TSTool Syntax Guide

Version 10.13.00, 2016-10-23

TSTool commands use a number of syntax (notation) conventions that have been implemented over time in response to functionality requirements. This appendix provides a summary of the syntax as a guide for users and future software development. Syntax standards listed here should be used where possible to ensure consistency in software features.

Where appropriate, notation has been selected based on other efforts. For example, date/time formatting is patterned after the C language `strftime()` function, which has been available for over 30 years. In cases where notation is specific to TSTool, an attempt has been made to consider common notation standards that can be adapted for TSTool. In cases where one or more existing standards are in place, the most common or relevant standard for TSTool has been selected, with an option to implement additional standards in the future.

Although standard notation is utilized into the software design, support for notation in commands may be incomplete because some commands use older code. For example, the ability to use properties to specify command parameters is implemented only for commands that have specifically required such functionality. Future software enhancements will continue to update code to universally provide standard features.

The following sections are ordered roughly in the order that topics are likely to be encountered, with headings grouped according to major TSTool design elements.

Commands – Basic Syntax

The syntax for commands adheres to the following syntax:

```
CommandName(Parameter1=Value1,Parameter2=Value2)
CommandName(Parameter1="Value1",Parameter2="Value2")
```

The `CommandName` matches a command from the TSTool ***Commands*** menu and as documented in the **Command Reference** documentation, which describes command parameters. Any parameter value can be surrounded by double quotes to protect whitespace and other characters (such as characters used in the command itself including equal sign, comma, and parenthesis). However, double quotes typically are used only for parameter values that are text, dates, filenames, etc., and not simple data such as numbers.

Commands currently cannot be indented, although this may be enabled in the future.

Command names and parameters generally are case-insensitive. However, "camel" notation (mixed upper and lower-case letters) is used to improve readability. In some cases this results in an acronym being converted from uppercase to missed-case (e.g., "USGS" becomes "Usgs").

Commands – Referring to Parameters

In some cases it is necessary to set one command parameter using the value of another command parameter. This capability has been implemented for a small number of commands, for example `NewStatisticEnsemble()`. To reference a command parameter in another parameter, use the notation:

```
CommandName(Parameter1=Value1,Parameter2="${C:Parameter1}…etc")
```

This notation uses `C:` to provide a "command scope", similar to how the `TS:` notation provides a scope for time series properties (discussed below).

Need to change the above to `${commandParameter:Parameter1}` to be more explicit and verbose to allow for more flexibility.

Commands – Comments

Command files use comments to disable commands without deleting them. A `#` character at the start of a line indicates a one-line comment. A group of lines that start with `/*` and end with `*/` indicate a block of comments and all intervening commands will be ignored in processing.

Commands – Time Series Identifiers

Time series identifiers (TDIDs) uniquely identify time series and are discussed in detail in the **Introduction** chapter. TSID commands, which match the syntax discussed below, are created when using the data browsing features of the TSTool main interface, are specified by some commands, and can be edited manually if the user edits a command file with a text editor. These commands are essentially "read" commands that use default parameters (e.g., the global input period and do not assign an alias).

There are two main forms of TSIDs:

```
Location.DataSource.DataType.Interval[.Scenario]~DataStore[~FileName]

Location.DataSource.DataType.Interval[.Scenario]
```

The first form of the TSID is a unique identifier for a time series, similar to a Universal Resource Indicator (URI) for a web page, and allows software to locate the data for reading. The datastore (or "input type" and corresponding filename) allow the software to find the source of the data.

The second form of the TSID is a unique identifier for a time series within TSTool and is used after reading the data. In cases where more than one time series will have the same TSID after reading, an alias can be assigned (see the **Introduction** chapter and the **Time Series – Properties** section below).

TSIDs may be more complex if, for example, the data type requires the use of multiple parts for uniqueness. In this case, a dash may be used (e.g., `Streamflow-Max`). The datastore appendices describe how time series properties from the original source are mapped into TSID notation.

Processor – Properties

TSTool commands are processed, and data managed, by a time series "processor". The processor interacts with all commands and is controlled with properties that initially have internal defaults (e.g., the default is to read all available data rather than a specified input period). Properties that control the processor are set with specific commands (e.g., `SetInputPeriod()`) and user-supplied properties can be set with the `SetProperty()` command (e.g., it is common to manage file locations and dates used in processing). The `ReadPropertiesFromFile()` and `WritePropertiesToFile()` commands can be used to save and manage properties outside of TSTool.

