access. Only useful for debugging.

```
replacecol [-name <name>] [-units <units>] [-ucd <ucd>] [-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` and `-desc` flags; for any of these items which you do not specify, they will take the values from the column being replaced. You can reference the replaced column in the expression, so for example "`replacecol pixsize pixsize*2`" just multiplies the values in column `pixsize` by 2.

`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.

`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).

`sequential`

Ensures that steps downstream see the table as sequential access. Only useful for debugging.

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

Sorts the table according to the columns named in `<colid-list>`. One or more columns may be specified; sorting is done on the values in the first-specified field., but if they are equal the tie is resolved by looking at the second-specified field, and so on. If the `-down` flag is used, the sort order is descending rather than ascending. Blank entries are usually 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.

`sortexpr [-down] [-nullsfirst] <expr>`

Sorts the table according to the value of an algebraic expression. `<expr>` 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 usually 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.

`tablename <name>`

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

`tail <nrows>`

Include only the last `<nrows>` rows of the table.

5.2 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

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.

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.

5.3 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 a 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

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.

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.

5.4 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 are listed below.

`mode=count`

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

`mode=meta`

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

`mode=out`

Writes a new table.

Additional parameters for this output mode are:

`out = <out-table>`

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>`

Specifies the format in which the output table will be written (one of the ones in Section 4.2 - 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)`]

`mode=stats`

Calculates and displays univariate statistics for each of the numeric columns in the table. Mean, standard deviation, minimum, maximum and number of good rows are shown.

`mode=topcat`

Displays the output table directly in TOPCAT. If a TOPCAT instance (version 1.6 or later) is already running on the local host, the table will be opened in that, otherwise a new TOPCAT instance will be launched for display. The latter mode only works if the TOPCAT classes are on the class path. There are currently limits to the size of table that can be transmitted to the application in this way - it is hoped that this can be improved in a future release.

`mode=tosql`

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

Additional parameters for this output mode are:

`protocol = <jdbc-protocol>`

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 `postgres`. For other drivers, you may have to consult the driver documentation.

`host = <value>`

The host which is acting as a database server. [Default: `localhost`]

`database = <db-name>`

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

`newtable = <table-name>`

The name of the new table which will be written to the database. If a table by this name already exists, it may be overwritten.

`user = <username>`

User name for the SQL connection to the database. [Default: `mbt`]

`password = <passwd>`

Password for the SQL connection to the database.

6 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 A.

Matching can in general be a computationally intensive process. The algorithm used by 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 only crossmatching task is `tmatch2` which finds matches between pairs of tables. In future versions however facilities for finding matches within the same table, and in more than two tables, will be introduced.

6.1 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....

To determine the matching criteria, you set the values of the following parameters of `tmatch2`:

`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 7) 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`.

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` and `values*` parameters.

Here is a list of all the basic `matcher` types and the requirements of their associated `params` and `values*` parameters. The units of the required values are given where significant.

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

Comparison of positions on the celestial sphere with a fixed error radius. Rows are considred to match when the two `ra`, `dec` positions are within `max-error` arcseconds of each other along a great circle.

```
matcher=skyerr values*='<ra/degrees> <dec/degrees> <error/arcsec>'
               params='<max-error/arcsec>'
```

Comparison of positions on the celestial sphere with per-row error radii. Rows are considered to match when the separation between the two `ra`, `dec` positions is smaller than *both* the fixed `max-error` value *and* the sum of the two per-row `error` values. If either of the `error` values is blank, then any separation up to `max-error` is considered a match. According to these rules, you might decide to set `max-error` to an arbitarily large number so that only the sum of `errors` will determine the actual match criteria. However please *don't* do this, since `max-error` also functions as a tuning parameter for the matching algorithm, and ought to be reasonably close to the actual maximum acceptable separation.

```
matcher=sky3d values*='<ra/degrees> <dec/degrees> <distance>'
              params='<error/Units of distance>'
```

Comparison of positions in 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`.

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

Comparison of 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.

```
matcher=1d values*='<x>'
           params='<error>'
```

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

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

Comparison of 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`.

