# Table of Contents

1. Introduction .................................................................................................................. 1  
2. Data Models .................................................................................................................. 1  
   2.1. Sales Input Event ........................................................................................................ 1  
   2.2. Sales Decision Event .................................................................................................. 1  
   2.3. Stock Control: Items ................................................................................................. 2  
   2.4. Personnel System: Assistants .................................................................................... 2  
   2.5. Locations: Branches .................................................................................................. 2  
3. Policy Step 1 .................................................................................................................... 2  
   3.1. Scenario ...................................................................................................................... 2  
   3.2. Create the an new empty Policy Model MyFirstPolicyModel ................................ 3  
   3.3. Create the input event SALE_INPUT and the output event SALE_AUTH ................ 4  
   3.4. Create a new Policy and add the “No Booze before 11:30” check ......................... 8  
   3.5. Test Policy Step 1 ....................................................................................................... 18  
   3.6. Policy 1 in CLI Editor ............................................................................................... 21  
4. Policy Step 2 .................................................................................................................... 25  
   4.1. Scenario ...................................................................................................................... 25  
   4.2. Extend the Policy with the new Scenario ................................................................. 25  
   4.3. Test Policy Step 2 ....................................................................................................... 31  
   4.4. Policy 2 in CLI Editor ............................................................................................... 34
1. Introduction

Consider a scenario where a supermarket chain called *HyperM* controls how it sells items in a policy-based manner. Each time an item is processed by *HyperM*'s point-of-sale (PoS) system an event is generated and published about that item of stock being sold. This event can then be used to update stock levels, etc..

*HyperM* want to extend this approach to allow some checks to be performed before the sale can be completed. This can be achieved by requesting a policy-controlled decision as each item is processed by for sale by each PoS system. The decision process is integrated with *HyperM*'s other IT systems that manage stock control, sourcing and purchasing, personnel systems, etc.

In this document we will show how APEX and APEX Policies can be used to achieve this, starting with a simple policy, building up to more complicated policy that demonstrates the features of APEX.

2. Data Models

2.1. Sales Input Event

Each time a PoS system processes a sales item an event with the following format is emitted:

<table>
<thead>
<tr>
<th>Event</th>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALE_INPUT</td>
<td>time, sale_ID, amount, item_ID, quantity, assistant_ID, branch_ID, notes, ...</td>
<td>Event indicating a sale of an item is occurring</td>
</tr>
</tbody>
</table>

In each SALE_INPUT event the sale_ID field is a unique ID generated by the PoS system. A timestamp for the event is stored in the time field. The amount field refers to the value of the item(s) to be sold (in cents). The item_ID field is a unique identifier for each item type, and can be used to retrieve more information about the item from *HyperM*'s stock control system. The quantity field refers to the quantity of the item to be sold. The assistant_ID field is a unique identifier for the PoS operator, and can be used to retrieve more information about the operator from the *HyperM*'s personnel system. Since *HyperM* has many branches the branch_ID identifies the shop. The notes field contains arbitrary notes about the sale.

2.2. Sales Decision Event

After a SALE_INPUT event is emitted by the PoS system *HyperM*'s policy-based controlled sales checking system emits a Sale Authorization Event indicating whether the sale is authorized or denied. The PoS system can then listen for this event before continuing with the sale.

<table>
<thead>
<tr>
<th>Event</th>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALE_AUTH</td>
<td>sale_ID, time, authorized, amount, item_ID, quantity, assistant_ID, branch_ID, notes, message...</td>
<td>Event indicating a sale of an item is authorized or denied</td>
</tr>
</tbody>
</table>
In each SALE_AUTH event the sale_ID field is copied from the SALE_INPUT event that trigger the decision request. The SALE_AUTH event is also timestamped using the time field, and a field called authorised is set to true or false depending on whether the sale is authorized or denied. The message field carries an optional message about why a sale was not authorized. The other fields from the SALE_INPUT event are also included for completeness.

2.3. Stock Control: Items

HyperM maintains information about each item for sale in a database table called ITEMS.

Table 3. Items Database

<table>
<thead>
<tr>
<th>Table</th>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEMS</td>
<td>item_ID, description, cost_price, barcode, supplier_ID, category, ...</td>
<td>Database table describing each item for sale</td>
</tr>
</tbody>
</table>

The database table ITEMS has a row for each items that HyperM sells. Each item is identified by an item_ID value. The description field stores a description of the item. The cost price of the item is given in cost_price. The barcode of the item is encoded in barcode, while the item supplier is identified by supplier_ID. Items may also be classified into categories using the category field. Useful categories might include: soft drinks, alcoholic drinks, cigarettes, knives, confectionery, bakery, fruit&vegetables, meat, etc..

2.4. Personnel System: Assistants

Table 4. Assistants Database

<table>
<thead>
<tr>
<th>Table</th>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSISTANTS</td>
<td>assistant_ID, surname, firstname, middlename, age, grade, phone_number, ...</td>
<td>Database table describing each HyperM sales assistant</td>
</tr>
</tbody>
</table>

The database table ASSISTANTS has a row for each sales assistant employed by HyperM. Each assistant is identified by an assistant_ID value, with their name given in the firstname, middlename and surname fields. The assistant’s age in years is given in age, while their phone number is contained in the phone_number field. The assistant’s grade is encoded in grade. Useful values for grade might include: trainee, operator, supervisor, etc..

2.5. Locations: Branches

Table 5. Branches Database

<table>
<thead>
<tr>
<th>Table</th>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRANCHES</td>
<td>branch_ID, branch_Name, category, street, city, country, postcode, ...</td>
<td>Database table describing each HyperM branch</td>
</tr>
</tbody>
</table>

HyperM operates a number of branches. Each branch is described in the BRANCHES database table. Each branch is identified by a branch_ID, with a branch name given in branch_Name. The address for the branch is encoded in street, city, country and postcode. The branch category is given in the category field. Useful values for category might include: Small, Large, Super, Hyper, etc..

3. Policy Step 1

3.1. Scenario

For the first version of our policy, let’s start with something simple. Let us assume that there exists some restriction
that alcohol products cannot be sold before 11:30am. In this section we will go through the necessary steps to define a policy that can enforce this for HyperM.

- Alcohol cannot be sold before 11:30am.

3.2. Create the an new empty Policy Model \textit{MyFirstPolicyModel}

Since an organisation like \textit{HyperM} may have many policies covering many different domains, policies should be grouped into policy sets. In order to edit or deploy a policy, or policy set, the definition of the policy(ies) and all required events, tasks, states, etc., are grouped together into a 'Policy Model'. An organization might define many Policy Models, each containing a different set of policies.

So the first step is to create a new empty Policy Model called \textit{MyFirstPolicyModel}. Using the APEX Policy Editor, click on the 'File' menus and select 'New'. Then define our new policy model called \textit{MyFirstPolicyModel}. Use the 'Generate UUID' button to create a new unique ID for the policy model, and fill in a description for the policy model. Press the \textit{Submit} button to save your changes.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure_1.png}
\caption{Create a new Policy Model 1/2}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure_2.png}
\caption{Create a new Policy Model 2/2}
\end{figure}
3.3. Create the input event **SALE_INPUT** and the output event **SALE_AUTH**

Using the APEX Policy Editor, click on the ‘Events’ tab. In the ‘Events’ pane, right click and select ‘New’:

![Event Editor](image)

**Figure 3. Create a new Event type**

Create a new event type called **SALE_INPUT**. Use the ‘Generate UUID’ button to create a new unique ID for the event type, and fill in a description for the event. Add a namespace, e.g. `com.hyperm`. We can add hard-coded strings for the **Source** and **Target**, e.g. **POS** and **APEX**. At this stage we will not add any parameter fields, we will leave this until later. Use the **Submit** button to create the event.

![Event Editor](image)

**Figure 4. Populate the SALE_INPUT event**

Repeat the same steps for a new event type called **SALE_AUTH**. Just use **APEX** as source and **POS** as target, since this is the
output event coming from APEX going to the sales point.