Processor properties can be used to specify parameters for commands using the following notation:

```
${PropertyName}
```

For example, some commands that operate on files allow the property `${WorkingDir}` to be used for the current working directory. Refer to command documentation to determine if properties are supported. Additional support is being phased in as resources allow and to satisfy requirements.

Properties internally have a specific data type. For example the input start and end use a "DateTime" object type supported by TSTool. All properties will convert to strings, for example when saved to a properties file. Some care may need to be taken to use properties of an appropriate type but a general rule is that properties used in file names or similar can simply be handled as strings.

Time Series – Properties

Time series properties are specific to individual time series. Some internal properties are handled as specific data values (e.g., data units are a string associated with a time series) whereas user-assigned properties are assigned to the time series as a list (see the `SetTimeSeriesProperty()` command). Time series properties are used by some commands to control the command functionality and output. For example, many commands that create time series allow the alias to be assigned using time series properties. The following notation is used when dealing with time series properties:

- `%` formatting – Many commands that create time series allow the `Alias` or other parameters to be assigned using `%` formatters. For example, `Alias="%L"` indicates that the time series alias should be assigned to the location part of the time series identifier, which for a read command is controlled by the rules of the command. Format specifiers are provided for fundamental time series data properties that are required for each time series (units, location, data type, etc.).
- `TS:Property` reference – Some command parameters need to specify a time series property by reference but the above formatting notation is inappropriate. In this case, the following design is being phased in (under development):
    - `TS:PropertyName`
    - `${TS:PropertyName}`

  The latter notation allows a time series property to be specified using a notation similar to processor properties, but the `TS:` prefix differentiates the property from the more generic processor notation.

Note that using time series properties in commands in some cases must be limited because TSTool uses a "discovery mode" to partially read/create time series so that they can be listed in "downstream" commands. Too much reliance on internal time series data might require reading more time series data, which can greatly decrease software performance in discovery mode.

Time Series – Data Flags

Time series data values (measurements, observations, etc.) are managed internally as lists of date/time, value, and flag data. A data flag is a string that is assigned a value based on one of the following cases:

- missing data value with a flag
- non-missing data value and no data flag
- non-missing data value with a flag

Data flags are useful for indicating the quality of a data value (e.g., `E` might indicate estimated) and for tracking how specific data values are manipulated (e.g., append to the data flag as specific actions are taken). TSTool generally does not implement a standard for data flags because flags used in input data may vary. However, some commands allow setting flags based on simple rules. For example fill commands generally have a `FillFlag` parameter to set the data flag for filled values. The following table lists notation that is used to provide flexibility in setting data flags. The first notation option is used by most commands and the other options are being phased in (refer to command documentation to confirm available data flag functionality).

Command Parameter Notation Used When Setting Data Flag

Notation	Description
`x`	Set the data flag to `x` regardless of whether it has already been set.
`+x`	If the flag has not been set, set to `x`. If the flag has been set, append `x`. This notation is useful when there are no concerns about the order of characters in multi-character flags.
`+,x`	If the flag has not been set, set to `x`. If the flag has been set, append `,x`. This notation is useful when flags are set for each step in a process.
`Auto`	Some commands allow `Auto` or another string as the flag. In this case, the command will decide the flag value that is assigned, based on some condition. For example, the flag may be assigned based on which time series was used to fill the value.

Data points in graphs can be labeled in various ways to facilitate interpretation of the data. For example, each data point can be labeled with the data value, flag, or other information. Similar to time series property formatting, the notation `%q` in graph data point labels indicates that the points should be labeled with the data flag.

Date/Time

Date/time notation is ubiquitous when dealing with time series, and includes use for the following:

- date/time associated with specific data values
- date/time pair that indicates data period or subset of the full data period
- date/time pair indicating a window within each year

In most cases TSTool will default to displaying date/time using the ISO 8601 specification, which is essentially `YYYY-MM-DD hh:mm:ss`. Not only does this implement a global standard, but it also ensures that date/times are formatted in a way that allows sequential sorting. The precision of formatted date/times is generally consistent with the time series data interval (e.g., monthly time series will have dates that are by default formatted as YYYY-MM).