```
matcher=Nd values*='<x> <y> ...'
         params='<error>'
```

Comparison of positions in N-dimensional Cartesian space. As for `matcher=2d`, but specify `matcher=3d` or whatever and the corresponding number of entries in the `values*` parameters.

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

Comparison of 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 radii `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.

```
matcher=Nd_anisotropic values*='<x> <y> ...'
                       params='<error-in-x> <error-in-y> ...'
```

Comparison of positions in N-dimensional Cartesian space using an anisotropic metric. As `matcher=2d_anisotropic`, but specify `matcher=3d_anisotropic` or whatever and the corresponding number of entries in the `values*` and `params` parameters.

In addition to those matching criteria listed above, 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 following can be used:

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

Comparison of 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 Algebraic Expression Syntax

The `tpipe` command allows you to use algebraic expressions when making row selections and defining new synthetic columns. They can also be used in defining the quantities to match against in `tmatch2`. In both 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 intutitive.

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 in one of `tpipe`'s commands, e.g. to define the per-row value of a new column (`addcol`, `replacecol`) make a row selection (`select`), and some other places. 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 7.6.3.

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.

7.1 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 two 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 or numbers. In particular it cannot have any spaces in it. The underscore and currency symbols count as letters for this purpose. 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.

There is a special column whose name is "Index" and whose ID is "$0". The value of this is the same as the row number (the first row is 1).

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.

7.2 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 7.5.

7.3 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)

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 7.4.3) conversion functions more convenient for numeric conversions than these.

Other

`+` (string concatenation)
`[]` (array dereferencing)
`?:` (conditional switch)
`instanceof` (class membership)

7.4 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,JMAG)
```

will give you the larger of the values in the columns IMAG and JMAG, and so on.

The functions available for use by default are listed by class in the following subsections with their arguments and short descriptions.

7.4.1 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'.

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

`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`

`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://java.sun.com/j2se/1.4.2/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 patttern
- 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

7.4.2 Strings

String manipulation and query functions.

`concat( s1, s2 )`

Concatenates two strings. In most cases the same effect can be achieved by writing `s1+s2`, but blank values can sometimes appear as the string "`null`" if you do it like that.

- `s1` *(String)*: first string
- `s2` *(String)*: second string
- return value *(String)*: `s1` followed by `s2`

`concat( s1, s2, s3 )`

Concatenates three strings. In most cases the same effect can be achieved by writing `s1+s2+s3`, but blank values can sometimes appear as the string "`null`" if you do it like that.

- `s1` *(String)*: first string
- `s2` *(String)*: second string
- `s3` *(String)*: third string
- return value *(String)*: `s1` followed by `s2` followed by `s3`

`concat( s1, s2, s3, s4 )`

Concatenates four strings. In most cases the same effect can be achieved by writing `s1+s2+s3+s4`, but blank values can sometimes appear as the string "`null`" if you do it like that.

- `s1` *(String)*: first string
- `s2` *(String)*: second string
- `s3` *(String)*: third string
- `s4` *(String)*: fourth string
- return value *(String)*: `s1` followed by `s2` followed by `s3` followed by `s4`

`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`

`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 an uppercased version of a string.

- `str` *(String)*: input string
- return value *(String)*: uppercased 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

7.4.3 Maths

Standard mathematical and trigonometric functions.

`E`

Euler's number *e*, the base of natural logarithms.

`PI`

Pi, the ratio of the circumference of a circle to its diameter.

`RANDOM`

Evaluates to a random number in the range 0<=x<1. This is different for each cell of the table. The quality of the randomness may not be particularly good.

`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.

`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 .

7.4.4 Coords

Functions for angle transformations and manipulations. In particular, methods for translating between radians and HH:MM:SS.S or DDD:MM:SS.S type sexagesimal representations are provided.

`DEGREE`

The size of one degree in radians.

`HOUR`

The size of one hour of right ascension in radians.

`ARC_MINUTE`

The size of one arcminute in radians.

`ARC_SECOND`

The size of one arcsecond in radians.

`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)*: HMS-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.

- `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.

- `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

`skyDistance( ra1, dec1, ra2, dec2 )`

Calculates the separation (distance around a great circle) of two points on the sky.