Before we can add parameter fields to an event we must first define APEX Context Item Schemas that can be used by those fields.

To create new item schemas, click on the 'Context Item Schemas' tab. In that 'Context Item Schemas' pane, right click and select 'Create new ContextSchema'.

![Create new ContextSchema](image)

*Figure 5. Create new Data Types*

Create item schemas with the following characteristics, each with its own unique UUID:

*Table 6. Item Schemas*

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema Flavour</th>
<th>Schema Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp_type</td>
<td>Java</td>
<td>java.lang.Long</td>
<td>A type for <em>time</em> values</td>
</tr>
<tr>
<td>sale_ID_type</td>
<td>Java</td>
<td>java.lang.Long</td>
<td>A type for <em>sale_ID</em> values</td>
</tr>
<tr>
<td>price_type</td>
<td>Java</td>
<td>java.lang.Long</td>
<td>A type for <em>amount/price</em> values</td>
</tr>
<tr>
<td>item_ID_type</td>
<td>Java</td>
<td>java.lang.Long</td>
<td>A type for <em>item_ID</em> values</td>
</tr>
<tr>
<td>assistant_ID_type</td>
<td>Java</td>
<td>java.lang.Long</td>
<td>A type for <em>assistant_ID</em> values</td>
</tr>
<tr>
<td>quantity_type</td>
<td>Java</td>
<td>java.lang.Integer</td>
<td>A type for <em>quantity</em> values</td>
</tr>
<tr>
<td>branch_ID_type</td>
<td>Java</td>
<td>java.lang.Long</td>
<td>A type for <em>branch_ID</em> values</td>
</tr>
<tr>
<td>notes_type</td>
<td>Java</td>
<td>java.lang.String</td>
<td>A type for <em>notes</em> values</td>
</tr>
<tr>
<td>authorised_type</td>
<td>Java</td>
<td>java.lang.Boolean</td>
<td>A type for <em>authorised</em> values</td>
</tr>
<tr>
<td>message_type</td>
<td>Java</td>
<td>java.lang.String</td>
<td>A type for <em>message</em> values</td>
</tr>
</tbody>
</table>
The item schemas can now be seen on the 'Context Item Schemas' tab, and can be updated at any time by right-clicking on the item schemas on the 'Context Item Schemas' tab. Now we can go back to the event definitions for SALE_INPUT and SALE_AUTH and add some parameter fields.

Field Schema types
APEX natively supports schema definitions in Java and Avro.

Java schema definitions are simply the name of a Java Class. There are some restrictions:

• the class must be instantiatable, i.e. not an Java interface or abstract class
• primitive types are not supported, i.e. use java.lang.Integer instead of int, etc.
• it must be possible to find the class, i.e. the class must be contained in the Java classpath.

Avro schema definitions can be any valid Avro schema. For events using fields defined with Avro schemas, any incoming event containing that field must contain a value that conforms to the Avro schema.

Click on the 'Events' tab, then right click the SALE_INPUT row and select 'Edit Event SALE_INPUT’. To add a new event parameter use the 'Add Event Parameter' button at the bottom of the screen. For the 'SALE_INPUT' event add the following event parameters:

Table 7. Event Parameter Fields for the SALE_INPUT Event
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>timestamp_type</td>
<td>no</td>
</tr>
<tr>
<td>sale_ID</td>
<td>sale_ID_type</td>
<td>no</td>
</tr>
<tr>
<td>amount</td>
<td>price_type</td>
<td>no</td>
</tr>
<tr>
<td>item_ID</td>
<td>item_ID_type</td>
<td>no</td>
</tr>
<tr>
<td>quantity</td>
<td>quantity_type</td>
<td>no</td>
</tr>
<tr>
<td>assistant ID</td>
<td>assistant_ID_type</td>
<td>no</td>
</tr>
<tr>
<td>branch ID</td>
<td>branch_ID_type</td>
<td>no</td>
</tr>
<tr>
<td>notes</td>
<td>notes_type</td>
<td>yes</td>
</tr>
</tbody>
</table>

Remember to click the ‘Submit’ button at the bottom of the event definition pane.

*Optional Fields in APEX Events*

Parameter fields can be *optional* in events. If a parameter is not marked as *optional* then by default it is *mandatory*, so it must appear in any input event passed to APEX. If an *optional* field is not set for an output event then value will be set to *null*.

![Figure 7. Add typed parameter fields to an event](image)

Select the **SALE_AUTH** event and add the following event parameters:

*Table 8. Event Parameter Fields for the **SALE_AUTH** Event*

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>sale_ID</td>
<td>sale_ID_type</td>
<td>no</td>
</tr>
<tr>
<td>time</td>
<td>timestamp_type</td>
<td>no</td>
</tr>
<tr>
<td>authorised</td>
<td>authorised_type</td>
<td>no</td>
</tr>
<tr>
<td>message</td>
<td>message_type</td>
<td>yes</td>
</tr>
<tr>
<td>amount</td>
<td>price_type</td>
<td>no</td>
</tr>
<tr>
<td>item_ID</td>
<td>item_ID_type</td>
<td>no</td>
</tr>
<tr>
<td>assistant ID</td>
<td>assistant_ID_type</td>
<td>no</td>
</tr>
<tr>
<td>quantity</td>
<td>quantity_type</td>
<td>no</td>
</tr>
<tr>
<td>branch ID</td>
<td>branch_ID_type</td>
<td>no</td>
</tr>
<tr>
<td>notes</td>
<td>notes_type</td>
<td>yes</td>
</tr>
</tbody>
</table>
Remember to click the 'Submit' button at the bottom of the event definition pane.

The events for our policy are now defined.

3.4. Create a new Policy and add the "No Booze before 11:30" check

APEX policies are defined using a state-machine model. Each policy comprises one or more states that can be individually executed. Where there is more than one state the states are chained together to form a Directed Acyclic Graph (DAG) of states. A state is triggered by passing it a single input (or 'trigger') event and once executed each state then emits an output event. For each state the logic for the state is embedded in one or more tasks. Each task contains specific task logic that is executed by the APEX execution environment each time the task is invoked. Where there is more than one task in a state then the state also defines some task selection logic to select an appropriate task each time the state is executed.

Therefore, to create a new policy we must first define one or more tasks.

To create a new Task click on the 'Tasks' tab. In the 'Tasks' pane, right click and select 'Create new Task'. Create a new Task called MorningBoozeCheck. Use the 'Generate UUID' button to create a new unique ID for the task, and fill in a description for the task.

![APEX Policy Editor](image)

Figure 8. Create a new Task

Tasks are configured with a set of input fields and a set of output fields. To add new input/output fields for a task use the 'Add Task Input Field' and 'Add Task Output Field' button. The list of input and out fields to add for the MorningBoozeCheck task are given below. The input fields are drawn from the parameters in the state's input event, and the task's output fields are used to populate the state's output event. The task's input and output fields must be a subset of the event parameters defined for the input and output events for any state that uses that task. (You may have noticed that the input and output fields for the MorningBoozeCheck task have the exact same names and reuse the item schemas that we used for the parameters in the SALE_INPUT and SALE_AUTH events respectively).

