Soto State Machine API – Tag Library Guide

This guide is a work in progress and is provided as a quick reference. It groups the different tags by categories. Have fun.

General

stm:abort

This tag aborts a state execution flow (internally call `abort()` on the `Result` instance).

Example

```
<stm:state id="processOrder" success="displayOk">
  <stm:if test="user.shoppingCart.items.size == 0" scopes="params">
    <stm:stateRef id="displayNoItemsInCart" />
    </stm:abort>
  </stm:if>
  ...
</stm:state>
```

stm:assert

Test if a given value is present in a scope, or given set of scopes. This tag helps avoiding `NullPointerException`s. If the assertion performed by this tag fails, state execution aborts with an error.

Attributes

Name	*Value*	*Comments*	*Mandatory*
key	The name under which the “asserted” value is expected to be bound in the scope(s).	-	no
msg	The message to pass to the execution `Result` if no value could be found for the given key.	Will assign a default message if none is specified.	no
scopes	The comma- delimited list of scopes that should be searched for the given key/value.	Scopes are searched in the order in which they are specified in the list. Will search all scopes if no scope is specified.	no

Example

```
<stm:state id=”processOrder” success=”displayOk”>
  <stm:assert key=”user” scopes=”session” />
  <stm:if test=”user.shoppingCart.items.size == 0” scopes=”params”>
    <stm:stateRef id=”displayNoItemsInCart” />
    </stm:abort>
  </stm:if>
  ...
</stm:state>
```

stm:echo

Echoes a message to stdout.

Attributes

Name	*Value*	*Comments*	*Mandatory*
msg	The message to echo.	-	no

Example

```
<stm:state id=”processOrder” success=”displayOk”>
  <stm:echo msg=”processing order” />
  <stm:assert key=”user” scopes=”session” />
  <stm:if test=”user.shoppingCart.items.size == 0” scopes=”params”>
    <stm:stateRef id=”displayNoItemsInCart” />
    </stm:abort>
  </stm:if>
  ...
</stm:state>
```

stm:export

Exports an object from one scope to another (so can be considered an import also: importing an object from another...). Note that the object also remains in the scope of origin.

Attributes

Name	*Value*	*Comments*	*Mandatory*
from	The name of the scope from which the object should be copied.	-	yes
to	The name of the scope to which the object should be copied.	-	yes
key	The key/name under which the object to copy appears in the "from" scope.	-	yes
exportKey	The key/name under which the object should be copied in the destination scope.	If not specified, the value of the attribute "key" (see above) is used.	no

Example

```
<stm:state id="processOrder" success="displayOk">
  <stm:assert key="user" scopes="session" />
  <stm:if test="user.shoppingCart.items.size == 0" scopes="params">
    <stm:stateRef id="displayNoItemsInCart" />
    </stm:abort>
  </stm:if>
  <stm:export key="user" from="session" to="view" />
  ...
</stm:state>
```

stm:push

Push an object from a given scope to the execution stack. The object therefore becomes the "current" object on the stack, and can from then on be acquired from that stack using the `currentObject()` method on of the `Context` instance.

Attributes

Name	*Value*	*Comments*	*Mandatory*
from	The name of the scope from which the object should be copied.	-	yes
key	The key/name under which the object to copy appears in the “from” scope.	-	yes

Example

```
<stm:state id=”processOrder” success=”displayOk”>
  <stm:assert key=”user” scopes=”session” />
  <stm:if test=”user.shoppingCart.items.size == 0” scopes=”params”>
    <stm:stateRef id=”displayNoItemsInCart” />
    </stm:abort>
  </stm:if>
  <stm:push key=”user” from=”session” />
  ...
</stm:state>
```

stm:var

Creates a variable in a given scope.

Attributes

Name	*Value*	*Comments*	*Mandatory*
scope	The name of the scope to which the variable will be bound.	-	yes
key	The key/name of the variable.	-	yes
value	The value of the variable.		yes

Example

```
<stm:state id=”processOrder” success=”displayOk”>
  <stm:assert key=”user” scopes=”session” />
  <stm:if test=”user.shoppingCart.items.size == 0” scopes=”params”>
    <stm:var key=”Your shopping cart is empty” scope=”view” />
    <stm:stateRef id=”displayNoItemsInCart” />
    </stm:abort>
  </stm:if>
  ...
</stm:state>
```

Note that any object reference can conveniently be passed as a value (as supported by Soto's configuration format):

```
<stm:state id="processOrder" success="displayOk">
  <stm:assert key="user" scopes="session" />
  <stm:if test="user.shoppingCart.items.size == 0" scopes="params">
    <stm:var key="Message"
             scope="view">
      <value>
        <stm:serviceRef id="acme/myorg/services/internationalization" />
      </value>
    <stm:var>
    <stm:stateRef id="displayNoItemsInCart" />
    </stm:abort>
  </stm:if>
  ...
</stm:state>
```

Form Handling

---

stm:form

Initializes an object conforming to the JavaBeans spec with values from given scopes.

Attributes

Name	*Value*	*Comments*	*Mandatory*
class	The JavaBean class to use.	If no class is specified, the current object on the execution stack is initialized with the values specified by nested `param`. elements within this tag (see below). If a class is specified, an instance of it is created and pushed onto the context's stack (the class must have a public, no-args constructor).	no

Elements

param

This tag p

Attributes:

Name	*Value*	*Comments*	*Mandatory*
scopes	The comma-delimited list of scopes that should be searched for the given key/value.	If no scope is specified, all are searched.	no
from	The key/name of the object to lookup in the specified scopes	-	yes
to	The name of the JavaBean attribute to which the "from" object will be assigned.	If no value is specified, the value of the "from" attribute (above) will be used.	no

Example

```
<stm:state id="fillUserInfo" success="displayOk">
  <stm:assert key="firstName"   scopes="params" />
  <stm:assert key="lastName"    scopes="params" />
  <stm:assert key="phoneNumber" scopes="params" />
  <stm:assert key="userAccount" scopes="session" />
  <stm:push key="userAccount" from="session" />
  <stm:form>
    <param scopes="params" from="firstName" />
    <param scopes="params" from="lastName" />
    <param scopes="params" from="phoneNumber" />
  </stm:form>
  ...
</stm:state>
```

Code

---

stm:groovy

This tag executes code following the Groovy language. The code is passed as character data as part of the tag's XML content.

Elements

Name	*Value*	*Comments*	*Mandatory*
import	An "import" string (corresponds to Java's `import` statement).	-	no

Example

```
<stm:state id=”createUserAccount” success=”displayOk”>
  <stm:assert key=”firstName”   scopes=”params” />
  <stm:assert key=”lastName”    scopes=”params” />
  <stm:assert key=”phoneNumber” scopes=”params” />

  <stm:form class=”net.acme.myorg.database.UserAccount”>
    <param scopes=”params” from=”firstName” />
    <param scopes=”params” from=”lastName” />
    <param scopes=”params” from=”phoneNumber” />
  </stm:form>

  <stm:groovy>
    <import>net.acme.myorg.database.UserAccountManager</import>
    account = result.getContext().getCurrentObject();
    UserAccountManager.getInstance().save(account);
  </stm:groovy>
  ...
</stm:state>
```