- `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

`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

`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

`raFK4toFK5( 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)

`decFK4toFK5( 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)

`raFK5toFK4( 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)

`decFK5toFK4( 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)

`raFK4toFK5( raFK4, decFK4, bepoch )`

Converts a B1950.0 FK4 position to J2000.0 FK5 yielding Right Ascension. This assumes zero proper motion in the FK5 frame. The `bepoch` 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)
- `bepoch` *(floating point)*: Besselian epoch
- return value *(floating point)*: right ascension in J2000.0 FK5 system (radians)

`decFK4toFK5( raFK4, decFK4, bepoch )`

Converts a B1950.0 FK4 position to J2000.0 FK5 yielding Declination. This assumes zero proper motion in the FK5 frame. The `bepoch` 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)
- `bepoch` *(floating point)*: Besselian epoch
- return value *(floating point)*: declination in J2000.0 FK5 system (radians)

`raFK5toFK4( raFK5, decFK5, bepoch )`

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)

`decFK5toFK4( raFK5, decFK5, bepoch )`

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)

7.4.5 Conversions

Functions for coverting between strings and numeric values.

`toString( value )`

Turns a numeric value into a string.

- `value` *(floating point)*: numeric value
- return value *(String)*: a string representation of `value`

`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

7.4.6 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

`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.

- `a` *(integer)*: an argument.
- `b` *(integer)*: another argument.
- return value *(integer)*: the larger of `a` and `b`.

`max( 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`.

`min( a, b )`

Returns the smaller of two integer values. If the arguments have the same value, the result is that same value.

- `a` *(integer)*: an argument.
- `b` *(integer)*: another argument.
- return value *(integer)*: the smaller of `a` and `b`.

`min( 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`.

7.5 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 radians

```
hmsToRadians(RA1950)
dmsToRadians(decDeg,decMin,decSec)
```

Conversion from radians to sexagesimal

```
radiansToDms($3)
radiansToHms(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
skyDistance(ra0,dec0,degreesToRadians(RA),degreesToRadians(DEC))<15*ARC_MINUTE
```

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
```

7.6 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!

7.6.1 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://galaxy.fzu.cz/JEL/).

7.6.2 Instance Methods

There is another category of functions which can be used apart from those listed in Section 7.4. 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.

7.6.3 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 abritrarily 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 called 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 called `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`"

A 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.

A.1 `calc`: Calculator

`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 7.4. Since no table is involved, you can't refer to column names in the expressions. It takes one parameter, the expression to evaluate, and writes the result to the screen.

A.1.1 Usage

The usage of `calc` is

```
stilts <stilts-flags> calc [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.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`expression = <expr>`
An expression to evaluate. The functions in Section 7.4 can be used.

A.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".

A.2 `tcat`: Table Concatenater

`tcat` is a tool for concatenating tables one after the other. 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 `tcat` 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. This will only work if the columns of the two tables to be joined have the same or compatible types in the same order; if they do not, you must use the `icmd` parameters to preprocess the input tables so that the column sequences are compatible.

In the current release of STILTS, `tcat` is rather rudimentary: you can only join two tables at once, you must arrange for the columns to be in the right order, and you may end up with an unhelpful error if the columns in matching positions are not of compatible types. Behaviour will be improved in a future release.

A.2.1 Usage

The usage of `tcat` is

```
stilts <stilts-flags> tcat ifmt1=<in-format> ifmt2=<in-format> icmd1=<cmds>
                           icmd2=<cmds> ocmd=<cmds>
                           omode=<out-mode> <mode-args> out=<out-table>
                           ofmt=<out-format>
                           [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.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`icmd1 = <cmds>`
Commands to operate on the first input table, before any other processing takes place. The value of this parameter is one or more of the filter commands described in Section 5.1. 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.

`icmd2 = <cmds>`
Commands to operate on the second input table, before any other processing takes place. The value of this parameter is one or more of the filter commands described in Section 5.1. 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.

`ifmt1 = <in-format>`
Specifies the format of the first input table (one of the known formats listed in Section 4.1). This flag can be used if you know what format your input 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>`
Specifies the format of the second input table (one of the known formats listed in Section 4.1). This flag can be used if you know what format your input 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>`
The location of the first input table. This is usually a filename or URL, and may point to a file compressed in one of the supported compression formats (Unix compress, gzip or bzip2). If it is omitted, or equal to the special value "-", the input table will be read from standard input. In this case the input format must be given explicitly using the `ifmt1` parameter.

`in2 = <table2>`
The location of the second input table. This is usually a filename or URL, and may point to a file compressed in one of the supported compression formats (Unix compress, gzip or bzip2). If it is omitted, or equal to the special value "-", the input table will be read from standard input. In this case the input format must be given explicitly using the `ifmt2` parameter.

`ocmd = <cmds>`
Commands to operate 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 5.1. 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.

`ofmt = <out-format>`

Specifies the format in which the output table will be written (one of the ones in Section 4.2 - 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-mode> <mode-args>`

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`, `topcat` and `tosql`. Use the `help=omode` flag or see Section 5.4 for more information. [Default: `out`]

`out = <out-table>`

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: `-`]

A.2.2 Examples

Here are some examples of `tcopy`:

