SMC Tutorial

Kevin Twidle
22nd Nov 2005

There are brief notes to accompany the tutorial, they are not extensive documentation but should be enough to get the tutorial done.

First make sure that you are set up correctly.

Unpack SMC_Tutorial.zip, it will create a directory called SMC_Tutorial with the following files it:

README.txt
smc
trustcom.jar
MyManagedObject.java
tutorial.xml
trymymo.xml

Read and follow the README.txt file to ensure that everything is working for you.

Tutorial 1 – XML only.
For this tutorial you simply need to know XML syntax and how to use a text editor.

1. Create a policy that sets the alarm (see documentation) off if the monitor window goes over 65 and it resets the alarm when it goes under 25.

2. Using the TickManager, Policy(ies) and Event(s) create a system that turns the alarm on for 3 seconds every 10 seconds.

Tutorial 2 – Java and XML
For this tutorial you need a knowledge of Java, XML and how to use a text editor.

Create a ManagedObject that generates an event that causes a policy to turn the alarm on and off. You could use the tick manager to get a policy to give tick commands to your new managed object, or you could use the shell to give your managed object a command.

Events are sent with an ordered list of argument strings. The number of arguments must match the event definition:

```
Element[] arglist = new Element[2]();
argList[0] = new TextElement(icon);
argList[1] = new TextElement(value);
sendEvent("/Event/monitor", "Any String", argList);
```

Very brief SMC Java XML documentation

All XML elements are type com.twicom.qdparser.Element
Sub-classes are:
TaggedElement for tagged XML elements e.g. <compile type="bin"/>
TextElement for text strings e.g. some text
A TaggedElement may have an ordered list of Elements as its children.

`new TaggedElement("tagname");`	Create a new TaggedElement
`String name = taggedElement.name();`	Get the tag name of a TaggedElement
`int children = taggedElement.elements();`	Get the number of sub-elements
`String attribute = taggedElement.getAttribute("attname");`	Get an attribute's value
`TaggedElement.setAttribute("name", "value");`	Set an attribute and its value
`TextElement te = new TextElement("some text");`	Create a new TextElement
`taggedElement.add(element);`	Add a sub-element to a TaggedElement
`for (Object o : taggedElement) { Element e = (Element)o; }`	Go through the sub-elements
`String str = element.toString();`	Convert to XML string

e.g. to create:
<sample att1='att1value'>
    <sub1 sub1att="s1a"/>
    <sub2>sub2 text</sub2>
    Sample text <!-- --> second text
</sample>

```
import com.twicom.qdparser.*;
TaggedElement sample = new TaggedElement("sample");
sample.setAttribute("att1", "att1value");
TaggedElement sub = new TaggedElement("sub1");
sub.setAttribute("sub1att", "s1a");
sample.add(sub);
sub = new TaggedElement("sub2");
sub.add("sub2 text");  // shorthand for sub.add(new TextElement("sub2
text"));
sample.add(sub);
sample.add("Sample text");
sample.add("second text");
System.out.println(sample.toString());
```

TickManager

Import

org.trustcom.managedobject.TickManager

Create

<create>

Commands

Tick to set up a timer, can be used with create or later with a use statement.

```
<tick
        at=”datetime” | delay=”secs”
        [repeat=”secs” [count=”number”]]
        event=”pathname”
>
        event arg 1
        event arg 2
</tick>
```

Event args can include the substitured arguments:
Tick – the number of ticks so far generated by this tick command
Time – the current fate and time as a string
Secs – the time in seconds since 1st Jan 1971

Cancel to cancel all timers associated with this managed object
<cancel/>

Example

```
<!--# Create an instance of the ticker, assuming that we have set up /Event/tick -->
<use name="/Timer">
        <add name="ticker">
                <use name="/Template/tick">
                        <create>
                                <tick event="/Event/tick" delay="10" repeat="5">
                                        !tick;
                                </tick>
                        </create>
                </use>
        </add>
</use>
```

Policy

Create

```
<use name=”/Template/policy”>
        <create type=”obligation” event=”/Event/myevent” active=”true”>
                <arg name=”arg1”/>
                <arg name=”arg2”/>
                <condition>
                        <and>
                                <cond1>
                                <cond2>
                        </and>
                </condition>
                <action>
                        <use …/>
                        <use …/>
                </action>
        </create>
</use>
```

Condition builtins are and, or, not, gt, lt, ge, le, eq. And and Or currently only take two arguments.

Can use a managed object for a condition. The managed object in question must respond to a command by setting the condition result e.g. result.setCondition(true); If a managed object does not set a condition then it is assumed to be false.

Example

```
<use name="/Policy">
      <add name="alarmedPDP">
            <use name="/Template/policy">
                  <create type="obligation" event="/Event/repLT40"
active="true">
                        <arg name="name"/>
                        <arg name="value"/>
                        <action>
                              <use name="/PDP/PDD1">
                                    <alarmedpdp>
                                          !name; <!-- --> !value;
                                    </alarmedpdp>
                              </use>
                        </action>
                  </create>
            </use>
      </add>
</use>
```

Event Type

Events are created by managed objects or the shell. However, the specific event type must be created from the standard Event Template before it can be used. Once created, it can be used by name in the policy secifications and by managed objects that want to create that particular event.