<table>
<thead>
<tr>
<th>Table 9. Input fields for MorningBoozeCheck task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: MorningBoozeCheck</td>
</tr>
<tr>
<td>Description: This task checks if the sales request is for an item that contains alcohol. If the local time is between 00:30:00 and 11:30:00 then the sale is not authorised. Otherwise the sale is authorised. In this implementation we assume that all items with item_ID values between 1000 and 2000 contain alcohol. :}</td>
</tr>
</tbody>
</table>

8
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>timestamp_type</td>
</tr>
<tr>
<td>sale_ID</td>
<td>sale_ID_type</td>
</tr>
<tr>
<td>amount</td>
<td>price_type</td>
</tr>
<tr>
<td>item_ID</td>
<td>item_ID_type</td>
</tr>
<tr>
<td>quantity</td>
<td>quantity_type</td>
</tr>
<tr>
<td>assistant_ID</td>
<td>assistant_ID_type</td>
</tr>
<tr>
<td>branch_ID</td>
<td>branch_ID_type</td>
</tr>
<tr>
<td>notes</td>
<td>notes_type</td>
</tr>
</tbody>
</table>

Table 10. Output fields for MorningBoozeCheck task

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sale_ID</td>
<td>sale_ID_type</td>
</tr>
<tr>
<td>time</td>
<td>timestamp_type</td>
</tr>
<tr>
<td>authorised</td>
<td>authorised_type</td>
</tr>
<tr>
<td>message</td>
<td>message_type</td>
</tr>
<tr>
<td>amount</td>
<td>price_type</td>
</tr>
<tr>
<td>item_ID</td>
<td>item_ID_type</td>
</tr>
<tr>
<td>assistant_ID</td>
<td>assistant_ID_type</td>
</tr>
<tr>
<td>quantity</td>
<td>quantity_type</td>
</tr>
<tr>
<td>branch_ID</td>
<td>branch_ID_type</td>
</tr>
<tr>
<td>notes</td>
<td>notes_type</td>
</tr>
</tbody>
</table>
Each task must include some 'Task Logic' that implements the behaviour for the task. Task logic can be defined in a number of different ways using a choice of languages. For this task we will author the logic using the Java-like scripting language called MVEL.

For simplicity use the following code for the task logic. Paste the script text into the 'Task Logic' box, and use "MVEL" as the 'Task Logic Type / Flavour'.

This logic assumes that all items with \texttt{item\_ID} between 1000 and 2000 contain alcohol, which is not very realistic, but we will see a better approach for this later. It also uses the standard Java time utilities to check if the current time is between \texttt{00:00:00 GMT} and \texttt{11:30:00 GMT}. For a detailed guide to how to write your own logic in JavaScript, MVEL or one of the other supported languages please refer to APEX Programmers Guide.

\textbf{MVEL code for the MorningBoozeCheck task}

```/
/*
 * Copyright (C) 2016-2018 Ericsson. All rights reserved.
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 */
```

\textbf{Figure 9. Add input and out fields for the Task}
/*
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * SPDX-License-Identifier: Apache-2.0
 */

import java.util.Date;
import java.util.Calendar;
import java.util.TimeZone;
import java.text.SimpleDateFormat;

logger.info("Task Execution: "+subject.id+". Input Fields: "+inFields+"");

outFields.put("amount", inFields.get("amount"));
outFields.put("assistant_ID", inFields.get("assistant_ID"));
outFields.put("notes", inFields.get("notes"));
outFields.put("quantity", inFields.get("quantity"));
outFields.put("branch_ID", inFields.get("branch_ID"));
outFields.put("item_ID", inFields.get("item_ID"));
outFields.put("time", inFields.get("time"));
outFields.put("sale_ID", inFields.get("sale_ID"));

item_id = inFields.get("item_ID");

//The events used later to test this task use GMT timezone!
gmt = TimeZone.getTimeZone("GMT");
timenow = Calendar.getInstance(gmt);
df = new SimpleDateFormat("HH:mm:ss z");
df.setTimeZone(gmt);
timenow.setTimeInMillis(inFields.get("time"));

midnight = timenow.clone();
midnight.set(
    timenow.get(Calendar.YEAR), timenow.get(Calendar.MONTH),
    timenow.get(Calendar.DATE), 11, 30, 0);

if (item_id != null && item_id >= 1000 && item_id < 2000)
    itemisalcohol = true;
else if (itemisalcohol && timenow.after(midnight) && timenow.before(eleven30))
    outFields.put("authorised", false);
    outFields.put("message", "Sale not authorised by policy task "+subject.taskName+
        " for time "+df.format(timenow.getTime())+
        " Alcohol can not be sold between "+df.format(midnight.getTime())+
        " and "+df.format(eleven30.getTime()));
    return true;
} else {
    outFields.put("authorised", true);
    outFields.put("message", "Sale authorised by policy task "+subject.taskName+
        " for time "+df.format(timenow.getTime()));
    return true;
}
*/
This task checks if a sale request is for an item that is an alcoholic drink. If the local time is between 00:00:00 GMT and 11:30:00 GMT then the sale is not authorised. Otherwise the sale is authorised. In this implementation we assume that items with item_ID value between 1000 and 2000 are all alcoholic drinks :-) /*

**Task logic:**

```javacode
var returnValueType = Java.type("java.lang.Boolean");
var returnValue = new returnValueType(true);
```

Figure 10. Add Task Logic the Task

An alternative version of the same logic is available in JavaScript. Just use “JAVASCRIPT” as the ‘Task Logic Type / Flavour’ instead.

**Javascript alternative for the MorningBoozeCheck task**

```javascript
/*
 * ============LICENSE_START=======================================================
 * Copyright (C) 2016-2018 Ericsson. All rights reserved.
 * ================================================================================
 * Licensed under the Apache License, Version 2.0 (the "License";
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * SPDX-License-Identifier: Apache-2.0
 * ============LICENSE_END=========================================================*/

var returnvaluetype = Java.type("java.lang.Boolean");
var returnvalue = new returnvaluetype(true);
```

// Load compatibility script for imports etc
dojo.require("nashorn:mozilla_compat.js");
importPackage(java.text);
Each policy must have at least one state. Since this is 'freeform' policy we can add as many states as we wish. Let's create a new Policy called MyFirstPolicy. Use the 'Generate UUID' button to create a new unique ID for the policy, and fill in a description for the policy. Use 'FREEFORM' as the 'Policy Flavour'.

This task checks if a sale request is for an item that is an alcoholic drink. If the local time is between 00:00:00 GMT and 11:30:00 GMT then the sale is not authorised. Otherwise the sale is authorised.

The task definition is now complete so click the 'Submit' button to save the task. The task can now be seen on the 'Tasks' tab, and can be updated at any time by right-clicking on the task on the 'Task' tab. Now that we have created our task, we can can create a policy that uses that task.
start with one state. Add a new state called BoozeAuthDecide to this MyFirstPolicy policy using the 'Add new State' button after filling in the name of our new state.

Figure 11. Create a new Policy

Each state must use one input event type. For this new state select the SALE_INPUT event as the input event.

Each policy must define a 'First State' and a 'Policy Trigger Event'. The 'Policy Trigger Event' is the input event for the policy as a whole. This event is then passed to the first state in the chain of states in the policy, therefore the 'Policy Trigger Event' will be the input event for the first state. Each policy can only have one 'First State'. For our MyFirstPolicy policy, select BoozeAuthDecide as the 'First State'. This will automatically select SALE_INPUT as the 'Policy Trigger Event' for our policy.
In this case we will create a reference the pre-existing MorningBoozeCheck task that we defined above using the ‘Add New Task’ button. Select the MorningBoozeCheck task, and use the name of the task as the ‘Local Name’ for the task.

In the case where a state references more than one task, a ‘Default Task’ must be selected for the state and some logic (‘Task Selection Logic’) must be specified to select the appropriate task at execution time. Since our new state BoozeAuthDecide only has one task the default task is automatically selected and no ‘Task Selection Logic’ is required.
State Output Mappings

In a 'Policy' 'State' a 'State Output Mapping' has 3 roles: 1) Select which 'State' should be executed next, 2) Select the type of the state's 'Outgoing Event', and 3) Populate the state's 'Outgoing Event'. This is how states are chained together to form a (Directed Acyclic Graph (DAG)) of states. The final state(s) of a policy are those that do not select any 'next' state. Since a 'State' can only accept a single type of event, the type of the event emitted by a previous 'State' must be match the incoming event type of the next 'State'. This is also how the last state(s) in a policy can emit events of different types. The 'State Output Mapping' is also responsible for taking the fields that are output by the task executed in the state and populating the state's output event before it is emitted.