```
stilts tcat obs1.fits 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 tcat ifmt1=ascii in1=obs1.txt ifmt2=ascii in2=ob2.txt omode=stats
```

Again two similar catalogues are joined, but in this case both are in ASCII format, and 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 tcat in1=survey.vot.gz icmd1='keepcols "OBJ_ID RA2000 DEC2000"' \
            in2=more_data.csv icmd2='keepcols "ident ra dec"' ifmt2=csv \
            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: an integer identifier and floating point RA and Dec coordinates. `tcat` joins these 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.

A.3 `tcopy`: Table Format Converter

`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 4.1) to any of the supported output formats (Section 4.2).

`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.

A.3.1 Usage

The usage of `tcopy` is

```
stilts <stilts-flags> tcopy ifmt=<in-format> ofmt=<out-format>
                            [in=]<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.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`ifmt = <in-format>`
Specifies the format of the input table (one of the known formats listed in Section 4.1). This flag can be used if you know what format your input 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 = <in-table>`
The location of the input table. This is usually a filename or URL, and may point to a file compressed in one of the supported compression formats (Unix compress, gzip or bzip2). If it is omitted, or equal to the special value "-", the input table will be read from standard input. In this case the input format must be given explicitly using the `ifmt` parameter.

`ofmt = <out-format>`
Specifies the format in which the output table will be written (one of the ones in Section 4.2 - 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>`
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: -]

A.3.2 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 "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.

A.4 `tmatch2`: Pair Crossmatcher

`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 6.1. 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.

