Command Reference: ReadDelimitedFile()

Read time series from a delimited file

Version 10.00.01, 2011-05-15

The `ReadDelimitedFile()` command reads one or more time series from a column-oriented delimited file, where columns contain date/time and values. This command is useful for processing comma-separated-value (CSV) files exported from spreadsheets and mining data from the web (see also the `WebGet()` and `FTPGet()` commands). The command processes the following types of information:

1. Comments in the header (before data) and embedded in data records (e.g., because bad data values were commented out).
2. Column headers embedded in the file.
3. Data records, in column format, containing date/time strings, data values, and other information.
4. Metadata, such as station identifiers, data types, units, and interval.

The mapping of data in the file to data in the time series occurs first by assigning column names, using one of the following methods:

1. Read column names from a line in the file, suitable when the column headings are simple strings and agree closely with the contents of the data columns.
2. Assign column names with command parameters. The file being read may include metadata within column headings and data records; however, the information can be difficult to extract because of formatting. For example, column headings may include the data type as "Precipitation\n(in)" (where \n indicates a newline). Consequently, the command supports assigning column names via command parameters in order to ensure robust data handling.

In any case, rather than trying to automatically determine other metadata like data type and units from the column heading, the values can be assigned with the `DataType` and `Units` parameters. Additional functionality may be added in the future automate metadata discovery. Examples of use for the two cases are shown below.

The command syntax is as follows:

```
ReadDelimitedFile(Parameter=Value,…)
```

Command Parameters

Parameter	Description	Default
`InputFile`	The name of the delimited input file to read, surrounded by double quotes to protect whitespace and special characters. Global property values can be used with the syntax `${PropertyName}` (see also the `SetProperty()` command).	None – must be specified.
`Delimiter`	The delimiter character(s) that separate columns.	None – must be specified.
`TreatConsecutive DelimitersAsOne`	Indicate whether consecutive delimiter characters should be treated as a single delimiter, for example, when multiple spaces are used to line up columns.	`False` (columns are separated by a single delimiter character)
`Comment`	Character(s) that if found at the start of lines in the	#

	file, indicate that the line is a comment. The characters are interpreted individually (e.g., `#$` indicates that lines starting with # or $ will be treated as comments).	
`SkipRows`	Indicate absolute rows (1+) in the file to skip, using single numbers and ranges a-b, separated by commas. Rows are skipped prior to other processing.	No rows will be skipped.
`SkipRowsAfterComments`	Indicate the number of rows to skip after header comments. Use this parameter to skip column headers prior to the data lines. This parameter is typically not used if column names are read from the file.	No rows will be skipped.
`ColumnNames`	The user-specified names for columns in the file, used to ensure that column headings in files are properly interpreted. These names are used in other parameters to specify columns in the file. Separate column names with commas. Column names can be specified as literal strings or as `FC[start:stop]` to read columns from the file header (assumed to be the first row after leading comments), where `start` is `1+` and `stop` is blank to read all columns or a negative number to indicate the offset from the end column.	None – must be specified.
`DateTimeColumn`	The column matching a value in `ColumnNames`, which indicates the date/time column in the file.	None – must be specified.
`DateTimeFormat`	The format for date/time strings in the date/time column.	Under development – the format is automatically determined in most cases.
`DateColumn`	The column matching a string in `ColumnNames`, which indicates the date column in the file.	Under development.
`TimeColumn`	The column matching a string in `ColumnNames`, which indicates the time column in the file.	Under development.
`ValueColumn`	The column(s) matching a string in `ColumnNames`, which indicate the data value columns. Separate column names with commas. The `FC[start:stop]` notation discussed for `ColumnNames` can also be used.	None – must be specified.
`LocationID`	The location identifier(s) to assign to time series for each of the value columns (or specify one value to apply to all columns). The `FC[start:stop]` notation discussed for `ColumnNames` can also be used.	None – must be specified.
`Provider`	The data provider identifier to assign to time series for each of the value columns (or specify one value to apply to all columns).	No provider will be assigned.
`DataType`	The data type to assign to time series for each of the value columns (or specify one value to apply to	Use the value column names for the data

