Command Reference: ExpandTemplateFile()

Process a template file to create a fully-expanded file

Version 10.00.01, 2011-04-05

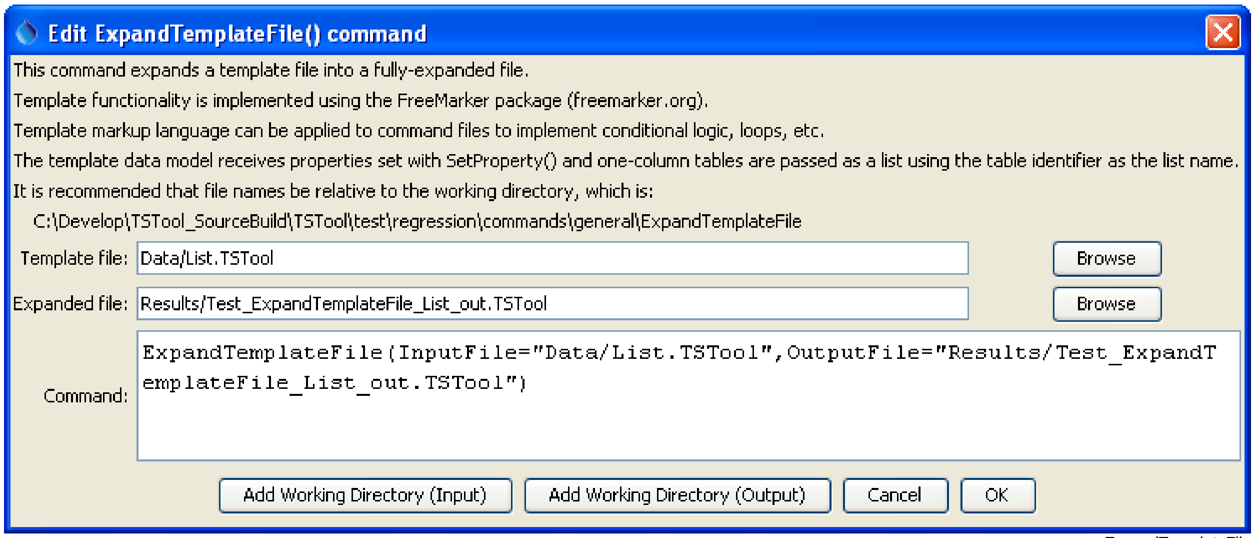
The `ExpandTemplateFile()` command processes a template file (typically a command file or time series product file but can be any text file) to create a fully-expanded file. Templates facilitate utilizing conditional logic, loops, and other dynamic processing functionality that is not provided in other TSTool commands. For example, a template can be used to repeat commands for multiple location identifiers. One advantage of using the template approach is that problems in the expanded file are clearly indicated, whereas a problem in logic that is represented as a loop might be difficult to diagnose.

The FreeMarker software (http://freemarker.org) is used to implement templates. Refer to the online documentation for information about the markup language used to create templates. Because TSTool checks commands for errors and does not itself understand FreeMarker syntax, template files must be edited with a text editor outside the normal TSTool editing. In the future, TSTool may be enhanced to provide template editing features. Examples below illustrate how to use common FreeMarker features.

The FreeMarker built-in `normalizeNewlines` user directive is automatically used to ensure that expanded files use newline characters appropriate for the operating system – this leads to temporary extra first and last lines in the template during processing, which need to be accounted for when interpreting FreeMarker warning messages. For example, a FreeMarker warning about line 21 would actually be line 20 in the template file. FreeMarker messages may be difficult to interpret. In general errors are because variable names are not spelled correctly or there is an error specifying FreeMarker syntax.

Properties set with the `SetProperty()` command are passed to the template processor. One-column tables are passed as lists, using the table identifier as the property name. For example, use the `CopyTable()` command to create a one-column table that can be used as a list for template expansion.