The arguments are typically used in Policy actions using string substitution.
i.e. if myarg=”hello”then

```
The argument is !myarg;, it comes from the event.
```

produces

```
The argument is hello, it comes from the event.
```

Create

```
<create>
      <arg name=”argname1”/>
      …
      <arg name=”argnameN”/>
</create>
```

Example

```
<use name="/Event">
      <add name="svcadd">
            <use name="/Template/event">
                  <create>
                        <arg name="domain"/>
                        <arg name="name"/>
                  </create>
            </use>
      </add>
</use>
```

Shell

To create the above event in the shell with domain=”/people” and name=”fred”

```
$ event /Event/svcadd /people fred
```

Monitor

A Java Swing slider to allow visual input from the user to create events within the system

Import

```
<import name=”org.trustcom.managedobject.Monitor”/>
```

Create

```
<create
      [title="window title"]
      [icon="iconvalue"]
      [min="0" max="100"]
      [visible="true"]
/>
```

Commands

Can be used with <create> or <use>. Any of the above attributes can be used with <use>.

```
<threshold
      trip="GT"
      value="70"
      event="/Event/repGT70"
/>
<show/>
<hide/>
```

The event must take two arguments they will be (in order): iconvalue and currentvalue

Where trip refers to when the trip should occur

GT	when the marker goes greater than the value
GE	when the marker goes greater than or equal to the value from below
LT	when the marker goes less than the value
LE	when the marker goes less than or equal to the value from below
X	when the marker crosses the value
XE	when the marker crosses the value or becomes equal to it

Example

```
<use name="/Template/monitor">
      <create title="Reputation Monitor - Airline VO"
            icon="badreputation.jpg" min="0" max="100"
            visible="true"
      />
</use>
<use name="/svc/monitor">
      <threshold trip="GT" value="70" event="/Event/repGT70"/>
      <threshold trip="LT" value="40" event="/Event/repLT40"/>
      <threshold trip="LT" value="50" event="/Event/repLT50"/>
</use>
```

Bugs

Icon should be called something else and should be part of the threshold command!

AlarmDisplay

A Java Swing window with an animated alarmclock that can be turned on and off

Import

```
<import name=”org.trustcom.managedobject.AlarmDisplay”/>
```

Create

```
<create
      [title="window title"]
      [alarm="on"|”off”]
/>
```

Commands

Show and hide commands can be used with the create above. The AlarmDisplay is controlled by the alarm attribute.

```
<use name=”...” alarm=”on”|”off”/>
<show/>
<hide/>
```

Example

```
<use name=”/Template/alarmdisplay”>
      <create title=”My Alarm”/>
</use>

<use name="/example/alarmdisplay" alarm=”on”/>
```

Shell

Could use the shell to turn the alarm on and off

```
/example/alarmdisplay alarm=on
```

XMLBlaster

An XMLBlaster client that can publish and subscribe to messages

Import

```
<import name="org.trustcom.externalobject.XMLBlaster"/>
```

Create

```
<create/>
```

Commands

Can be used with <create> or <use>. Any of the above attributes can be used with <use>.

```
<connect [protocol="SOCKET"|"IOR"|"XMLRPC"]
      [uname="name" passwd="pass"]/>
```

Connect to xmlBlaster using protocol `socket` (default), `IOR` (CORBA) or `XMLRPC`. A username and password may be given, if one is required then both must be used. Additional attributes that can be used here include: `localhost`, `localport`, `remotehost` and `remoteport`. They should be self-explanatary.

```
<subscribe name="xmlblaster topic name"/>
```

Create a new subscription with topic name. See XMLBlasterSubscription create for more information and parameters.
A new Managed Object of the type XMLBlasterSubscription is returned. This must be saved in the domain hierarchy for future use. See example below.

```
<disconnect>
```

Disconnect from xmlBlaster, closing all current subscriptions.

Example

```
<use name="/Template/xmlblaster">
      <create/>
</use>

<!--Connect to xmlBlaster -->
<use name="/xmlblaster">
      <connect/>
</use>

<!—Create a subscription and call it /subscription/hello -->
<use name="/subscription">
      <add name="hello">
            <use name="/xmlblaster">
                  <subscribe name="pstrial" event="/Event/hello/>
            </use>
      </add>
</use>
```

XMLBlasterSubscriber

An XMLBlaster client that can send and receive notifications using a single xmlBlaster topic.

Import

This Managed Object come from the `subscribe` operation on XMLBlaster.

Create

This Managed Object come from the `subscribe` operation on XMLBlaster.

```
<subscribe name="xmlblaster topic name" [event="event type]"
      [type="txrx"||"tx"|"rx"] [initialupdate="false|true"]/>
```

Subscribe to an XMLBlaster topic. The type of subscription can be publish and subscribe (`txrx`), publish only (`tx`) or subscribe only (`rx`). If subscribing (`txrx` or `tx`) then the event type must be given. Default is `txrx`.

`Initialupdate` specifies whether an existing notification on the topic is sent when subscribing (`true`) or whether only new notifications should be sent (`false`). The default is `false`. With `txrx`, all messages sent will be received as well.

Commands

Subscribe, above, may be used again to subscribe to a different topic, the previous topic will be dropped.

```
<send>contents of message to be sent</send>
```

Sends contents as message to xmlBlaster on the current topic, the `send` tags are not sent. Spaces may not be preserved.
NB. XmlBlaster messages received as part of a subscription must be valid XML structures. E.g. if a message sent from an XMLBlasterSubscriber could be:

```
<send><mymessage>arg1 <!-- --> arg2</mymessage></send>
```

```
<erase/>
```

`Erase` tells xmlBlaster to remove any currently held messages with this topic.

```
<unsubscribe/>
```

`Unsubscribe` tells xmlBlaster to unsubscribe from this topic. `Subscribe` must be used before anything else may be done with this managed object.

Example

```
<!—Send a message to xmlBlaster -->
<use name="/subscription/hello">
      <send><mynotification>alarm</mynotification></send>
</use>

<!—Remove messages and unsubscribe -->
<use name="/subscription/hello">
      <erase/>
      <unsubscribe/>
</use>
```