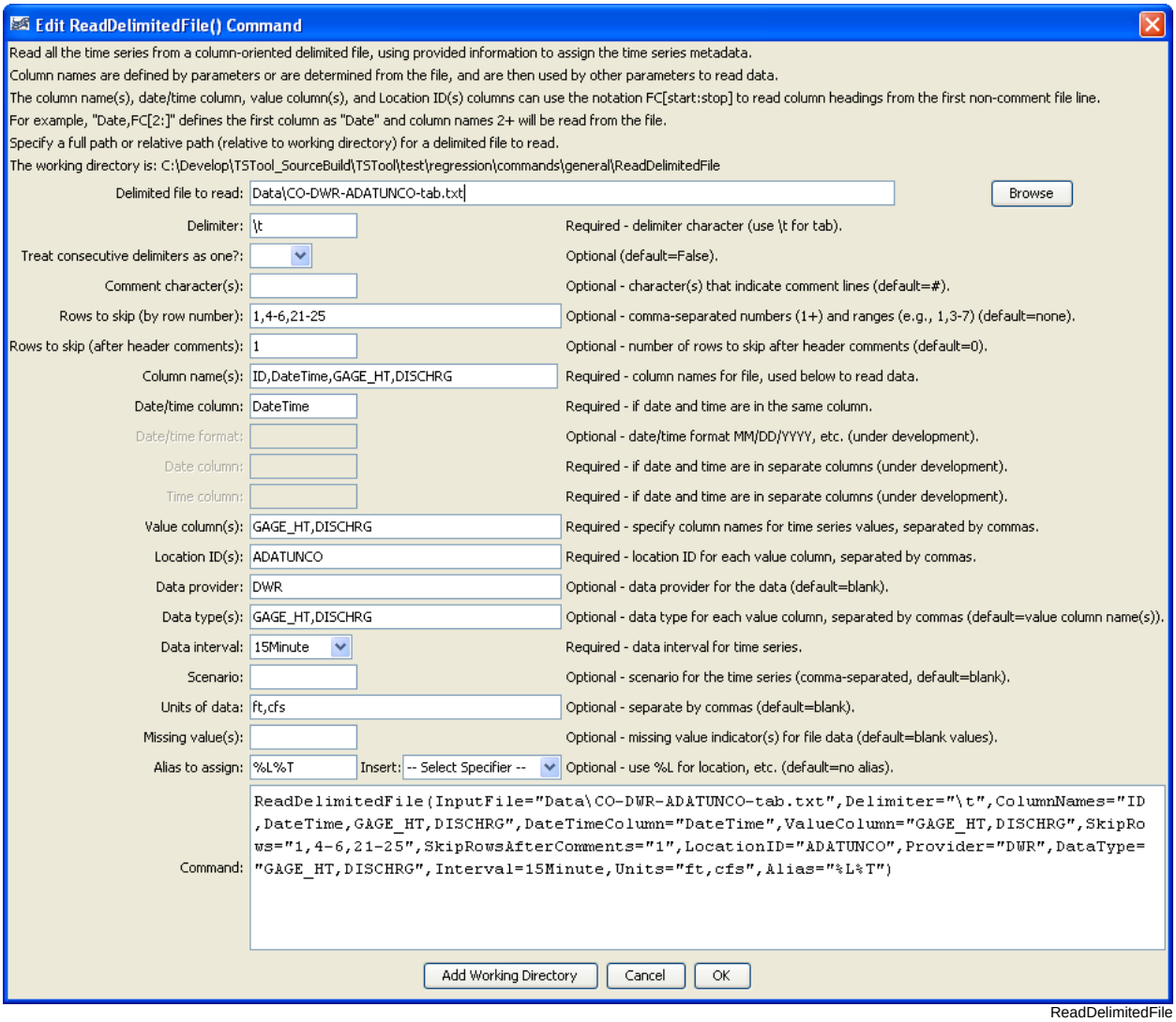
	all columns).	types.
`Interval`	The interval for the time series. Only one interval is recognized for all the time series in the file. Interval choices are provided when editing the command. If it is possible that the date/times are not evenly spaced, then use the `IRREGULAR` interval.	None – must be specified.
`Scenario`	The scenario to assign to time series for each of the value columns (or specify one value to apply to all columns).	No scenario will be assigned.
`Units`	The data units to assign to time series for each of the value columns (or specify one value to apply to all columns).	No units will be assigned.
`Missing`	Strings that indicate missing data in the file (e.g., "m").	Interpret empty column values as missing data.
`Alias`	The alias to assign to time series, as a literal string or using the special formatting characters listed by the command editor. The alias is a short identifier used by other commands to locate time series for processing.	No alias will be assigned.