A.4.1 Usage

The usage of `tmatch2` is

```
   stilts <stilts-flags> tmatch2 ifmt1=<in-format> ifmt2=<in-format>
                                 icmd1=<cmds> icmd2=<cmds>
                                 matcher=<matcher-name> values1=<expr-list>
                                 values2=<expr-list> params=<match-params>
                                 join=1and2|1or2|all1|all2|1not2|2not1|1xor2
                                 find=best|all ocmd=<cmds>
                                 omode=<out-mode> <mode-args> out=<out-table>
                                 ofmt=<out-format>
                                 [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.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`find = best|all`

Determines which matches are retained. If `best` is selected, then only the best match between the two tables will be retained; in this case the data from a row of either input table will appear in at most one row of the output table. If `all` is selected, then all pairs of rows from the two input tables which match the input criteria will be represented in the output table. [Default: `best`]

`icmd1 = <cmds>`

Commands to operate on the first input table, before any other processing takes place. The value of this parameter is one or more of the filter commands described in Section 5.1. 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.

`icmd2 = <cmds>`

Commands to operate on the second input table, before any other processing takes place. The value of this parameter is one or more of the filter commands described in Section 5.1. 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.

`ifmt1 = <in-format>`

Specifies the format of the first input table (one of the known formats listed in Section 4.1). This flag can be used if you know what format your input 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>`

Specifies the format of the second input table (one of the known formats listed in Section 4.1). This flag can be used if you know what format your input 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>`

The location of the first input table. This is usually a filename or URL, and may point to a file compressed in one of the supported compression formats (Unix compress, gzip or bzip2). If it is omitted, or equal to the special value "-", the input table will be read from standard input. In this case the input format must be given explicitly using the `ifmt1` parameter.

`in2 = <table2>`

The location of the second input table. This is usually a filename or URL, and may point to a file compressed in one of the supported compression formats (Unix compress, gzip or bzip2). If it is omitted, or equal to the special value "-", the input table will be read from standard input. In this case the input format must be given explicitly using the `ifmt2` parameter.

`join = 1and2|1or2|all1|all2|1not2|2not1|1xor2`

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
- `1or2`: An output row for each row represented in either or both of the input tables

- `all1`: An output row for each matched or unmatched row in table 1
- `all2`: An output row for each matched or unmatched row in table 2
- `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>`

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 6.1. The value supplied for this parameter determines the meanings of the values required by the `params`, `values1` and `values2` parameters. [Default: `sky`]

`ocmd = <cmds>`

Commands to operate 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 5.1. 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.

`ofmt = <out-format>`

Specifies the format in which the output table will be written (one of the ones in Section 4.2 - 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-mode> <mode-args>`

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`, `topcat` and `tosql`. Use the `help=omode` flag or see Section 5.4 for more information. [Default: `out`]

`out = <out-table>`

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>`

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".

`values1 = <expr-list>`

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 7.

`values2 = <expr-list>`

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 7.

A.4.2 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 \
               icmd1='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` still has to contain a value which gives the maximum error for matching (i.e. >= the largest value in the POS_ERR column). 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 int1=cat1.txt in2=cat2.txt \
               matcher=2d values1="X Y" values2="X Y" params="5" join=1and2 \
               ocmd='addcol XDIFF X_1-X_2; addcol Y_1-Y_2' \
               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 called X_1, X_2, Y_1 and Y_2 (along with any others in the input tables) - the _1 and _2 are appended to the names to disambiguate them since the same column names appear in the two input tables. 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).

A.5 `tpipe`: Generic Table Pipeline Utility

`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 5.1.

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 5 for a more detailed explanation of these ideas.

The parameters mentioned above are listed in detail in the next section.

A.5.1 Usage

The usage of `tpipe` is

```
stilts <stilts-flags> tpipe ifmt=<in-format> istream=true|false
                            script=<script-file> cmd=<cmds>
                            omode=<out-mode> <mode-args> 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.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`cmd = <cmds>`
Text of table processing commands. The value of this parameter is one or more of the filter commands described in Section 5.1. 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. The `script` and `cmd` flags should not be mixed in the same invocation.

`ifmt = <in-format>`
Specifies the format of the input table (one of the known formats listed in Section 4.1). This flag can be used if you know what format your input 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>`
The location of the input table. This is usually a filename or URL, and may point to a file compressed in one of the supported compression formats (Unix compress, gzip or bzip2). If it is omitted, or equal to the special value "-", the input table will be read from standard input. In this case the input format must be given explicitly using the `ifmt` parameter.

`istream = true|false`
If set true, the `in` table 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 input table more than once). It is not normally useful or necessary to set this flag - in most cases the data will be streamed automatically if that is the best thing to do. [Default: `false`]

`ofmt = <out-format>`
Specifies the format in which the output table will be written (one of the ones in Section 4.2 - 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-mode> <mode-args>`
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`, `topcat` and `tosql`. Use the `help=omode` flag or see Section 5.4 for more information. [Default: `out`]

`out = <out-table>`
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: `-`]

`script = <script-file>`

Location of a file containing table processing commands. Each line of this file contains one of the filter commands described in Section 5.1. The sequence of commands given by the lines of this file defines the processing pipeline which is performed on the table. The `script` and `cmd` flags should not be mixed in the same invocation.

A.5.2 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 called 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 omode=meta 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.

```
stilts tpipe in=survey.fits
             cmd='select "skyDistance(hmsToRadians(RA),dmsToRadians(DEC), \
                                      hmsToRadians(2,28,11),dmsToRadians(-6,49,45) \
                          < 5 * ARC_MINUTE"' \
             omode=count
```

