STILTS - Starlink Tables Infrastructure Library Tool Set

Version 0.2b

Starlink User Note 256
Mark Taylor
30 June 2005

$Id: sun256.xml,v 1.6 2005/06/30 11:48:23 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 manipulation, row selection, sorting, 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.

STILTS is available under the GNU General Public Licence. This document describes the initial, beta, release of the package.

Contents

Abstract **1**
1 Introduction **4**
2 Invocation **5**
2.1 Class Path 5
2.2 Java Arguments 5
2.3 Command Classname 6
2.4 Application Arguments 6
2.5 System Properties 6
2.6 JDBC Configuration 7
3 Table Formats **9**
3.1 Input Formats 9
3.2 Output Formats 9
4 Algebraic Expression Syntax **11**
4.1 Referencing Column Values 11
4.2 Null Values 12
4.3 Operators 12
4.4 Functions 13
4.4.1 Times 13
4.4.2 Strings 15
4.4.3 Maths 17
4.4.4 Coords 19
4.4.5 Conversions 22
4.4.6 Arithmetic 23
4.5 Examples 25
4.6 Advanced Topics 26
4.6.1 Expression evaluation 26
4.6.2 Instance Methods 27
4.6.3 Adding User-Defined Functions 27

Appendix A: Command Reference..... 29
A.1 tcopy: Table Format Converter..... 29
A.1.1 Usage..... 29
A.1.2 Examples..... 30
A.2 tpipe: Generic Table Pipeline Utility..... 30
A.2.1 Usage..... 31
A.2.1.1 Input Specifier..... 31
A.2.1.2 Filter Specifiers..... 32
A.2.1.3 Mode Specifier..... 33
A.2.1.4 Other Flags..... 34
A.2.2 Examples..... 34
A.3 votcopy: VOTable Encoding Translator..... 36
A.3.1 Usage..... 36
A.3.2 Examples..... 37
A.4 votlint: VOTable Validity Checker..... 39
A.4.1 Usage..... 40
A.4.2 Items Checked..... 40
A.4.3 Examples..... 41
Appendix B: Release Notes..... 43
B.1 Version History..... 43

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 four commands:

- `tcopy` (Appendix A.1): Table Format Converter
- `tpipe` (Appendix A.2): Generic Table Pipeline Utility
- `votcopy` (Appendix A.3): VOTable Encoding Translator
- `votlint` (Appendix A.4): VOTable Validity Checker

It is expected that more will be introduced in the future, in particular a crosssmatching tool and perhaps some more convenience commands which make it easier to do some of the operations provided by `tpipe`.

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, you can pass the results through one of the tools here. Since on the whole operation is streaming, the conversion can easily and efficiently be done on the fly, for instance as part of a web service providing a choice of output formats.

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.

This release of STILTS is beta software. While all the functions are believed to be working, the interface (command line arguments etc) may undergo changes in the future. All user comments are most welcome.

2 Invocation

There are two ways of invoking the commands in this package. If you're using a Unix-like operating system and have downloaded the package in a form which includes the shell scripts, the easiest way is to use the scripts `tpipe`, `votlint` etc. These will either be in the same location as the `stilts.jar` file or in `starjava/bin` The form of invocation in this case is:

```
<command-name> <java-args> <application-args>
```

(the `java` should be on your path).

A simple example would be:

```
votcopy -f binary t1.xml t2.xml
```

For convenience you can mix up java-args and application-args - the script will untangle them and reorder them properly. 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, or if you don't have the STILTS shell scripts, you will need to invoke Java directly with the appropriate classes on your classpath. The general form of an invocation command in this case will depend on your system, but will probably look like:

```
java <java-args> -classpath <command-classpath> <command-classname>
     <application-args>
```

The example above in this case would look something like:

```
java -classpath some/where/stilts.jar uk.ac.starlink.table.VotCopy
     -f binary t1.xml t2.xml
```

More detail is given on the parts of these command lines in the following subsections.

2.1 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.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 2.6)
- Providing extended algebraic functions (see Section 4.6.3)
- Installing I/O handlers for new table formats (see SUN/252)

2.2 Java Arguments

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 2.5. 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 something about an `OutOfMemoryError` 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 poor 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 on most of the tools, or possibly `-Dstartable.storage=disk`.

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.