Each 'Task' referenced in 'State' must have a defined 'Output Mapping' to take the output of the task, select an 'Outgoing Event' type for the state, populate the state's outgoing event, and then select the next state to be executed (if any).

There are 2 basic types of output mappings:

1. **Direct Output Mappings** have a single value for 'Next State' and a single value for 'State Output Event'. The outgoing event for the state is automatically created, any outgoing event parameters that were present in the incoming event are copied into the outgoing event, then any task output fields that have the same name and type as parameters in the outgoing event are automatically copied into the outgoing event.

2. **Logic-based State Output Mappings / Finalizers** have some logic defined that dynamically selects and creates the 'State Outgoing Event', manages the population of the outgoing event parameters (perhaps changing or adding to the outputs from the task), and then dynamically selects the next state to be executed (if any).

Each task reference must also have an associated 'Output State Mapping' so we need an 'Output State Mapping' for the BoozeAuthDecide state to use when the MorningBoozeCheck task is executed. The simplest type of output mapping is a 'Direct Output Mapping'.

Create a new 'Direct Output Mapping' for the state called MorningBoozeCheck_Output_Direct using the 'Add New Direct State Output Mapping' button. Select SALE_AUTH as the output event and select None for the next state value. We can then select this output mapping for use when the MorningBoozeCheck task is executed. Since there is only state, and only one task for that state, this output mapping ensures that the BoozeAuthDecide state is the only state executed and the state (and the policy) can only emit events of type SALE_AUTH. (You may remember that the output fields for the MorningBoozeCheck task have the exact same names and reuse the item schemas that we used for the parameters in SALE_AUTH event. The MorningBoozeCheck_Output_Direct direct output mapping can now automatically copy the values from the MorningBoozeCheck task directly into outgoing SALE_AUTH events.)
Click the 'Submit' button to complete the definition of our **MyFirstPolicy** policy. The policy **MyFirstPolicy** can now be seen in the list of policies on the 'Policies' tab, and can be updated at any time by right-clicking on the policy on the 'Policies' tab.

The **MyFirstPolicyModel**, including our **MyFirstPolicy** policy can now be checked for errors. Click on the 'Model' menu and select 'Validate'. The model should validate without any 'Warning' or 'Error' messages. If you see any 'Error' or 'Warning' messages, carefully read the message as a hint to find where you might have made a mistake when defining some aspect of your policy model.

Congratulations, you have now completed your first APEX policy. The policy model containing our new policy can now be exported from the editor and saved. Click on the 'File' menu and select 'Download' to save the policy model in JSON format. The exported policy model is then available in the directory you selected, for instance $APEX_HOME/examples/models/MyFirstPolicy/1/MyFirstPolicyModel_0.0.1.json. The exported policy can now be loaded into the APEX Policy Engine, or can be re-loaded and edited by the APEX Policy Editor.
3.5. Test Policy Step 1

To start a new APEX Engine you can use the following configuration. In a full APEX installation you can find this configuration in \$APEX_HOME/examples/config/MyFirstPolicy/1/MyFirstPolicyConfigStdin2StdoutJsonEvent.json. This configuration expects incoming events to be in JSON format and to be passed into the APEX Engine from stdin, and result events will be printed in JSON format to stdout. This configuration loads the policy model stored in the file 'MyFirstPolicyModel_0.0.1.json' as exported from the APEX Editor. Note, you may need to edit this file to provide the full path to wherever you stored the exported policy model file.
To test the policy try paste the following events into the console as the APEX engine executes:

Table 11. Inputs and Outputs when testing My First Policy
<table>
<thead>
<tr>
<th>Input Event (JSON)</th>
<th>Output Event (JSON)</th>
<th>comment</th>
</tr>
</thead>
</table>
| {  
  "nameSpace": "com.hyperm",
  "name": "SALE_INPUT",
  "version": "0.0.1",
  "time": 1483351989000,
  "sale_ID": 99999991,
  "amount": 299,
  "item_ID": 5123,
  "quantity": 1,
  "assistant_ID": 23,
  "branch_ID": 1,
  "notes": "Special Offer!!"
} | {  
  "name": "SALE_AUTH",
  "version": "0.0.1",
  "nameSpace": "com.hyperm",
  "source": "",
  "target": "",
  "amount": 299,
  "assistant_ID": 23,
  "authorised": true,
  "branch_ID": 1,
  "item_ID": 5123,
  "message": "Sale authorised by policy task MorningBoozeCheck for time 10:13:09 GMT",
  "notes": "Special Offer!!",
  "quantity": 1,
  "sale_ID": 99999991,
  "time": 1483351989000
} | Request to buy a non-alcoholic item (item_ID=5123) at 10:13:09 on Tuesday, 10 January 2017. Sale is authorized. |
| {  
  "nameSpace": "com.hyperm",
  "name": "SALE_INPUT",
  "version": "0.0.1",
  "time": 1483346466000,
  "sale_ID": 99999992,
  "amount": 1249,
  "item_ID": 1012,
  "quantity": 1,
  "assistant_ID": 12,
  "branch_ID": 2
} | {  
  "nameSpace": "com.hyperm",
  "name": "SALE_AUTH",
  "version": "0.0.1",
  "source": "",
  "target": "",
  "amount": 1249,
  "assistant_ID": 12,
  "authorised": false,
  "branch_ID": 2,
  "item_ID": 1012,
  "message": "Sale not authorised by policy task MorningBoozeCheck for time 08:41:06 GMT. Alcohol can not be sold between 00:00:00 GMT and 11:30:00 GMT",
  "notes": null,
  "quantity": 1,
  "sale_ID": 99999992,
  "time": 1483346466000
} | Request to buy alcohol item (item_ID=1012) at 08:41:06 on Monday, 02 January 2017. Sale is not authorized. |
3.6. Policy 1 in CLI Editor

An equivalent version of the MyFirstPolicyModel policy model can again be generated using the APEX CLI editor. A sample APEX CLI script is shown below:

APEX CLI Editor code for Policy 1

```bash
# Copyright (C) 2016-2018 Ericsson. All rights reserved.
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# SPDX-License-Identifier: Apache-2.0
#
(model create name=MyFirstPolicyModel version=0.0.1 uuid=540226fb-55ee-4f0e-a444-983a0494818e
description="This is my first Apex Policy Model.")

(schema create name=assistant_ID_type version=0.0.1 uuid=36df4c71-9616-4206-8b53-976a5cd4bd87
description="A type for 'assistant_ID' values" flavour=Java schema=java.lang.Long)

(schema create name=authorised_type version=0.0.1 uuid=d48b619e-d0d0-4008-b884-02d76ea4350b
description="A type for 'authorised' values" flavour=Java schema=java.lang.Boolean)

(schema create name=branch_ID_type version=0.0.1 uuid=6468845f-4122-4128-8e49-0f52c26078b5
description="A type for 'branch_ID' values" flavour=Java schema=java.lang.Long)
```
schema create name=item_ID_type version=0.0.1 uuid=4f227ff1-ae08-453a-b6b6-9a4b2e0da932 description="A type for 'item_ID' values" flavour=Java schema=java.lang.Long

schema create name=message_type version=0.0.1 uuid=ad1431bb-3155-4e73-b5a3-b89bee498749 description="A type for 'message' values" flavour=Java schema=java.lang.String

schema create name=notes_type version=0.0.1 uuid=eecefe98-896c-4343-8f9c-26b3ced94e2d description="A type for 'notes' values" flavour=Java schema=java.lang.String

schema create name=price_type version=0.0.1 uuid=52c2fca5-fd8c-463c-bd6f-d91b0554ae7 description="A type for 'amount'/price values" flavour=Java schema=java.lang.Long

schema create name=quantity_type version=0.0.1 uuid=ac3d9842-80af-4a98-951c-bd79a431c613 description="A type for 'quantity' values" flavour=Java schema=java.lang.Integer

schema create name=sale_ID_type version=0.0.1 uuid=cca47d74-7754-4a61-b163-ca31f66b157b description="A type for 'sale_ID' values" flavour=Java schema=java.lang.Long

schema create name=timestamp_type version=0.0.1 uuid=fd594e88-411d-4a94-b2be-697b3a0d7ad description="A type for 'time' values" flavour=Java schema=java.lang.Long

task create name=MorningBoozeCheck version=0.0.1 uuid=3351b0f4-cf06-4fa2-8823-edf67bd38223 description=LS
This task checks if the sales request is for an item that contains alcohol. If the local time is between 00:00:00 and 11:30:00 then the sale is not authorised. Otherwise the sale is authorised.
In this implementation we assume that all items with item_ID values between 1000 and 2000 contain alcohol :-)