Example of Column Names Assigned with Command Parameter

The following example for two time series (gate height and discharge) illustrates a format where column headings are complex enough to require assignment of column names using a command parameter:

```
#---------------------------- Provisional Data ------------------------------
#This system is maintained by the Colorado Division of Water Resources.
#Contact: Colorado Division of Water Resources (303) 866-3581
#
#All data presented on the Colorado Surface Water Conditions web site are
#provisional and subject to revision. Data users are cautioned to consider
#carefully the provisional nature of the information before using it for
#decisions that concern personal or public safety or the conduct of business
#that involves substantial monetary or operational consequences.
#
#Data is returned in TAB delimited format. Data miners may find help on automating
#queries and formatting parameters at http://www.dwr.state.co.us/help
#
#Gaging Station: ALVA B. ADAMS TUNNEL AT EAST PORTAL NEAR ESTES PARK (ADATUNCO)
#Retrieved: 3/30/2010 03:04
#-------------------------------------------------------------------------------
Station Date/Time       GAGE_HT (ft)    DISCHRG (cfs)
ADATUNCO        2006-10-01 00:00        2.34    225
ADATUNCO        2006-10-01 00:15        2.34    225
…etc…
```

The following dialog is used to edit the command and illustrates the syntax for the command. Note that the column headings are skipped because they are assigned with a command parameter.



ReadDelimitedFile

ReadDelimitedFile() Command Editor when Literally Specifying Column Names

The following example command file retrieves real-time time series data from the State of Colorado's website and reads the data:

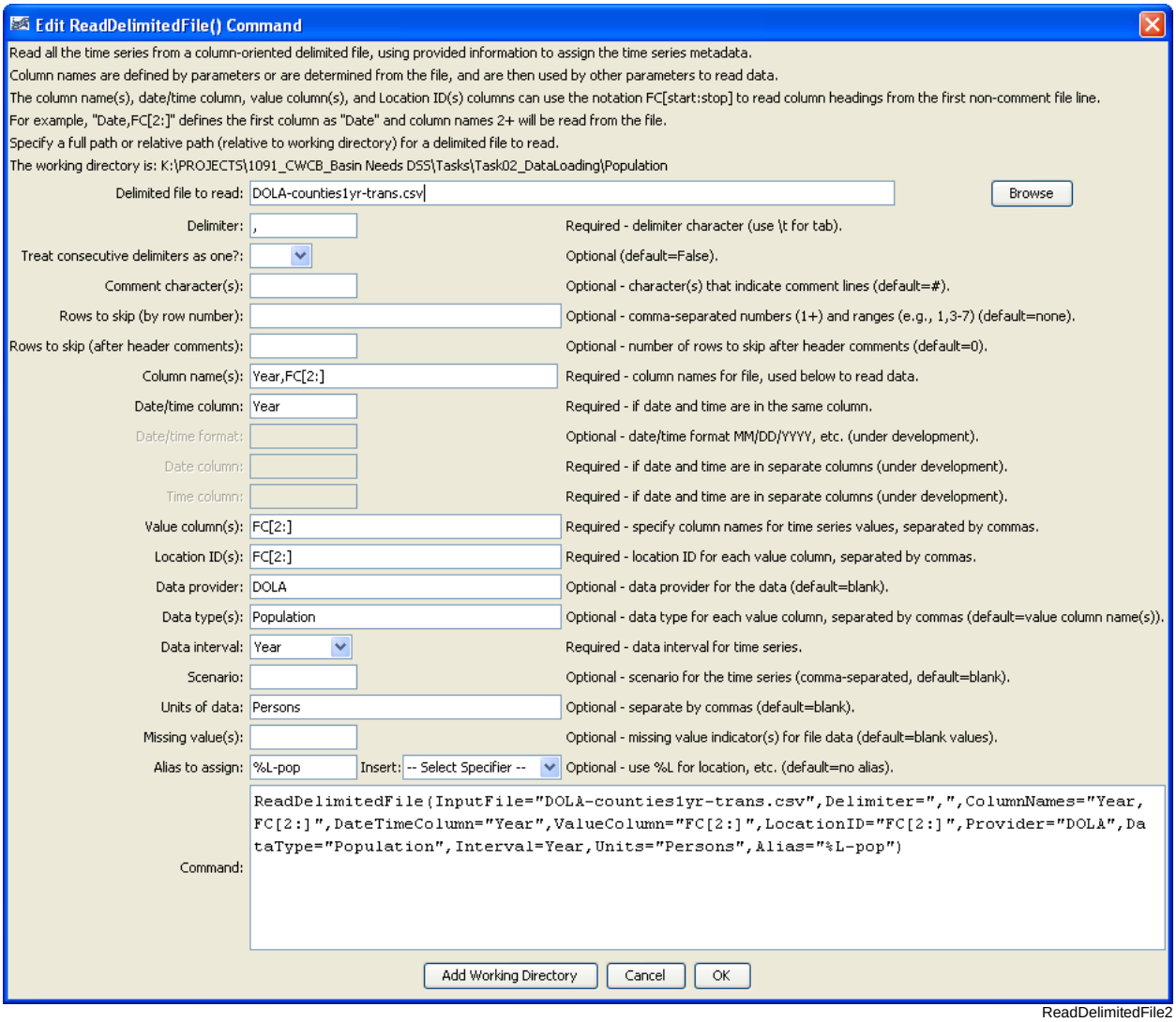
```
WebGet(URI="http://www.dwr.state.co.us/SurfaceWater/data/export_tabular.aspx?
  IDADATUNCO&MTYPEGAGE_HT,DISCHRG&INTERVAL1&START10/1/06&END10/6/06",
  LocalFile="Data\ Data\CO-DWR-ADATUNCO-tab.txt ")
ReadDelimitedFile(InputFile="Data\CO-DWR-ADATUNCO-tab.txt",
  Delimiter="\t",ColumnNames="ID,
  DateTime,GAGE_HT,DISCHRG",
  DateTimeColumn="DateTime",ValueColumn="GAGE_HT,DISCHRG",
  SkipRowsAfterComments="1",LocationID="ADATUNCO",
  Provider="DWR",DataType="GAGE_HT,DISCHRG",Interval=15Minute,
  Units="ft,cfs",Alias="%L%T")
```