2.3 Command Classname

Each command in the package is defined at the top level with a single class in the namespace `uk.ac.starlink.ttools`. For instance `tpipe` is defined by the class `uk.ac.starlink.ttools.TablePipe`, and if you don't have the `tpipe` script you can run it using

```
java -classpath some/where/stilts.jar uk.ac.starlink.ttools.TablePipe
```

The classname of each command is listed with its description in Appendix A.

2.4 Application Arguments

The arguments for each application are listed with the command descriptions elsewhere in this document, but some of them are common to most or all of the commands:

`-h[elp]`
Prints a usage message and exits.

`-v[erbose]`
May cause more information about progress to be written as the command runs.

`-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 getting `OutOfMemoryErrors`. This flag is in most cases equivalent to specifying the system property `-Dstartable.storage=disk`.

`-debug`
Causes any error messages, which are usually made brief, to be accompanied by a stack trace. If you are reporting a bug (or debugging the code yourself), then you should use this flag to get the most information about what has gone wrong.

2.5 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 2.2) 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 2.4).

`jdbc.drivers`
Can be set to a (colon-separated) list of JDBC driver classes using which SQL databases can be accessed (see Section 2.6).

`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 4.6.3).

`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 2.4).

`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.

2.6 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 this 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 2.1

2. Set the `jdbc.drivers` system property to the name of the driver class as described in Section 2.5

These steps are all standard for use of the JDBC system. See SUN/252 for information about JDBC drivers known to work with STIL.

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:

```
tcopy -classpath /usr/local/jars/mysql-connector-java.jar \
      -Djdbc.drivers=com.mysql.jdbc.Driver \
      "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
```

In the latter case you have to exercise some care to get the arguments in the right order (see Section 2).

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.

3 Table Formats

The generic commands in STILTS (currently `tpipe` and `tcopy`) 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.

3.1 Input Formats

Some of the tools in this package ask you to specify the format of input tables using the `-f` or `-ifmt` flag. 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, and many illegal 1.0 or 1.1 format VOTable document 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 convensions 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, and may lead to more helpful error messages in the case that the table can't be read correctly.

3.2 Output Formats

Some of the tools ask you to specify the format of output tables using the `-ofmt` flag. 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.

4 Algebraic Expression Syntax

The `tpipe` command allows you to use algebraic expressions when making row selections and defining new synthetic columns. 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 define the per-row value of a new column (`tpipe -addcol`) or a selection flag (`tpipe -select`). 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 4.6.3.

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.

4.1 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.

4.2 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 4.5.

4.3 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 4.4.3) conversion functions more convenient for numeric conversions than these.

Other

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

4.4 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, as well as providing actions to take when a point is activated. 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.

4.4.1 Times

Functions for conversion of time values between various forms. The two main forms used here are Modified Julian Date (MJD) and the string format and underlying calendar model described by ISO 8601. MJD is a continuous measure in days since midnight at the start of 17 November 1858. ISO 8601 format is a string representation of this of the form `yyyy-mm-ddThh:mm:ss.s`, where the `T` is a literal character. In both cases the time is UTC.

Therefore midday on the 25th of October 2004 is 53303.5 as an MJD value and `2004-10-25T12:00:00` in ISO 8601 format.

Currently this implementation does not keep track of values 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`

4.4.2 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

4.4.3 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.

`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 .

4.4.4 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.

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)

4.4.5 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

4.4.6 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`.

4.5 Examples

Here are some examples for defining new columns; the expressions below could appear as the `<expr>` in a `tpipe -addcol` or `-sortexpr` filter specifier.

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` filter specifier)

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 -head 100` instead)

Every tenth row

```
index % 10 == 0
```

(though you could use `tpipe -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
```

4.6 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!

4.6.1 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/).

4.6.2 Instance Methods

There is another category of functions which can be used apart from those listed in Section 4.4. 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.

4.6.3 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 or Starlink's QUICK service). 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 2.1
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 2.5

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

```
tpipe -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 "`tpipe -Djel.classes=AuxFuncs -classpath .`"

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 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 3.1) to any of the supported output formats (Section 3.2).

`tcopy` is designed as a drop-in replacement for the old `tablecopy` (`uk.ac.starlink.table.TableCopy`) tool which was supplied with STIL and TOPCAT - it has the same arguments and behaviour as `tablecopy`, but is implemented on top of `tpipe` and will in some cases be more efficient.