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



This command expands a template file into a fully-expanded file.
Template functionality is implemented using the FreeMarker package (freemarker.org).
Template markup language can be applied to command files to implement conditional logic, loops, etc.
The template data model receives properties set with SetProperty() and one-column tables are passed as a list using the table identifier as the list name.
It is recommended that file names be relative to the working directory, which is:
C:\Develop\TSTool_SourceBuild\TSTool\test\regression\commands\general\ExpandTemplateFile

Template file: Data/List.TSTool
Expanded file: Results/Test_ExpandTemplateFile_List_out.TSTool
Command:

```
ExpandTemplateFile(InputFile="Data/List.TSTool",OutputFile="Results/Test_ExpandTemplateFile_List_out.TSTool")
```

ExpandTemplateFile

ExpandTemplateFile() Command Editor

The command syntax is as follows:

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

Command Parameters

Parameter	Description	Default
`InputFile`	The name of the template file to process.	None – must be specified.
`OutputFile`	The name of the expanded output file.	None – must be specified.

Example Using Simple Variable Assignment

The following example illustrates a simple template command file and expanded result.

```
# Simple test to expand a text file using FreeMarker
<#assign message="Hello World">
${message}
```

```
# Simple test to expand a text file using FreeMarker
Hello World
```

Example Using Variable Assignment and Loop Using List

The following example illustrates a template command file repeat a command for a list of location identifiers. A block of multiple commands can be repeated, as appropriate. Long lines are indented for illustration but would exist on a single line without indentation in the template file. Note that the `loc_index` FreeMarker syntax allows the loop counter to be used.

```
# Simple template to illustrate how to repeat commands with a list of
# location identifiers
# Create a time series for each location
# The following ensures that the created template is read-only, so users
# modify the template instead:
#@readOnly
<#assign setStart = "2000-01-01">
<#assign setEnd = "2000-03-15">
<#assign units = "CFS">
<#assign locList = ["loc1", "loc2", "loc3", "loc4"]>
<#list locList as loc>
NewPatternTimeSeries(Alias="${loc}",NewTSID="${loc}..Streamflow.Day",
    SetStart="${setStart}",SetEnd="${setEnd}",Units="${units}",
    PatternValues="${loc_index + 1},0")
</#list>
```

The expanded command file is as follows:

```
# Simple template to illustrate how to repeat commands with a list of
# location identifiers
# Create a time series for each location
# The following ensures that the created template is read-only, so users
# modify the template instead:
#@readOnly
NewPatternTimeSeries(Alias="loc1",NewTSID="loc1..Streamflow.Day",
    SetStart="2000-01-01",SetEnd="2000-03-15",Units="CFS",PatternValues="1,0")
```

```
NewPatternTimeSeries(Alias="loc2",NewTSID="loc2..Streamflow.Day",
    SetStart="2000-01-01",SetEnd="2000-03-15",Units="CFS",PatternValues="2,0")
NewPatternTimeSeries(Alias="loc3",NewTSID="loc3..Streamflow.Day",
    SetStart="2000-01-01",SetEnd="2000-03-15",Units="CFS",PatternValues="3,0")
NewPatternTimeSeries(Alias="loc4",NewTSID="loc4..Streamflow.Day",
    SetStart="2000-01-01",SetEnd="2000-03-15",Units="CFS",PatternValues="4,0")
```

Example Using a Table for the List

The following example illustrates a template command file that reads the location list from a table. Note that the list must be a one-column table. If the original table has more than one column, read the original file and then use the `CopyTable()` command to create a new one-column table. A comma-separated-value (CSV) file is used for the list:

```
# Simple list to use during template expansion
"Location"
loc1
loc2
loc3
loc4
```

