Command Reference: ExpandTemplateFile()

Process a template file to create a fully-expanded file

Version 11.00.00, 2015-02-16

The `ExpandTemplateFile()` command processes a template specified with one of the following methods:

- template file, such as a command file, time series product file, or HTML, but can be any text file
- input text

and creates a fully-expanded:

- file
- and/or processor property.

Templates facilitate utilizing conditional logic, loops, and other dynamic processing functionality that is not provided directly by TSTool's limited `If()` and `For()` 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. Freemarker 2.3.15 is used prior to TSTool version 11.00.00, and Freemarker version 2.3.21 is used as of TSTool version 11.00.00. Support for other templating engines such as Apache Velocity can be added if needed. Refer to the online Freemarker documentation for information about the markup language used to create templates. Because TSTool checks commands for errors and does not itself understand FreeMarker syntax, templates must be edited with a text editor outside the normal TSTool command editors. Attempts to edit a template command file in TSTool may result in error indicators and some command editors may not allow changes to be saved, such as when template notation is used for a filename and the command expects a parent folder name to exist. TSTool may be enhanced in the future 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. Otherwise the results may have all lines merged together (not an issue for HTML used by web browsers but a big issue with TSTool). The `normalizeNewlines` directive 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 original template file. FreeMarker messages may be difficult to interpret. Common errors include variable names spelled incorrectly or incorrect FreeMarker syntax.

The following information is automatically passed from TSTool to the `ExpandTemplateFile()` command:

- Properties set with the `SetProperty()` command are passed to the template processor. Consequently, the property names can be referenced with `${Property}` in the template without using a FreeMarker assign command.
- One-column tables are passed as FreeMarker lists, using the table identifier (`TableID`) as the list property name. Null values in the table are passed as an empty string so that list have the

correct number of items for iteration. Use the `CopyTable()` command to create a one-column table that can be used as a list for template expansion. The `UseTables` command parameter can be used to turn off this transfer, for example in cases where an `ExpandTemplateFile()` command is being repeated many times, does not use the tables, and is slowed down by converting the tables to FreeMarker lists.

The following figures illustrate the dialog is used to edit the command. Separate tabs are used for input and output parameters.
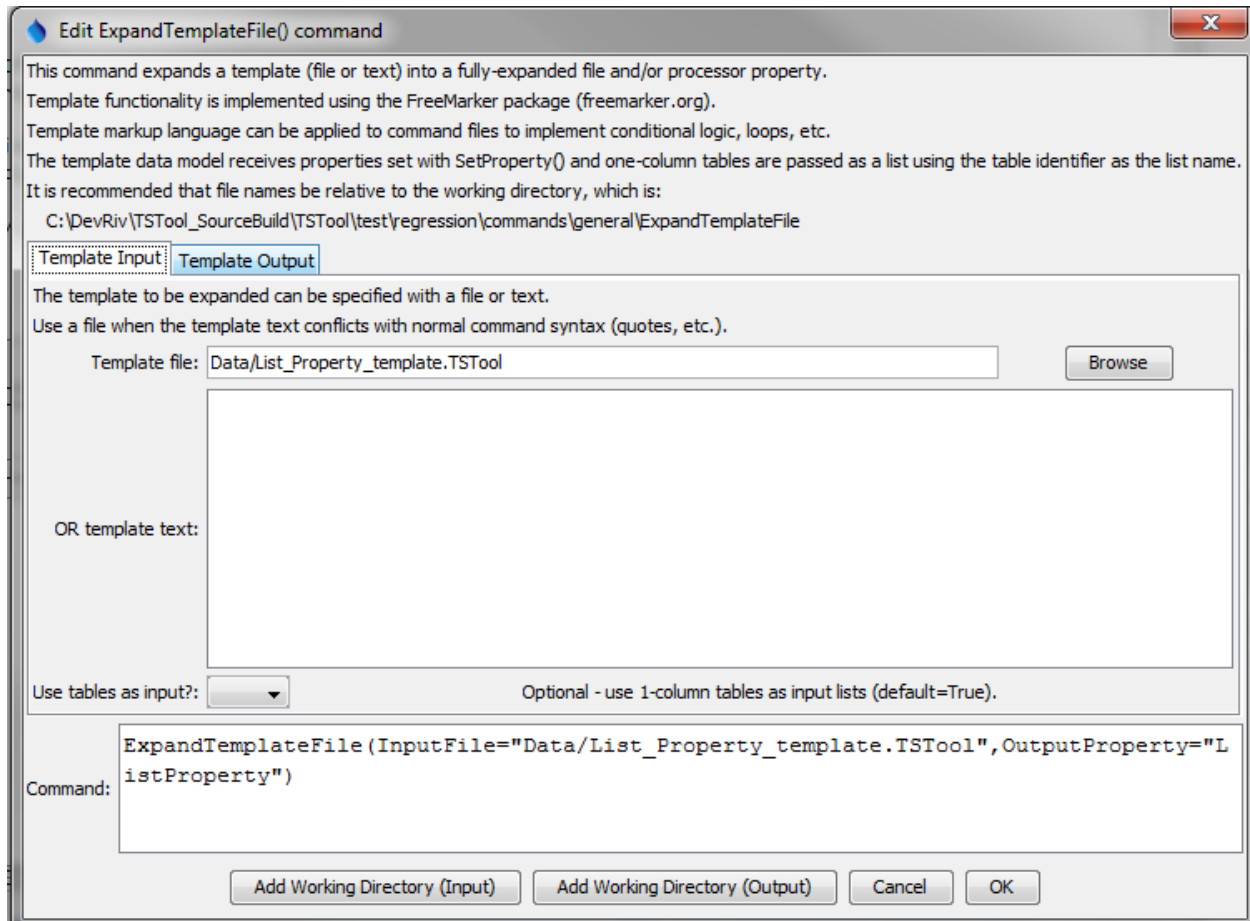
<div align="right">ExpandTemplateFile</div>