A.1.1 Usage

The basic usage of `tcopy` is

```
tcopy [-ifmt <in-format>] [-ofmt <out-format>] [<other-flags>]
      <in-table> [<out-table>]
```

If you don't have the Unix scripts installed, invoke it as described in Section 2 using the classname `uk.ac.starlink.ttools.TableCopy`.

The most important arguments are as follows:

`-ifmt <in-format>`
: Specifies the format of the input table (one of the known formats listed in Section 3.1 - matching is case-insensitive). This flag can be used if you know what format your input table is in. If it's omitted, then an attempt will be made to detect the format of `<in-table>` automatically, but this cannot always be done correctly, in which case the program will exit with an error explaining which formats were attempted.

`-ofmt <out-format>`
: Specifies the format in which the output table will be written (one of the ones listed in Section 3.2 - matching is case-insensitive and you can use just the first few letters). If this flag is omitted, 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 required, the program will exit with an error.

`<in-table>`
: The location of the input table. This is usually a filename or a URL, and may point to a file compressed in one of the supported compression formats (Unix compress, gzip or bzip2). If it is 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` flag.

`<out-table>`
: The location of the output table. This is usually a filename to write to. If omitted, or if it is equal to the special value "`-`", the output table will be written to standard output. In this case the output format must be given explicitly using the `-ofmt` flag.

The following generic flags can also be used:

`-h[elp]`
: Prints a usage message and exits.

-debug
Causes any error messages, which are usually made brief, to be accompanied by a stack trace. If you are reporting a bug (or debugging the code yourself), then you should use this flag to get the most information about what has gone wrong.

-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 getting `OutOfMemoryErrors`. This flag is in most cases equivalent to specifying the system property `-Dstartable.storage=disk`.

-v[erbose]
May cause more information about progress to be written as the command runs.

A.1.2 Examples

Here are some examples of `tcopy` in use:

`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.

`tcopy -ifmt fits stars.fits -ofmt votable`

Does the same as the previous example, but the input and output formats have been specified explicitly.

`tcopy -ofmt text http://remote.host/data/vizer.xml.gz#4 -`

Prints the contents of a remote 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.

`tcopy -ifmt csv -ofmt latex spec.csv`

Converts a comma-separated values file to a LaTeX table environment.

```
tcopy -classpath /usr/local/jars/pg73jdbc3.jar \
      -Djdbc.drivers=org.postgresql.Driver \
      "jdbc:postgresql://localhost/imsim#SELECT ra, dec, Imag, Kmag FROM dqc" \
      -ofmt fits wfslist.cat
```

Writes the results of an SQL query on a PostgreSQL database to a FITS table. The classpath is augmented to include the PostgreSQL driver class, and the driver class is named using the `jdbc.drivers` system property. If a username or password is required, it will be prompted for on the command line. 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 2.6.

A.2 tpipe: Generic Table Pipeline Utility

`tpipe` is the main tool in this package for general purpose manipulation of tables. 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 in almost any way you can think of. You can think of it as a supercharged table copying tool, or as a way to perform most of the operations that TOPCAT can perform on a table from the command line.

A.2.1 Usage

Using `tpipe` is more like using a Unix pipeline than using a single Unix command. You give an input specifer which determines the table to be operated on, some filter specifiers which determine the operations which will be performed on it, and an output specifier which determines what should happen to the processed data. The table is streamed through the processing filters in the order in which you've given them, and the processed data are eventually sent to the destination, which may be an output table file or some other operation like displaying it in TOPCAT or calculating the per-column statistics. In most cases the processing is passed through the pipeline a row at a time, meaning that the amount of memory required is small (though in some cases, for instance sorting, this is not possible). Although a similar effect could be achieved by stringing together several single-operation `tpipe` invocations in an actual Unix pipe, with the data flowing between the commands in byte streams using one or other of the supported table formats, the way `tpipe` works makes it much more efficient to do all the work within one invocation.

The basic form of a `tpipe` invocation is therefore

```
tpipe <input-specifier> <filter-specifiers> <mode-specifier> <other-flags>
```

(though in fact these elements can appear in any order and they are all optional). If you don't have the Unix scripts installed, invoke it as described in Section 2 using the classname `uk.ac.starlink.ttools.TablePipe`.