Example of Column Names Read from the File

The following simple example of annual county population data illustrates a format that allows reading column names from the file. In this case, the rows and columns have been transposed from the original format to be compatible with this command and in the command example shown in the figure below the "County" heading is replaced with "Year" to more clearly indicate the contents.

```
County,COLORADO,Adams,Alamosa,Arapahoe,Archuleta,Baca,Bent,Boulder,Broomfield,Chaffee,…
2000,4338793,366660,15132,491134,10027,4514,5991,296018,0,16294,2229,9386,…
2001,4456408,360389,15314,502567,10532,4486,5911,282794,41529,16382,2195,9479,…
…etc..
```

The following dialog is used to edit the command and illustrates the syntax for the command.



ReadDelimitedFile2

ReadDelimitedFile() Command Editor when Reading Column Names from the File

The following example command file retrieves population forecast data from the State of Colorado's website, transposes the rows and columns using a Python script, and reads the time series data.

```
StartLog(LogFile="DOLA-county-pop.TSTool.log")
# This command file retrieves population data from the Colorado State Demographer
# website and processes the data into time series for use in analysis.
#
# First retrieve the data from the DOLA web site.
WebGet(URI="http://www.dola.state.co.us/dlg/demog/population/forecasts/counties1yr.csv",
  LocalFile="DOLA-counties1yr.csv")
#
# Transpose the rows/columns to match TSTool time series notation with dates in the
# first column.
SetProperty(PropertyName="ScriptDir",PropertyType=String,
RunPython(InputFile="${InstallDir}\python\table\transpose-csv.py",
  Arguments="\"${WorkingDir}\DOLA-counties1yr.csv\"
  \"${WorkingDir}\DOLA-counties1yr-trans.csv\"",Interpreter="Python")
#
# Read into time series from the delimited CSV file.
# Define column names dynamically based on the first non-comment line in the file
ReadDelimitedFile(InputFile="DOLA-counties1yr-trans.csv",Delimiter=",",
  ColumnNames="Year,FC[2:]",DateTimeColumn="Year",ValueColumn="FC[2:]",
  LocationID="FC[2:]",Provider="DOLA",DataType="Population",Interval=Year,Units="Persons",
  Alias="%L-pop")
```