Managing Trigger Conditions

This chapter contains the following sections:

Trigger Conditions

To create a trigger for a specific purpose, you must have a trigger condition that is correctly defined. If a trigger condition does not already exist, you have to implement it. Likewise, if the appropriate and necessary components of the condition are not yet defined, then you can implement them using the information provided here.

In the Create Trigger Wizard (found under **Policies** > **Orchestration** > **Triggers**, at the **Specify Conditions** step, you should have the options available to set up the new trigger condition.

A trigger is composed of two components:

- An implementation of **com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoredContextIf**.
- At least one implementation of **com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoredParameterIf**.

The **MonitoredContextIf** is supposed to describe the object that is to be monitored and supply a list of references to the object. When you use the **Edit Trigger** > **Specify Conditions** element of the Wizard, you should see controls and related options that allow you to select the object and the references to it. For example, the **MonitoredContextIf** might be used to monitor the "Dummy Device" objects and to return a list of all the Dummy Devices available.

The **MonitoredParamterIf** is used in the definition of a trigger condition as follows:

- It provides the specific parameter to be examined. For example, it could be a parameter representing the status of the particular Dummy Device (for example, ddTwo) as defined by the **MonitorContextIf**.
- It supplies the operations that can be applied to the parameter. Typical operations include, for example:,
  - less than
  - equal to
  - greater than

  (The appropriate operations depend on the implementation.)

- It supplies a list of values, each of which can be logically compared against the parameter to activate the trigger.

So, for example, a trigger condition such as "Dummy Device ddTwo Status is down" can be logically tested as a condition. If the monitored Status parameter renders the statement True, the trigger condition is met.

Adding Trigger Conditions

Before you begin

Refer to the Open Automation javadocs for details on the implementation of the interface.

Step 1 Implement a **MonitoredContextIf** and all the applicable **MonitoredParameterIf**s.

```
public class MonitorDummyDeviceStatusParam implements MonitoredParameterIf {
@Override
public String getParamLabel() {
//this is the label of this parameter shown in the ui
return "Dummy Device Status";
}
@Override
public String getParamName() {
//each parameter needs a unique string, it's a good idea to //prefix each
parameter
//with your module id, this way it basically guarantees //uniqueness
return "foo.dummy.device.status";
}
@Override
public FormLOVPair[] getSupportedOps() {
//this should return all the supported operations that can be //applied to
this parameter
FormLOVPair isOp = new FormLOVPair("is", "is");
FormLOVPair[] ops = { isOp };
return ops;
}
@Override
public int getValueConstraintType() {
return 0;
}
@Override
public FormLOVPair[] getValueLOVs() {
//this should return all the values you want to compare against //e.g.
threshold values
FormLOVPair valueUP = new FormLOVPair("Up", "up");
FormLOVPair valueDOWN = new FormLOVPair("Down", "down");
FormLOVPair valueUNKNOWN = new FormLOVPair("Unknown", "unknown");
FormLOVPair[] statuses = { valueDOWN, valueUNKNOWN, valueUP };
return statuses;
}
@Override
public int getApplicableContextType() {
//this parameter is binded to MonitorDummyDeviceType, so it needs //to return
the same
//value returned by MonitorDummyDeviceType.getContextType()
DynReportContext dummyContextOneType =
ReportContextRegistry.getInstance().getContextByName(FooConstants.DUMMY_CONTEX
T_ONE);
return dummyContextOneType.getType();
}
```

```
@Override
public String getApplicableCloudType() {
return null;
}
@Override
public int checkTrigger(StringBuffer messageBuf, int contextType,
String objects, String param, String op, String values) {
//you want to basically do if (objects.param op values) { //activate } else {
not activate }
//first step, you'd look up what objects is pointing to, usually objects
should be an identifier
//for some other object you actually want
//in this example, objects is either ddOne (dummy device) or ddTwo, for
simplicity's sake, we'll
//say ddOne is always up and ddTwo is always down
if (objects.equals("ddOne")) {
if (op.equals("is")) {
//ddOne is always up, so trigger only gets activated when "ddOne is up"
if (values.equals("up")) {
return RULE_CHECK_TRIGGER_ACTIVATED;
} else {
return RULE_CHECK_TRIGGER_NOT_ACTIVATED;
}
} else {
return RULE_CHECK_ERROR;
}
} else {
if (op.equals("is")) {
//ddTwo is always down, so trigger only gets activated when "ddTwo is not up"
if (values.equals("up")) {
return RULE_CHECK_TRIGGER_NOT_ACTIVATED;
} else {
return RULE_CHECK_TRIGGER_ACTIVATED;
}
} else {
return RULE_CHECK_ERROR;
}
}
}
}
public class MonitorDummyDeviceType implements MonitoredContextIf {
@Override
public int getContextType() {
//each monitored type is uniquely identified by an integer
//we usually use the report context type
DynReportContext dummyContextOneType =
ReportContextRegistry.getInstance().getContextByName(FooConstants.DUMMY_CONTEX
T_ONE);
return dummyContextOneType.getType();
}
@Override
public String getContextLabel() {
//this is the label shown in the ui
return "Dummy Device";
}
@Override
public FormLOVPair[] getPossibleLOVs(WizardSession session)
//this should return all the dummy devices that could potentially be monitored
//in this example i only have two dummy devices, usually the value should be
an identifier you can use
//to reference back to the actual object
FormLOVPair deviceOne = new FormLOVPair("ddOne", "ddOne");
FormLOVPair deviceTwo = new FormLOVPair("ddTwo", "ddTwo");
FormLOVPair[] dummyDevices = { deviceOne, deviceTwo };
```

```
return dummyDevices;
}
@Override
public String getContextValueDetail(String selectedContextValue) {
//this is additional info to display in the ui, i'm just returning a dummy
string
return "you picked " + selectedContextValue;
}
@Override
public String getCloudType(String selectedContextValue) {
// TODO Auto□generated method stub
return null;
}
}
```

Step 2 Register the trigger condition into the system.

`com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoringTriggerUtil` has a static method for this purpose.

```
// adding new monitoring trigger, note, these new trigger components
// utilize the dummy context one i've just registered
// you have to make sure to register contexts before you execute
// this code, otherwise it won't work
MonitoringTrigger monTrigger = new MonitoringTrigger(
new MonitorDummyDeviceType(),new MonitorDummyDeviceStatusParam());
MonitoringTriggerUtil.register(monTrigger);
menuProvider.registerWithProvider();
```

a) Group your **MonitoredContextIf** and its **MonitoredParameterIf**s together into a **com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoringTrigger**.

b) Register the monitoring trigger with the utility.