The different sets of arguments are described in the following subsections. There are many different flags, some with supplementary arguments, which may look daunting. However, if you make an error in specifying them, `tpipe` will try to print a message which explains what has gone wrong, so with a little bit of trial and error it should be possible to make it do what you want.

A.2.1.1 Input Specifier

The input specifier determines the input table on which the processing will be performed. It has the form:

```
[-ifmt <in-format> [-stream]] <in-table>
```

which is interpreted as follows:

`-ifmt <in-format>`
`<in-format>` is the name of one of the input formats described in Section 3.1. If the `-ifmt` flag is not used, auto format-detection is used (OK for FITS and VOTables). For other formats, such as CSV or ASCII, you must name a format using this flag.

If you give an unknown format (e.g. `-ifmt help`) a list of the formats that are known will be printed.

`-stream`
This flag can be specified to ensure that the input table is read as a stream. You need the `-ifmt` flag in this case. Depending on the required operations and processing mode, this may fail (sometimes you need to read the input file more than once) - if so specifying `-cache` near the start of the filter specifiers may help. It is not normally necessary to specify this flag - in most cases the data will be streamed automatically if that is the best thing to do.

<in-table>
Names the input table. May be "-" to specify standard input (in which case -stream is implicit).

A.2.1.2 Filter Specifiers

The filter specifiers each specify a processing step which is performed on a table, transforming an input table to an output one. You can have any combination of them, and they are used in the order that they are given on the command line. They are like filter-type commands in a Unix pipeline. Some of them have additional optional or mandatory arguments.

-select <expr>
Include in the output table only rows for which the given expression <expr> evaluates to true. <expr> is an expression using the syntax described in Section 4 with a boolean-type value.

-sort [-down] [-nullsfirst] <colid-list>
Sorts the table according to the columns named in <colid-list>. <colid-list> is a space-separated list of column identifiers (names, $IDs or numbers, where 1 is the first column). 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 instead of ascending. If the -nullsfirst flag is used, blank entries are considered to come at the start of the collation sequence instead of the end.

-sortexpr <expr>
Sorts the table according to the value of an algebraic expression. The syntax of <expr> is described in Section 4. Its value must be of a type that it makes sense to sort, for instance numeric.

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

-head <nrows>
Include only the first <nrows> rows of the table.

-tail <nrows>
Include only the last <nrows> rows of the table.

-addcol [-after <col-id> | -before <col-id>] <col-name> <expr>
Add a new column called <col-name> defined by the algebraic expression <expr>. Expression syntax is described in Section 4. By default the new row appears after the last row of the table, but you can position it using either the -after or -before flags. In either case, a <col-id> is either the column's name (if it is syntactically a Java identifier), or its number (the first column is 1), or its $ID ($1 is the first column).

-keepcols <colid-list>
Output table consists of only those columns named in <colid-list>, in that order. <colid-list> is space-separated. col-id is either the column's name (if it is syntactically a Java identifier) or its number (the first column is 1) or its $ID ($1 is the first column).

-delcols <colid-list>
Delete named columns. <colid-list> is a space-separated list of identifiers which are either a column's name (if it is syntactically a Java identifier) or its number (the first column is 1) or its $ID ($1 is the first column).

-explode
Turns any column which is an N-element array into N scalar columns. Only works if the array size is fixed.

-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 flag near the start of the filters might help.

-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 won't 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.

Specifying `-verbose` has the effect of inserting a `-progress` flag at the start of the pipeline, so you can see how much progress has been made through the initial input table. By putting a `-progress` at different points in pipeline you can monitor how far different stages of the processing have progressed. If you insert more than one `-progress` however, output to the terminal is going to get quite messy.

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

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

If no filter specifiers are given, the input table will be sent directly to its destination without any modifications.

A.2.1.3 Mode Specifier

The mode specifier determines what happens to the processed table when it reaches the output end of the pipeline. Only one of the following should be specified:

-write [-ofmt <out-format>] [-o <out-table>]
A new table is written. If the `-o` flag is specified, output is `<out-table>`; otherwise, or if `<out-table>` has the special value "`-`", it's streamed to standard output. The output format is named using the `-ofmt` flag (see Section 3.2); if not supplied, an attempt is made to guess the format from the destination name. If neither `-o` nor `-ofmt` is specified, it's written in formatted text format to standard output (equivalent to `-o - -ofmt text`).