Counts the number of rows within a given 5 arcmin cone of sky in a FITS table. The `skyDistance` 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 `skyDistance`'s arguments and return value are in radians, some conversions are required: the RA and DEC columns are sexagesimal strings which are converted using the `hmsToRadians` and `dmsToRadians` functions respectively. Different versions of these functions (ones which take numeric arguments) are used to convert the coordinates of the fixed point to radians. The result is compared to a multiple of the `ARC_MINUTE` constant, which is the size of an arcminute in radians. Any rows of the input table for which this comparison is true are included in the output. An alternative function, `skyDistanceDegrees` which works in degrees, is also available. The functions and constants used here are described in detail in Section 7.4.4.

```
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 database=ASTRO1 newtable=TABLEX \
             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 called 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 \
     tpipe 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 tpipe 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.

A.6 `votcopy`: VOTable Encoding Translator

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.

A.6.1 Usage

The usage of `votcopy` is

```
stilts <stilts-flags> votcopy charset=<xml-encoding> cache=true|false
                              href=true|false base=<location>
                              [in=]<location> [out=]<location>
                              [format=]tabledata|binary|fits|empty
```

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.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`base = <location>`
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 no `out=-` (output is to standard out), since in this case there is no default base name to use.

`cache = true|false`
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. [Default: `false`]

`charset = <xml-encoding>`
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|fits|empty`
Determines the encoding format of the table data in the output document. If `empty` 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. [Default: `tabledata`]

`href = true|false`
In the case of BINARY or FITS encoding, this determines whether the STREAM elements output will contain their data inline or externally. If set true, the output document will be self-contained, with STREAM data inline as base64-encoded characters. If false, 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. [Default: `false`]

`in = <location>`
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: -]

`out = <location>`
Location of the output VOTable. May be a filename or "-" to indicate standard output. [Default: -]

A.6.2 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'>
AAAAD0NoYXJsZXMgTWVzc2llcgAAAAtNYXJrIFRheWxvcg==
</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="fcat-1.fits"/>
</FITS>
</DATA>
...
<DATA>
<FITS>
<STREAM href="fcat-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, `fcat-1.fits` and `fcat-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.

A.7 `votlint`: VOTable Validity Checker

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.

A.7.1 Usage

The usage of `votlint` is

```
   stilts <stilts-flags> votlint validate=true|false version=1.0|1.1
                                 [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.

Parameter values are assigned on the command line as explained in Section 2.3. They are as follows:

`validate = true|false`
Whether to validate the input document aganist 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`
Selects the version of the VOTable standard which the input table is supposed to exemplify. Currently the version can be 1.0 or 1.1. 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.

`votable = <location>`
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: `-`]

A.7.2 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 or 1.1 (as appropriate) DTD. Although VOTable 1.1 is properly defined against an XML Schema rather than a DTD, in conjunction with the other checks done, the DTD validation turns out to be pretty comprehensive. Some of the DTD 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 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 `FIELD`s) with the same `name` attributes

A.7.3 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 (l.4): No arraysize for character, FIELD implies single character
ERROR (l.7): Element "TABLE" does not allow "DESCRIPTION" here.
WARNING (l.11): Characters after first in char scalar ignored (missing arraysize?)
WARNING (l.15): Wrong number of TDs in row (expecting 3 found 4)
ERROR (l.18): Row count (1) not equal to nrows attribute (2)
```

Note 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">                               <!-- change row count -->
   <DESCRIPTION>A very small table</DESCRIPTION>   <!-- move DESCRIPTION -->
   <FIELD name="Identifier" datatype="char"
                            arraysize="*"/>        <!-- add 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>                                         <!-- remove extra TD -->
    </TABLEDATA>
   </DATA>
  </TABLE>
 </RESOURCE>
</VOTABLE>
```

When fed this version, `votlint` gives no warnings.

B Release Notes

This is STILTS, Starlink Tables Infrastructure Library Tool Set. It is a collection of non-graphical utilites for general purpose table and VOTable manipulation developed by Starlink.

Author
Mark Taylor (Starlink, 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.

B.1 Acknowledgements

The initial development of STILTS was done under the UK's now-deceased Starlink project, without which it would not have been written.

Apart from the excellent Java 2 Standard Edition itself, the following external libraries provide important parts of TOPCAT's functionality:

- JEL (GNU) for algebraic expression evaluation
- PixTools (Fermilab EAG) for HEALPix-based celestial sphere row matching
- 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 Giaretta (Starlink, RAL)
- Clive Page (AstroGrid, Leicester)

B.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.