LE

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=sale_ID schemaName=sale_ID_type schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=amount schemaName=price_type schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=assistant_ID schemaName=assistant_ID_type schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=notes schemaName=notes_type schemaVersion=0.0.1 optional=true

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=quantity schemaName=quantity_type schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=branch_ID schemaName=branch_ID_type schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=item_ID schemaName=item_ID_type schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=time schemaName=timestamp_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=sale_ID schemaName=sale_ID_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=amount schemaName=price_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=assistant_ID schemaName=assistant_ID_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=notes schemaName=notes_type schemaVersion=0.0.1 optional=true

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=quantity schemaName=quantity_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=branch_ID schemaName=branch_ID_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=item_ID schemaName=item_ID_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=time schemaName=timestamp_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=message schemaName=message_type schemaVersion=0.0.1 optional=true

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=authorised schemaName=authorised_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=branch_ID schemaName=branch_ID_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=item_ID schemaName=item_ID_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=time schemaName=timestamp_type schemaVersion=0.0.1

(task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=message schemaName=message_type schemaVersion=0.0.1 optional=true

(task logic create name=MorningBoozeCheck version=0.0.1 logicFlavour=MVEL logic=LS
import java.util.Date;
import java.util.Calendar;
import java.util.TimeZone;
import java.text.SimpleDateFormat;
logger.info("Task Execution: '" + subject.id + ". Input Fields: '" + inFields + ":\n        outFields.put("amount" , inFields.get("amount"));
        outFields.put("assistant_ID", inFields.get("assistant_ID"));
        outFields.put("notes"    , inFields.get("notes"));
        outFields.put("quantity", inFields.get("quantity"));
        outFields.put("branch_ID", inFields.get("branch_ID"));
        outFields.put("item_ID" , inFields.get("item_ID"));
        outFields.put("time"    , inFields.get("time"));
        outFields.put("sale_ID" , inFields.get("sale_ID"));

item_id = inFields.get("item_ID");

//The events used later to test this task use GMT timezone!
gmt = TimeZone.getTimeZone("GMT");
timenow = Calendar.getInstance(gmt);
df = new SimpleDateFormat("HH:mm:ss z");
df.setTimeZone(gmt);
timenow.setTimeInMillis(inFields.get("time"));
midnight = timenow.clone();
midnight.set(
        timenow.get(Calendar.YEAR),timenow.get(Calendar.MONTH),
        timenow.get(Calendar.DATE),0,0,0);

        eleven30 = timenow.clone();
        eleven30.set(
        timenow.get(Calendar.YEAR),timenow.get(Calendar.MONTH),
        timenow.get(Calendar.DATE),11,30,0);

itemisalcohol = false;
if(item_id != null & item_id >=1000 & item_id < 2000)
    itemisalcohol = true;

if( itemisalcohol & timenow.after(midnight) & timenow.before(eleven30))
    outFields.put("authorised", false);
    outFields.put("message", "Sale not authorised by policy task "+subject.taskName+ 
                " for time "+df.format(timenow.getTime())+
                ". Alcohol can not be sold between "+df.format(midnight.getTime())+ 
                " and "+df.format(eleven30.getTime()));
    return true;
} else{
    outFields.put("authorised", true);
    outFields.put("message", "Sale authorised by policy task "+subject.taskName+
        " for time "+df.format(timenow.getTime()));
    return true;
}

/*
This task checks if a sale request is for an item that is an alcoholic drink.
If the local time is between 00:00:00 GMT and 11:30:00 GMT then the sale is not
authorised. Otherwise the sale is authorised.
In this implementation we assume that items with item_ID value between 1000 and
2000 are all alcoholic drinks :-) */

LE

event create name=SALE_AUTH version=0.0.1 uuid=c4509941-3f98-4080-a9cc-5b9753ed050b description="An event emitted by the Policy to indicate whether the sale of an item has been authorised" nameSpace=com.hyperm source="APEX" target="POS"
    event parameter create name=SALE_AUTH version=0.0.1 parName=amount schemaName=price_type schemaVersion=0.0.1
    event parameter create name=SALE_AUTH version=0.0.1 parName=assistant_ID schemaName=assistant_ID_type schemaVersion=0.0.1
    event parameter create name=SALE_AUTH version=0.0.1 parName=authorised schemaName=authorised_type schemaVersion=0.0.1
    event parameter create name=SALE_AUTH version=0.0.1 parName=branch_ID schemaName=branch_ID_type schemaVersion=0.0.1
    event parameter create name=SALE_AUTH version=0.0.1 parName=item_ID schemaName=item_ID_type schemaVersion=0.0.1
    event parameter create name=SALE_AUTH version=0.0.1 parName=message schemaName=message_type schemaVersion=0.0.1 optional=true
    event parameter create name=SALE_AUTH version=0.0.1 parName=notes schemaName=notes_type schemaVersion=0.0.1 optional=true
    event parameter create name=SALE_AUTH version=0.0.1 parName=quantity schemaName=quantity_type schemaVersion=0.0.1
    event parameter create name=SALE_AUTH version=0.0.1 parName=sale_ID schemaName=sale_ID_type schemaVersion=0.0.1
    event parameter create name=SALE_AUTH version=0.0.1 parName=time schemaName=timestamp_type schemaVersion=0.0.1

event create name=SALE_INPUT version=0.0.1 uuid=4f84aa98-e917-4f4a-882a-c75ba5a9374 description="An event raised by the PoS system each time an item is scanned for purchase" nameSpace=com.hyperm source="POS" target="APEX"
    event parameter create name=SALE_INPUT version=0.0.1 parName=amount schemaName=price_type schemaVersion=0.0.1
    event parameter create name=SALE_INPUT version=0.0.1 parName=assistant_ID schemaName=assistant_ID_type schemaVersion=0.0.1
    event parameter create name=SALE_INPUT version=0.0.1 parName=branch_ID schemaName=branch_ID_type schemaVersion=0.0.1
    event parameter create name=SALE_INPUT version=0.0.1 parName=item_ID schemaName=item_ID_type schemaVersion=0.0.1
    event parameter create name=SALE_INPUT version=0.0.1 parName=notes schemaName=notes_type schemaVersion=0.0.1 optional=true
    event parameter create name=SALE_INPUT version=0.0.1 parName=quantity schemaName=quantity_type schemaVersion=0.0.1
    event parameter create name=SALE_INPUT version=0.0.1 parName=sale_ID schemaName=sale_ID_type schemaVersion=0.0.1
    event parameter create name=SALE_INPUT version=0.0.1 parName=time schemaName=timestamp_type schemaVersion=0.0.1

policy create name=MyFirstPolicy version=0.0.1 uuid=6c5e410f-489a-46ff-964e-982ce68b6d0 description="This is my first Apex policy. It checks if a sale should be authorised or not." template=FREEFORM
4. Policy Step 2

4.1. Scenario

HyperM have just opened a new branch in a different country, but that country has different rules about when alcohol can be sold! In this section we will go through the necessary steps to extend our policy to enforce this for HyperM.

• In some branches alcohol cannot be sold before 1pm, and not at all on Sundays.

Although there are a number of ways to accomplish this the easiest approach for us is to define another task and then select which task is appropriate at runtime depending on the branch identifier in the incoming event.