If you give an unknown output format (e.g. `-ofmt help`) then a list of all known formats will be printed.

-tosql <jdbc-url> [-user <username>] [-password <password>]
A new table is written to an SQL database. You need the appropriate JDBC drivers and `-Djdbc.drivers` set as usual (see Section 2.6). You can specify your SQL connection username and password or not - you will be prompted on the terminal if they are required.

-stats
Calculates and displays statistics (mean, standard deviation, minimum, maximum and number of good columns) for each column in the input table.

-meta
Prints to standard output the table metadata: parameters and column names.

-count
Counts the number of rows and columns and writes the answers to standard output.

-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 future versions (Axis upgrade).

If no mode specifier is given, `-write` is assumed. This means that with no mode specifier, the processed table is written to standard output in `text` format.

A.2.1.4 Other Flags

The following generic flags can also be issued:

`-h[elp]`
Prints a usage message and exits.

`-v[erbose]`
May cause more information about progress to be written as the command runs.

`-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 getting `OutOfMemoryErrors`. This flag is in most cases equivalent to specifying the system property `-Dstartable.storage=disk`.

`-debug`
Causes any error messages, which are usually made brief, to be accompanied by a stack trace. If you are reporting a bug (or debugging the code yourself), then you should use this flag to get the most information about what has gone wrong.

A.2.2 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 `tpipe` script installed and are using a Unix-like shell; see Section 2 for an explanation of how to invoke the command if you just have the Java classes.

The examples are arranged with one step (input, filter or destination) per line, to make it easier to see what's going on.

```
tpipe cat.fits
```

Writes a table to standard output in human-readable form. Since no mode specifier is given, `-write` is assumed, and output is to standard output in `text` format.

```
tpipe -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.

```
tpipe -ifmt csv xxx.csv \
      -keepcols "index ra dec" \
      -write -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.

```
cat tab1.vot | tpipe - -addcol IV_SUM "(IMAG+VMAG)" \
                       -addcol IV_DIFF "(IMAG-VMAG)" \
                       -delcols "IMAG VMAG" \
                       -write -ofmt votable -o - \
```

```
            > tab2.vot
```

Replaces two columns by their sum and difference in a VOTable. 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 column representing the difference, then delete the original columns.

```
tpipe 2dfgrs_ngp.fits \
      -disk \
      -keepcols "SEQNUM AREA ECCENT" \
      -sort -down AREA \
      -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.

```
tpipe http://archive.org/data/survey.vot -meta
```

Outputs column and table metadata about a table. In this case the table is a VOTable at the end of a URL.

```
tpipe survey.fits -select "skyDistance(hmsToRadians(RA),dmsToRadians(DEC), \
                                       0.6457,-0.1190) < 5 * ARC_MINUTE" \
                  -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 (RA=0.6457 radians, DEC=-0.1190 radians). 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. 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. The functions and constants used here are described in detail in Section 4.4.4.

```
tpipe -ifmt ascii survey.txt \
      -select "OBJTYPE == 3 && Z > 0.15" \
      -keepcols "IMAG JMAG KMAG" \
      -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.

```
tpipe -classpath lib/drivers/mysql-connector-java.jar \
      -Djdbc.drivers=com.mysql.jdbc.Driver \
      x.fits \
      -explode \
      -tosql jdbc:mysql://localhost/ASTRO1#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 TABLEX in the MySQL database called

ASTRO1 on a MySQL server on the local host, using the priveliges of MySQL user `mbt`. If a password is required, it will be prompted for on the terminal (the `-password` flag could be used to specify it instead). Any existing table in ASTRO1 with the name TABLEX is overwritten. The only processing done here is by the `-explode` flag, which takes any columns which have fixed-size array values and replaces them in the output with multiple scalar columns.

`tpipe USNOB.FITS -every 1000000 -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.3 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 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 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.3.1 Usage

The basic usage of `votcopy` is

```
votcopy [<flags>] [<in-file> [<out-file>]]
```

If you don't have the Unix scripts installed, invoke it as described in Section 2 using the classname `uk.ac.starlink.ttools.VotCopy`.