It may be desirable or necessary to specify the format of date/times, for example to indicate the format for output or parsing. When this is necessary, the notation utilizes an optional format type prefix and the format itself, as follows:

- The default is to parse the date/time string by matching ISO or other common formats (this works most of the time). The default output format is the ISO format.
- `C:%m%d%y` – Indicates that a C-style format is being used, where the formats match the UNIX `strftime()` function syntax. See the `FormatDateTimeProperty()` command documentation.
- In the future support for Microsoft Excel or other notation may be added (e.g., `MM-YYYY`).

Regular Expression – Notation

Regular expressions are strings that indicate how to match patterns, for example to match file names or time series identifiers (see: http://en.wikipedia.org/wiki/Regular_expression). Many software tools and programming languages implement regular expressions to facilitate efficient data processing; however, the notation can be confusing, especially if not used on a regular basis. Within TSTool the following regular expression notations are used:

- "globbing" – This notation was popularized by UNIX and in simple terms relies on the * character to indicate "match zero or more characters". For example, it can be used to match a list of comma-separated-value files using the expression `*.csv`.
- Regular expression syntax – True regular expression syntax provides much more power than globbing notation, but also introduces complexity in notation. TSTool is written in Java and internally relies on Java's regular expression syntax.

In most cases, TSTool commands and configuration files use the simpler globbing notation because it is easier to use and explain. However, in some cases the more powerful regular expression syntax is needed. Where confusion may result, the command documentation clearly indicates the syntax that is supported, and commands may accept the notation `glob:xxxx` or `regex:xxxx` to indicate the type of regular expression that is being specified.

Tables

TSTool includes significant functionality to process tables. Internally, tables are managed using a flexible in-memory storage structure that consists of metadata and data space. Tables currently must consistent of columns containing the same data type (similar to database tables). A list of table rows (or records) is maintained, and each row contains a list of objects corresponding to the columns. Metadata is associated with the columns, such as column type, width and precision (for data types that support width and precision). Very large tables can cause TSTool to run out of memory. Consequently, care should be taken when processing tables, such as using the `FreeTable()` command when needed. Tables, when used with `For()` command, can result in compact command files, meaning that a few commands can perform significant processing work.

Table – Identification

Each table has an identifier, commonly called the "TableID" in TSTool, and a name. The TableID should be unique and is used to reference tables during processing.

Tables – Referring to Properties

Some table properties are built-in using metadata (such as column type, width, and precision) and are not typically needed by other processing commands. However, it is useful to property notation to retrieve

table values, in particular when used as command parameters to configure processing. The following properties are enabled for some commands that process tables.

Command Parameter Notation Used With Tables

Table Property Notation	Description
`${tablecolumnvalue:columnName}`	Indicates that the table column value for the specified column should be accessed/processed. For example, this notation is used in graph annotations to indicate that rows in the annotation table should be used to provide data values for the annotation.

Template – Syntax

Template files are used when processing is automated to iterate over one or more lists of input data. For example, the same 10 commands may be executed for each of 100 time series. TSTool uses the FreeMarker template library to process templates. See the `ExpandTemplateFile()` command documentation for an explanation of syntax.

Configuration File – TSTool Configuration File

The TSTool configuration file uses a simple notation to assign properties:

```
[Section]

Property = Value
```

The `[Section]` notation is internally used as a prefix on the property name (e.g., `Section.Property = Value`). Comments are lines that start with `#`. Property values can be surrounded by double quotes.

Configuration File – Datastore Properties

Datastore property files use the simple notation:

```
Property = Value
```

Comments are lines that start with `#`. Property values can be surrounded by double quotes. The specific property values are described in TSTool datastore appendices.

Configuration File – Time Series Product Files

Time series product configuration file uses a simple notation to assign properties:

```
[Section]

Property = Value
```

The `[Section]` notation is internally used as a prefix on the property name (e.g., `Section.Property = Value`). Comments are lines that start with `#`. Property values can be surrounded by double quotes. See also the **TSView Time Series Viewing Tools** appendix.

This column is intentionally blank.