The template file is similar to the previous example; however, the list of locations is now provided via the table (no `<#assign>` element for the list) rather than having to hard-code in the template, which separates data from the processing logic:

```
# Simple template to illustrate how to repeat commands with a list of
# location identifiers
# Create a time series for each location
# The following ensures that the created template is read-only,
# so users modify the template instead:
# The list is provided by the processor as a one-column table with ID
# matching the list name
#@readOnly
<#assign setStart = "2000-01-01">
<#assign setEnd = "2000-03-15">
<#assign units = "CFS">
<#list locList as loc>
NewPatternTimeSeries(Alias="${loc}",NewTSID="${loc}..Streamflow.Day",
    SetStart="${setStart}",SetEnd="${setEnd}",Units="${units}",
    PatternValues="${loc_index + 1},0")
</#list>
```

The following command file reads the list of locations from the table and then expands the template file. Note that the `TableID` must match the list name in the `<#list…>` element in the template.

```
# Test expanding a FreeMarker template for a list of time series, using a
# one-column table as the list
# Read a one-column table that will be passed to the template as a list
ReadTableFromDelimitedFile(TableID="locList",InputFile="Data\loclist.csv")
ExpandTemplateFile(InputFile="Data\ProcessorTable.txt",
    OutputFile="Results/Test_ExpandTemplateFile_ProcessorTable_out.txt")
```

Example Defining Variables Using Time Series Processor Properties

A template file cannot be expanded to multiple files using the approach illustrated above. However, by placing an `ExpandTemplateFile()` command inside a template that uses a loop (a list), it is possible to expand a template to multiple files. One example of this is the automated generation of time series product files used with TSTool graphs, where a graph is created for each location being processed. Graphs that are formatted with time series product files allow a certain amount of dynamic information to be considered. However, because the products are organized into product (page), sub-product (graph on page), and data (time series in graph), dynamic data may not be configurable at the desired level. For example, time series legend text can be configured to automatically use the time series identifier; however, this information is not appropriate for the page title because the software cannot automatically decide which time series to use for the main title.

A solution is to use a template time series product (TSP) file that has a place-holder variable for the title and then expand the TSP file as the command file is processed. For troubleshooting and data management purposes, it is recommended that the TSP files are saved in a folder separate from final output. The same TSP filename could be reused; however, the files are small and saving distinct files allows them to be used individually if necessary.

The following TSP file illustrates a simple graph:

```
# Template product file for graphs

[Product]

ProductType = "Graph"
TotalWidth = "600"
TotalHeight = "400"
MainTitleString = "${loc} Streamflow"

[SubProduct 1]

GraphType = "Line"

[Data 1.1]

TSID = "${loc}..Streamflow.Day"
TSAlias = "${loc}"
```

The following template command file illustrates how a property is set to control expansion of a template time series product file (note that templates are stored in a *Data* folder and final output in a *Results* folder for testing purposes but data management will vary by application):

```
# Simple template to illustrate how to repeat commands
# with a list of location identifiers, and produce individual graphs.
# The list is provided by the processor as a one-column table
# with ID matching the list name
# The @readOnly comment ensures that the created template is read-only,
# so users modify the template instead.
#@readOnly
<#assign setStart = "2000-01-01">
<#assign setEnd = "2000-03-15">
<#assign units = "CFS">
<#list locList as loc>
```

```
# Set the loc variable for the processor so that it can pass
# to the ExpandTemplateFile command below
SetProperty(PropertyName="loc",PropertyType="String",PropertyValue="${loc}")
# Create the time series
NewPatternTimeSeries(Alias="${loc}",NewTSID="${loc}..Streamflow.Day",
    SetStart="${setStart}",SetEnd="${setEnd}",Units="${units}",
    PatternValues="${loc_index + 1},0")
# Expand the time series product file (graph) for the time series
ExpandTemplateFile(InputFile="..\Data\ProcessorTable_TSP_template.tsp",
    OutputFile="${loc}.tsp")
# Process the graph
ProcessTSProduct(TSProductFile="${loc}.tsp",OutputFile="${loc}.tsp)
</#list>
```