4.2. Extend the Policy with the new Scenario

To create a new Task click on the 'Tasks' tab. In the 'Tasks' pane, right click and select 'Create new Task':

Create a new Task called MorningBoozeCheckAlt1. Use the 'Generate UUID' button to create a new unique ID for the task, and fill in a description for the task. Select the same input and output fields that we used earlier when we defined the MorningBoozeCheck task earlier.

Table 12. Input fields for MorningBoozeCheckAlt1 task

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>timestamp_type</td>
</tr>
<tr>
<td>sale_ID</td>
<td>sale_ID_type</td>
</tr>
<tr>
<td>amount</td>
<td>price_type</td>
</tr>
<tr>
<td>item_ID</td>
<td>item_ID_type</td>
</tr>
<tr>
<td>quantity</td>
<td>quantity_type</td>
</tr>
<tr>
<td>assistant_ID</td>
<td>assistant_ID_type</td>
</tr>
<tr>
<td>branch_ID</td>
<td>branch_ID_type</td>
</tr>
<tr>
<td>notes</td>
<td>notes_type</td>
</tr>
</tbody>
</table>

Table 13. Output fields for MorningBoozeCheckAlt1 task

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sale_ID</td>
<td>sale_ID_type</td>
</tr>
<tr>
<td>time</td>
<td>timestamp_type</td>
</tr>
<tr>
<td>authorised</td>
<td>authorised_type</td>
</tr>
<tr>
<td>message</td>
<td>message_type</td>
</tr>
<tr>
<td>amount</td>
<td>price_type</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Parameter Type</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>item_ID</td>
<td>item_ID_type</td>
</tr>
<tr>
<td>assistant_ID</td>
<td>assistant_ID_type</td>
</tr>
<tr>
<td>quantity</td>
<td>quantity_type</td>
</tr>
<tr>
<td>branch_ID</td>
<td>branch_ID_type</td>
</tr>
<tr>
<td>notes</td>
<td>notes_type</td>
</tr>
</tbody>
</table>

This task also requires some 'Task Logic' to implement the new behaviour for this task.

For simplicity use the following code for the task logic. It again assumes that all items with item_ID between 1000 and 2000 contain alcohol. We again use the standard Java time utilities to check if the current time is between 00:00:00 CET and 13:00:00 CET or if it is Sunday.

For this task we will again author the logic using the MVEL scripting language. Sample task logic code (specified in MVEL) is given below. For a detailed guide to how to write your own logic in JavaScript, MVEL or one of the other supported languages please refer to APEX Programmers Guide.

**MVEL code for the MorningBoozeCheckAlt1 task**

```mvel
import java.util.Date;
import java.util.Calendar;
import java.util.TimeZone;
import java.text.SimpleDateFormat;

logger.info("Task Execution: "+subject.id". Input Event: "+inFields";

outFields.put("amount", inFields.get("amount"));
outFields.put("assistant_ID", inFields.get("assistant_ID"));
outFields.put("notes", inFields.get("notes"));
outFields.put("quantity", inFields.get("quantity"));
outFields.put("branch_ID", inFields.get("branch_ID"));
outFields.put("item_ID", inFields.get("item_ID"));
outFields.put("time", inFields.get("time"));
outFields.put("sale_ID", inFields.get("sale_ID"));

item_id = inFields.get("item_ID");

//The events used later to test this task use CET timezone!
cet = TimeZone.getTimeZone("CET");
timernow = Calendar.getInstance(cet);
df = new SimpleDateFormat("HH:mm:ss z");
```

26
df.setTimeZone(new TimeZonedDateTime(cet, false));
timenow.setTimeInMillis(inFields.get("time"));

midnight = timenow.clone();
midnight.set(Calendar.YEAR, timenow.get(Calendar.YEAR) - 1, timenow.get(Calendar.MONTH), 0, 0, 0);
onepm = timenow.clone();
onepm.set(Calendar.YEAR, timenow.get(Calendar.YEAR), Calendar.MONTH + 1, 0, 0, 0);

itemisalcohol = false;
if (item_id != null && item_id >= 1000 && item_id < 2000)
  itemisalcohol = true;
if (itemisalcohol &&
   (timenow.after(midnight) && timenow.before(onepm))
   ||
   (timenow.get(Calendar.DAY_OF_WEEK) == Calendar.SUNDAY))
  {
    outFields.put("authorised", false);
    outFields.put("message", "Sale not authorised by policy task " + subject.taskName +
      " for time " + df.format(timenow.getTime()) +
      " Alcohol can not be sold between " + df.format(midnight.getTime()) +
      " and " + df.format(onepm.getTime()) + " or on Sunday");
    return true;
  }
else
  {
    outFields.put("authorised", true);
    outFields.put("message", "Sale authorised by policy task " + subject.taskName +
      " for time " + df.format(timenow.getTime()));
    return true;
  }

/*
This task checks if a sale request is for an item that is an alcoholic drink.
If the local time is between 00:00:00 CET and 13:00:00 CET then the sale is not authorised.
Also alcohol sales are not allowed on Sundays. Otherwise the sale is authorised.
In this implementation we assume that items with item_ID between 1000 and 2000 are all alcoholic drinks :-)
*/
Figure 16. Create a new Task
The task definition is now complete so click the 'Submit' button to save the task. Now that we have created our task, we can add this task to the single pre-existing state (BoozeAuthDecide) in our policy.

To edit the BoozeAuthDecide state in our policy click on the 'Policies' tab. In the 'Policies' pane, right click on our MyFirstPolicy policy and select 'Edit'. Navigate to the BoozeAuthDecide state in the 'states' section at the bottom of the policy definition pane.

![Image of Apex Policy Editor]

Figure 17. Edit a Policy

To add our new task MorningBoozeCheckAlt1, scroll down to the BoozeAuthDecide state in the 'States' section. In the 'State Tasks' section for BoozeAuthDecide use the 'Add new task' button. Select our new MorningBoozeCheckAlt1 task, and use the name of the task as the 'Local Name' for the task. The MorningBoozeCheckAlt1 task can reuse the same MorningBoozeCheck_Output_Direct 'Direct State Output Mapping' that we used for the MorningBoozeCheck task. (Recall that the role of the 'State Output Mapping' is to select the output event for the state, and select the next state to be executed. These both remain the same as before.)

Since our state has more than one task we must define some logic to determine which task should be used each time the state is executed. This task selection logic is defined in the state definition. For our BoozeAuthDecide state we want the choice of which task to use to be based on the branch_ID from which the SALE_INPUT event originated. For simplicity sake let us assume that branches with branch_ID between 0 and 999 should use the MorningBoozeCheck task, and the branches with with branch_ID between 1000 and 1999 should use the MorningBoozeCheckAlt1 task.

This time, for variety, we will author the task selection logic using the JavaScript scripting language. Sample task selection logic code (specified in JavaScript) is given below. Paste the script text into the 'Task Selection Logic' box, and use "JAVASCRIPT" as the 'Task Selection Logic Type / Flavour'. It is necessary to mark one of the tasks as the 'Default Task' so that the task selection logic always has a fallback default option in cases where a particular task cannot be selected. In this case the MorningBoozeCheck task can be the default task.
JavaScript code for the BoozeAuthDecide task selection logic

```javascript
/*
 * ============LICENSE_START=======================================================
 * Copyright (C) 2016-2018 Ericsson. All rights reserved.
 * ============LICENSE_END=========================================================*/

var returnValueType = Java.type("java.lang.Boolean");
var returnValue = new returnValueType(true);

executor.logger.info("Task Selection Execution: "+executor.subject.id+". Input Event: "+executor.inFields);
branchid = executor.inFields.get("branch_ID");
taskorig = executor.subject.getTaskKey("MorningBoozeCheck");
taskalt = executor.subject.getTaskKey("MorningBoozeCheckAlt1");
taskdef = executor.subject.getDefaultTaskKey();

if(branchid >= 0 && branchid <1000){
    taskorig.copyTo(executor.selectedTask);
} else if (branchid >=1000 && branchid <2000){
    taskalt.copyTo(executor.selectedTask);
} else{
    taskdef.copyTo(executor.selectedTask);
}

/*
This task selection logic selects task "MorningBoozeCheck" for branches with 
0<=branch_ID<1000 and selects task "MorningBoozeCheckAlt1" for branches with 
1000<=branch_ID<2000. Otherwise the default task is selected.
In this case the default task is also "MorningBoozeCheck"
*/
```
Figure 18. State definition with 2 Tasks and Task Selection Logic

When complete don’t forget to click the ‘Submit’ button at the bottom of ‘Policies’ pane for our MyFirstPolicy policy after updating the BoozeAuthDecide state.