If `<out-file>` is omitted the result is written to standard output, and if `<in-file>` is also omitted the document to be copied is read from standard input. `<in-file>` may be a filename or URL, and may represent a VOTable compressed using one of the supported compression formats (gzip, Unix compress and bzip2).

The flags, which may be given in any order, are as follows:

`-f[ormat] tabledata|binary|fits|none`
Determines the encoding format of the table data in the output document. If `none` is selected, then the tables will be data-less (contain no `DATA` element), leaving only the document structure. Data-less tables are legal VOTable elements.

`-href`

In the case of BINARY or FITS encoding, this determines whether the STREAM elements will contain their data inline or externally. If -href is not specified, the output document will be self-contained, with STREAM data inline as base64-encoded characters. If -href is specified, then for each table in the document the binary data will be written to a separate file and referenced by a 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.

-base <name>
Determines the name of external output files written when the -href flag is specified. Normally these are given names based on the name of the output file. But if -base <name> is given, then these will given a name based on <name>. The -base flag is compulsory if -href is given and no output file is specified (output is to standard out), since in this case there is no default base name to use.

-cache
This flag causes any tables being copied to be read into a cache prior to being written out. Usually this doesn't provide any benefit, but under some circumstances it is necessary when the output is in FITS format. If you attempt to run votcopy without the -cache flag when it is required, an error message will tell you so.

-disk
When caching table data, uses a temporary disk file for storage rather than memory. Only has any effect if -cache is specified, and only required for large tables. Equivalent to setting the system property -Dstartable.storage=disk.

-encode <xml-encoding>
Selects the Unicode encoding used for the output XML. The available options and default are dependent on your JVM, but the default is probably UTF-8 and others available probably include UTF-16 etc.

-debug
Causes any error messages, which are usually made brief, to be accompanied by a stack trace. If you are reporting a bug (or debugging the code yourself), then you should use this flag to get the most information about what has gone wrong.

-h[elp]
Prints a usage message and exits.

A.3.2 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

```
votcopy -f 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 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

```
votcopy -f 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` flag is used. We will output in FITS format this time. Executing:

```
votcopy -f fits -href cat.vot fcat.vot
```

results in the error message:

```
Can't stream, table requires multiple reads for metadata
Try -cache option
```

- for technical reasons (FITS output requires the input tables to to be read in two passes to assess the number of rows) this can't be done in a single stream which is how `votcopy` usually works. So we follow the offered advice and use the `-cache` flag:

```
votcopy -f fits -href cat.vot fcat.vot -cache
```

which 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.4 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. If a non-flag argument is given it is assumed to be 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 typically made in authoring VOTables.

A.4.1 Usage

The basic usage of `votlint` is

```
votlint [<flags>] [<in-file>]
```

If you don't have the Unix scripts installed, invoke it as described in Section 2 using the classname `uk.ac.starlink.ttools.VotLint`.

If `<in-file>` is omitted then the document to be checked is read from standard input. `<in-file>` may be a filename or URL, and may represent a VOTable compressed using one of the supported compression formats (gzip, Unix compress and bzip2).

The flags, which may be given in any order, are as follows:

`-novalid`
Prevents validation against the VOTable DTD. Normally, as well as `votlint`'s own checks on the submitted document, it is validated against an appropriate version of the VOTable DTD which picks up such things as the existence 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. Specifying the `-novalid` flag prevents this validation step so that only `votlint`'s checks are performed. In this case a few, but by no means all violations of the VOTable standard concerning document structure will be picked up.

`-version <vers>`
Selects the VOTable version which the input table is supposed to exemplify. Currently `<vers>` can be `1.0` or `1.1`. The version may be noted within the document using the `version` attribute of the document's `VOTABLE` element; if it is and it conflicts with the version specified using this flag, a warning is issued.

`-h[elp]`
Prints a usage message and exits.

`-debug`
Causes any error messages, which are usually made brief, to be accompanied by a stack trace. If you are reporting a bug (or debugging the code yourself), then you should use this flag to get the most information about what has gone wrong.

A.4.2 Items Checked

Votlint checks that the XML input is well-formed, and, unless the `-novalid` flag 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 with `FIELD` declared `datatype`/`arraysize` attributes
- BINARY data streams which don't match declared `FIELD` metadata
- `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` 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` eleements
- 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.4.3 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 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.