ExpandTemplateFile() Command Editor for Template Input Parameters
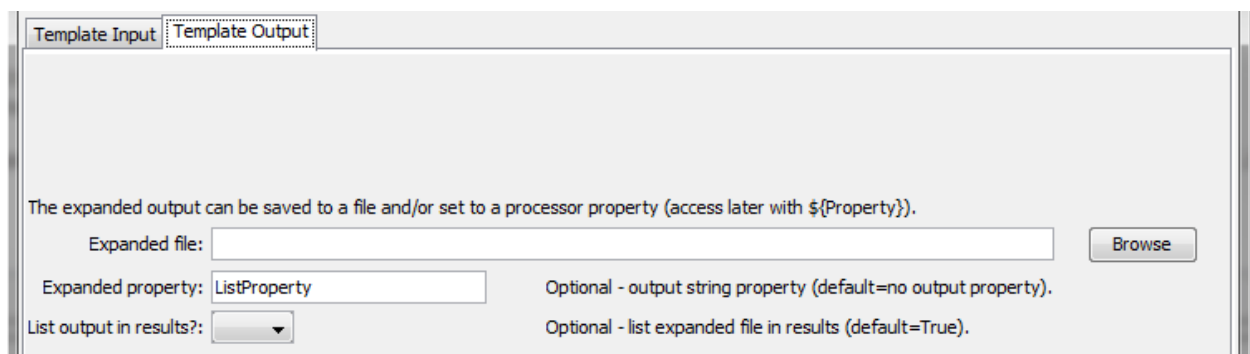
<div align="right">ExpandTemplateFile_Out</div>

ExpandTemplateFile() Command Editor for Template Output Parameters

The command syntax is as follows:

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

Command Parameters

Parameter	Description	Default
InputFile	The name of the template file to process. It is recommended that the filename include "template" and that a comment in the file include @readOnly, which will cause TSTool to warn users when saving the expanded result.	Specify if InputText is not specified.
InputText	The template text to expand. If special characters in the template text conflict with command syntax, specify InputFile instead.	Specify if InputFile is not specified.
OutputFile	The name of the expanded output file.	None – must be specified.
OutputProperty	The name of a property to receive the results of the template expansion. This is appropriate when templates are used to expand single-line text, for example.	No property value will be set.
UseTables	Indicate whether 1-column tables should be passed to the template expander. Doing so is a performance hit and should be avoided if tables are not used in the template.	True
ListInResults	Indicate whether the results of the expansion should be listed in the TSTool results area. This may be undesirable for "worker" files that users will normally not view.	True

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
#@readOnly
<#assign message="Hello World">
${message}
```

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

Example of Passing Time Series Processor Properties to Templates

TSTool uses the ${Property} notation to dynamically replace the string with the corresponding property value (as a string). FreeMarker uses the ${Variable} notation to dynamically replace the string with the corresponding variable (as a string). Because the same notation is used by both software components, care must be taken to ensure that values are properly interpreted.

TSTool automatically passes all TSTool properties to the `ExpandTemplateFile()` command.
Consequently, one of the main ways to avoid conflicts is to ensure that template command files do not use
any of the properties defined in TSTool. One way to check property names is to insert a
`WritePropertyToFile()` command at the appropriate line in a command file and review the list of
properties that are shown when editing the command.

To utilize TSTool processor properties in a template, do not use the FreeMarker assign command and
instead reference the property directly. The following TSTool command file illustrates how to define a
property that is used by the `ExpandTemplateFile()` command:

```
# Simple test to expand a text file using FreeMarker
SetProperty(PropertyName="HelloWorldProp",PropertyType=String,
    PropertyValue="Hello World")
ExpandTemplateFile(InputFile="Data\ProcessorStringProperty.txt",
    OutputFile="Results/Test_ExpandTemplateFile_HelloWorld_out.txt")
```

The corresponding template command file is as follows:

```
# Simple test to expand a text file using FreeMarker
${HelloWorldProp}
```

The variables also can be used in assignment, similar to the following:

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

Example of Protecting TSTool Properties in Template with a Literal FreeMarker String

It is possible to use TSTool property notation in a template and cause FreeMarker to ignore the property.
This will ensure that the property notation is present in the output, for TSTool to interpret at run-time.
The following example illustrates an input file that uses the FreeMarker `${r"…"}` raw literal string
notation:

```
# Simple test to expand a text file using FreeMarker
# and also escape text so that it passes through to the expanded file
# @readOnly
<#assign message="Hello World">
${r"SomeCommand($SomeProperty)"}
```

The corresponding output file is as follows:

```
# Simple test to expand a text file using FreeMarker
# and also escape text so that it passes through to the expanded file
# @readOnly
Hello World
SomeCommand($SomeProperty)
```

Example of Using a Comment in the Template, which is Omitted from Expanded Output

It often is desirable to have comments in the template file to explain the template, but not have the comments propagated to the expanded output. The following example illustrates an input file that uses the FreeMarker `$<#-- … -->` notation for comments:

```
# Simple test to expand a text file using FreeMarker
# There should be no comment in the expanded output below this line
<#-- This is a comment in the template -->
<#assign message="Hello World">
${message}
```

The corresponding output file is as follows:

```
# Simple test to expand a text file using FreeMarker
# There should be no comment in the expanded output below this line
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 One-Column Table for a List for Looping

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 Using a Multiple-Column Table to Loop Through Two Lists

The previous example illustrated how a one-column table can be used to loop over a list. However, often it is necessary to loop over one list and access the corresponding items from another list. The following example illustrates how a template command file can perform this task. Note that each list must be a one-column table in TSTool. If the original table has more than one column, use the CopyTable()

command to create as many one-column tables as are necessary. In this example, a comma-separated-value (CSV) file is used for the table:

```
# Simple list to use during template expansion
"Location","Value"
loc1,1.0
loc2,2.0
loc3,3.0
loc4,4.0
```

The template file to expand is similar to the one-column example; however, the second list is also used to provide information when expanding the commands (see bold text below):

```
# 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 location list is provided by the processor as a one-column table
# with ID matching the list name
# The value list is provided by the processor as a corresponding one-column
# table with ID "valueList"
#@readOnly
<#assign setStart = "2000-01-01">
<#assign setEnd = "2000-03-15">
<#assign units = "CFS">
# The loc_index is referenced to zero
<#list locList as loc>
<#assign value = valueList[loc_index]>
NewPatternTimeSeries(Alias="${loc}",NewTSID="${loc}..Streamflow.Day",
    SetStart="${setStart}",SetEnd="${setEnd}",Units="${units}",
    PatternValues="${value}")
NewPatternTimeSeries(Alias="${loc}",NewTSID="${loc}..Streamflow.Day",
    SetStart="${setStart}",SetEnd="${setEnd}",Units="${units}",
    PatternValues="${valueList[loc_index]}")
</#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 and corresponding arrays in the template.

```
# Test expanding a FreeMarker template for a list of time series, using a
# two-column table as the list
# Read a one-column table that will be passed to the template as a list
ReadTableFromDelimitedFile(TableID="locList2",
    InputFile="Data\2column-table.csv")
CopyTable(TableID="locList2",NewTableID="locList",
    IncludeColumns="Location")
CopyTable(TableID="locList2",NewTableID="valueList",
    IncludeColumns="Value")
ExpandTemplateFile(InputFile="Data\ProcessorTable.txt",
    OutputFile="Results/Test_ExpandTemplateFile_ProcessorTable_out.txt")
```

Example of Expanding a Template to a Processor Property

The following example illustrates how to expand a list into a SQL "in" clause, which is used to query specific matching records. A one-column table with identifier `locList` must have been created to supply the list of identifiers. The property set with the `OutputProperty` command parameter can then be used in the SQL statement for the `ReadTableFromDataStore()` command.

```
IN (<#list locList as loc><#if (loc_index > 0)>,</#if>'${loc}'</#list>)
```

Example of Using ExpandTemplateFile() in a Loop to Expand Multiple Files

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.