Congratulations, you have now completed the second step towards your first APEX policy. The policy model containing our new policy can again be validated and exported from the editor and saved as shown in Step 1.

The exported policy model is then available in the directory you selected, as MyFirstPolicyModel_0.0.1.json. The exported policy can now be loaded into the APEX Policy Engine, or can be re-loaded and edited by the APEX Policy Editor.

4.3. Test Policy Step 2

To start a new APEX Engine you can use the following configuration. In a full APEX installation you can find this configuration in $APEX_HOME/examples/config/MyFirstPolicy/2/MyFirstPolicyConfigStdin2StdoutJsonEvent.json. Note, this has changed from the configuration file in Step 1 to enable the JAVASCRIPT executor for our new ‘Task Selection Logic’.
To test the policy try paste the following events into the console as the APEX engine executes. Note, all tests from Step 1 will still work perfectly since none of those events originate from a branch with branch_ID between 1000 and 2000. The ‘Task Selection Logic’ will therefore pick the MorningBoozeCheck task as expected, and will therefore give the same results.

Table 14. Inputs and Outputs when testing My First Policy
<table>
<thead>
<tr>
<th>Input Event (JSON)</th>
<th>Output Event (JSON)</th>
<th>comment</th>
</tr>
</thead>
</table>
| {  
|    "nameSpace": "com.hyperm",
|    "name": "SALE_INPUT",
|    "version": "0.0.1",
|    "time": 1483346466000,
|    "sale_ID": 99999992,
|    "amount": 1249,
|    "item_ID": 1012,
|    "assistant_ID": 12,
|    "branch_ID": 2
| } | {  
|    "nameSpace": "com.hyperm",
|    "name": "SALE_AUTH",
|    "version": "0.0.1",
|    "source": "",
|    "target": "",
|    "amount": 1249,
|    "assistant_ID": 12,
|    "authorised": false,
|    "branch_ID": 2,
|    "item_ID": 1012,
|    "message": "Sale not authorised by policy"
| } | Request to buy alcohol item (item_ID=1249) at 08:41:06 GMT on Monday, 02 January 2017. Sale is not authorized. Uses the MorningBoozeCheck task. Note this test is copied from Step 1 above, and demonstrates that the original MorningBoozeCheck task is executed. |
| {  
|    "nameSpace": "com.hyperm",
|    "name": "SALE_INPUT",
|    "version": "0.0.1",
|    "time": 1482398073000,
|    "sale_ID": 99999981,
|    "amount": 299,
|    "item_ID": 1047,
|    "quantity": 1,
|    "assistant_ID": 1212,
|    "branch_ID": 1002
| } | {  
|    "nameSpace": "com.hyperm",
|    "name": "SALE_AUTH",
|    "version": "0.0.1",
|    "source": "",
|    "target": "",
|    "sale_ID": 99999981,
|    "amount": 299,
|    "assistant_ID": 1212,
|    "notes": null,
|    "quantity": 1,
|    "branch_ID": 1002,
|    "item_ID": 1047,
|    "authorised": false,
|    "time": 1482398073000,
|    "message": "Sale not authorised by policy"
| } | Request to buy alcohol (item_ID=1047) at 10:14:33 on Thursday, 22 December 2016. Sale is not authorized. Uses the MorningBoozeCheckAlt1 task. Note this test is copied from Step 1 above, and demonstrates that the original MorningBoozeCheckAlt1 task is executed.
4.4. Policy 2 in CLI Editor

An equivalent version of the MyFirstPolicyModel policy model can again be generated using the APEX CLI editor. A sample APEX CLI script is shown below:

APEX CLI Editor code for Policy 2

```apex
#-------LICENSE_START-----------------------------
# Copyright (C) 2016-2018 Ericsson. All rights reserved.
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0
#-------LICENSE_END-----------------------------

```

Sample APEX CLI script:

```json
{
    "nameSpace": "com.hyperm",
    "name": "SALE_INPUT",
    "version": "0.0.1",
    "time": 1482077977000,
    "sale_ID": 99999982,
    "amount": 2199,
    "item_ID": 1443,
    "quantity": 12,
    "assistant_ID": 94,
    "branch_ID": 1003,
    "notes": "Buy 3, get 1 free!!"
}
```

Request to buy alcohol (item_ID=1443) at 17:19:37 on Sunday, 18 December 2016.

Sale is not authorized. Uses the MorningBoozeCheckAlt1 task.

```json
{
    "nameSpace": "com.hyperm",
    "name": "SALE_AUTH",
    "version": "0.0.1",
    "source": "",
    "target": "",
    "sale_ID": 99999982,
    "amount": 2199,
    "assistant_ID": 94,
    "notes": "Buy 3, get 1 free!!",
    "quantity": 12,
    "branch_ID": 1003,
    "item_ID": 1443,
    "authorised": false,
    "time": 1482077977000,
    "message": "Sale not authorised by policy task MorningBoozeCheckAlt1 for time 17:19:37 CET. Alcohol can not be sold between 00:00:00 CET and 13:00:00 CET or on Sunday"
}
```

Request to buy non-alcoholic item (item_ID=5321) at 11:13:09 on Monday, 2 January 2017.

Sale is authorized. Uses the MorningBoozeCheckAlt1 task.

```json
{
    "nameSpace": "com.hyperm",
    "name": "SALE_INPUT",
    "version": "0.0.1",
    "time": 1483351989000,
    "sale_ID": 99999983,
    "amount": 699,
    "item_ID": 5321,
    "quantity": 1,
    "assistant_ID": 2323,
    "branch_ID": 1001,
    "notes": ""
}
```

```json
{
    "nameSpace": "com.hyperm",
    "name": "SALE_AUTH",
    "version": "0.0.1",
    "source": "",
    "target": "",
    "sale_ID": 99999983,
    "amount": 699,
    "assistant_ID": 2323,
    "notes": "",
    "quantity": 1,
    "branch_ID": 1001,
    "item_ID": 5321,
    "authorised": true,
    "time": 1483351989000,
    "message": "Sale authorised by policy task MorningBoozeCheckAlt1 for time 11:13:09 CET"
}
```
model create name=MyFirstPolicyModel version=0.0.1 uuid=540226fb-55ee-4f0e-a444-983a0494818e
description="This is my first Apex Policy Model."

schema create name=assistant_ID_type version=0.0.1 uuid=36df4c71-9616-4206-8b53-976a5cd4bd87
description="A type for 'assistant_ID' values" flavour=Java schema=java.lang.Long

schema create name=authorised_type version=0.0.1 uuid=d48b619e-d00d-4008-b884-02d76ea4350b
description="A type for 'authorised' values" flavour=Java schema=java.lang.Boolean

schema create name=branch_ID_type version=0.0.1 uuid=6468845f-4122-4128-8e49-0f52c26078b5
description="A type for 'branch_ID' values" flavour=Java schema=java.lang.Long

schema create name=item_ID_type version=0.0.1 uuid=4f227ff1-aee0-453a-b6b6-9a4b2e0da932
description="A type for 'item_ID' values" flavour=Java schema=java.lang.Long

schema create name=message_type version=0.0.1 uuid=ad1431bb-3155-4e73-b5a3-b89bee498749
description="A type for 'message' values" flavour=Java schema=java.lang.String

schema create name=notes_type version=0.0.1 uuid=eeefde98-896c-4343-8f9c-2603ced94e2d
description="A type for 'notes' values" flavour=Java schema=java.lang.String

schema create name=price_type version=0.0.1 uuid=52c2cfc45-fd8c-463c-bd6f-d91b0554aea7
description="A type for 'amount'/'price' values" flavour=Java schema=java.lang.Long

schema create name=quantity_type version=0.0.1 uuid=ac3d9842-80af-4a98-951c-bd79a431c613
description="A type for 'quantity' values" flavour=Java schema=java.lang.Integer

schema create name=sale_ID_type version=0.0.1 uuid=cca47d74-7754-4a61-b163-ca31f66b157b
description="A type for 'sale_ID' values" flavour=Java schema=java.lang.Long

schema create name=timestamp_type version=0.0.1 uuid=fd594e88-411d-4a94-b2be-697b3a0d7adf
description="A type for 'time' values" flavour=Java schema=java.lang.Long

task create name=MorningBoozeCheck version=0.0.1 uuid=3351b0f4-cf06-4fa2-8823-edf67bd30223
description=LS

This task checks if the sales request is for an item that contains alcohol. If the local time is between 00:00:00 and 11:30:00 then the sale is not authorised. Otherwise the sale is authorised.