The following expanded command file creates time series and graphs. Although blocks of commands are repeated, the location identifier is different in each block of commands. Any errors in processing can be pinpointed when the expanded command file is loaded and generally are due to logic errors in the original template (errors repeated throughout the expanded command file) or data availability issues in specific time series (errors in one part of the expanded command file).

```
# Simple template to illustrate how to repeat commands
# with a list of location identifiers, and produce individual graphs.
# The list is provided by the processor as a one-column table
# with ID matching the list name
# The @readOnly comment ensures that the created template is read-only,
# so users modify the template instead.
#@readOnly
# Set the loc variable for the processor so that it can pass
# to the ExpandTemplateFile command below
SetProperty(PropertyName="loc",PropertyType="String",PropertyValue="loc1")
# Create the time series
NewPatternTimeSeries(Alias="loc1",NewTSID="loc1..Streamflow.Day",
    SetStart="2000-01-01",SetEnd="2000-03-
15",Units="CFS",PatternValues="1,0")
# Expand the time series product file (graph) for the time series
ExpandTemplateFile(InputFile="..\Data\ProcessorTable_TSP_template.tsp",
    OutputFile="loc1.tsp")
# Process the graph
ProcessTSProduct(TSProductFile="loc1.tsp",OutputFile="loc1.tsp)
# Set the loc variable for the processor so that it can pass
# to the ExpandTemplateFile command below
SetProperty(PropertyName="loc",PropertyType="String",PropertyValue="loc2")
# Create the time series
NewPatternTimeSeries(Alias="loc2",NewTSID="loc2..Streamflow.Day",
    SetStart="2000-01-01",SetEnd="2000-03-
15",Units="CFS",PatternValues="2,0")
# Expand the time series product file (graph) for the time series
ExpandTemplateFile(InputFile="..\Data\ProcessorTable_TSP_template.tsp",
    OutputFile="loc2.tsp")
# Process the graph
ProcessTSProduct(TSProductFile="loc2.tsp",OutputFile="loc2.tsp)
# Set the loc variable for the processor so that it can pass
# to the ExpandTemplateFile command below
SetProperty(PropertyName="loc",PropertyType="String",PropertyValue="loc3")
# Create the time series
NewPatternTimeSeries(Alias="loc3",NewTSID="loc3..Streamflow.Day",
    SetStart="2000-01-01",SetEnd="2000-03-
15",Units="CFS",PatternValues="3,0")
# Expand the time series product file (graph) for the time series
ExpandTemplateFile(InputFile="..\Data\ProcessorTable_TSP_template.tsp",
    OutputFile="loc3.tsp")
# Process the graph
ProcessTSProduct(TSProductFile="loc3.tsp",OutputFile="loc3.tsp)
# Set the loc variable for the processor so that it can pass
# to the ExpandTemplateFile command below
SetProperty(PropertyName="loc",PropertyType="String",PropertyValue="loc4")
# Create the time series
NewPatternTimeSeries(Alias="loc4",NewTSID="loc4..Streamflow.Day",
    SetStart="2000-01-01",SetEnd="2000-03-
15",Units="CFS",PatternValues="4,0")
# Expand the time series product file (graph) for the time series
ExpandTemplateFile(InputFile="..\Data\ProcessorTable_TSP_template.tsp",
    OutputFile="loc4.tsp")
# Process the graph
ProcessTSProduct(TSProductFile="loc4.tsp",OutputFile="loc4.tsp)
```

Also note that when the expanded command file is first opened in TSTool the `ProcessTSProduct()` commands will have a failure indicated. This is because the TSP file being used by the command has not yet been created (it is created as the commands are run). After the commands are run one time, the files will exist and subsequent loads of the command file will not show the warnings. This illustrates a potential issue, which is that templates can result in large numbers of files and the files should be cleared at appropriate times to ensure that old files are not used by mistake.

The FreeMarker language provides many features beyond those illustrated in these examples, including conditional (“if”) statements. However, the more complex the templates become, the more difficult they are to implement, troubleshoot, and maintain. Enhancements to TSTool may help with a solution that otherwise might require undesirable complexity.