In this implementation we assume that all items with item_ID values between 1000 and 2000 contain alcohol."

LE

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=sale_ID schemaName=sale_ID_type
schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=amount schemaName=price_type
schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=assistant_ID
schemaName=assistant_ID_type schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=notes schemaName=notes_type
schemaVersion=0.0.1 optional=true

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=quantity schemaName=quantity_type
schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=branch_ID schemaName=branch_ID_type
schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=item_ID schemaName=item_ID_type
schemaVersion=0.0.1

task inputfield create name=MorningBoozeCheck version=0.0.1 fieldName=time schemaName=timestamp_type
task outputfield create name=MorningBoozeCheck version=0.0.1 fieldName=sale_ID schemaName=sale_ID_type

// The events used later to test this task use GMT timezone!
gmt = TimeZone.getTimeZone("GMT");
timenow = Calendar.getInstance(gmt);
df = new SimpleDateFormat("HH:mm:ss z");
df.setTimeZone(gmt);

import java.util.Date;
import java.util.Calendar;
import java.util.TimeZone;
import java.text.SimpleDateFormat;

logger.info("Task Execution: "+subject.id+". Input Fields: "+inFields+" ");

outFields.put("amount" , inFields.get("amount"));
outFields.put("assistant_ID", inFields.get("assistant_ID"));
outFields.put("notes"    , inFields.get("notes"));
outFields.put("quantity" , inFields.get("quantity"));
outFields.put("branch_ID", inFields.get("branch_ID"));
outFields.put("item_ID"  , inFields.get("item_ID"));
outFields.put("time"     , inFields.get("time"));
outFields.put("sale_ID"  , inFields.get("sale_ID"));

// The events used later to test this task use GMT timezone!
timenow.setTimeInMillis(inFields.get("time"));

midnight = timenow.clone();
midnight.set(
    timenow.get(Calendar.YEAR), timenow.get(Calendar.MONTH),
    timenow.get(Calendar.DATE), 0, 0, 0);
eleven30 = timenow.clone();
eleven30.set(
    timenow.get(Calendar.YEAR), timenow.get(Calendar.MONTH),
    timenow.get(Calendar.DATE), 11, 30, 0);

itemisalcohol = false;
if (item_id != null && item_id >= 1000 && item_id < 2000)
    itemisalcohol = true;

if (itemisalcohol
    && timenow.after(midnight) && timenow.before(eleven30)) {
    outFields.put("authorised", false);
    outFields.put("message", "Sale not authorised by policy task "+subject.taskName+
    
    for time "+df.format(timenow.getTime())+
    
    Alcohol cannot be sold between "+df.format(midnight.getTime())+
    
    and "+df.format(eleven30.getTime());
    return true;
}
else {
    outFields.put("authorised", true);
    outFields.put("message", "Sale authorised by policy task "+subject.taskName+
    
    for time "+df.format(timenow.getTime()));
    return true;
}

/*
This task checks if a sale request is for an item that is an alcoholic drink.
If the local time is between 00:00:00 GMT and 11:30:00 GMT then the sale is not
authorised. Otherwise the sale is authorised.
In this implementation we assume that items with item_ID value between 1000 and
2000 are all alcoholic drinks :-)*/
LE

task create name=MorningBoozeCheckAlt1 version=0.0.1 uuid=bc6d90c9-c902-4686-afd3-925b30e39990
description=This task checks if a sale request is for an item that is an alcoholic drink.
If the local time is between 00:00:00 CET and 13:00:00 CET then the sale is not authorised.
Also alcohol sales are not allowed on Sundays. Otherwise the sale is authorised.
In this implementation we assume that items with item_ID between 1000 and 2000 are all alcoholic drinks
LE
task inputfield create name=MorningBoozeCheckAlt1 version=0.0.1 fieldName=sale_ID schemaName=sale_ID_type
schemaVersion=0.0.1
task inputfield create name=MorningBoozeCheckAlt1 version=0.0.1 fieldName=amount schemaName=price_type
schemaVersion=0.0.1
task inputfield create name=MorningBoozeCheckAlt1 version=0.0.1 fieldName=assistant_ID
schemaName=assistant_ID_type schemaVersion=0.0.1
if optional=true
task inputfield create name=MorningBoozeCheckAlt1 version=0.0.1 fieldName=notes schemaName=notes_type
schemaVersion=0.0.1 optional=true
task inputfield create name=MorningBoozeCheckAlt1 version=0.0.1 fieldName=quantity
schemaName=quantity_type schemaVersion=0.0.1
task inputfield create name=MorningBoozeCheckAlt1 version=0.0.1 fieldName=branch_ID
schemaName=branch_ID_type schemaVersion=0.0.1
if optional=true
task inputfield create name=MorningBoozeCheckAlt1 version=0.0.1 fieldName=item_ID schemaName=item_ID_type
schemaVersion=0.0.1
task inputfield create name=MorningBoozeCheckAlt1 version=0.0.1 fieldName=time
schemaName=timestamp_type schemaVersion=0.0.1
if optional=true
task outputfield create name=MorningBoozeCheckAlt1 version=0.0.1 fieldName=sale_ID schemaName=sale_ID_type
import java.util.Date;
import java.util.Calendar;
import java.util.TimeZone;
import java.text.SimpleDateFormat;

logger.info("Task Execution: "+subject.id+". Input Event: "+inFields+";");

outFields.put("amount" , inFields.get("amount"));
outFields.put("assistant_ID", inFields.get("assistant_ID"));
outFields.put("notes"       , inFields.get("notes"));
outFields.put("quantity"    , inFields.get("quantity"));
outFields.put("branch_ID"   , inFields.get("branch_ID"));
outFields.put("item_ID"     , inFields.get("item_ID"));
outFields.put("time"        , inFields.get("time"));
outFields.put("sale_ID"     , inFields.get("sale_ID"));

item_id = inFields.get("item_ID");

//The events used later to test this task use CET timezone!
cet = TimeZone.getTimeZone("CET");
timenow = Calendar.getInstance(cet);
df = new SimpleDateFormat("HH:mm:ss z");
df.setTimeZone(cet);
timenow.setTimeInMillis(inFields.get("time"));
midnight = timenow.clone();
midnight.set(
    timenow.get(\Calendar.\YEAR),timenow.get(\Calendar.\MONTH),
    timenow.get(\Calendar.\DATE),0,0,0);
onepm = timenow.clone();
onepm.set(
    timenow.get(\Calendar.\YEAR),timenow.get(\Calendar.\MONTH),
    timenow.get(\Calendar.\DATE),13,0,0);

itemisalcohol = false;
if(item_id != null && item_id >=1000 && item_id < 2000)
    itemisalcohol = true;

if( itemisalcohol &&
    ( (timenow.after(midnight) && timenow.before(onepm))
    ||
    (timenow.get(\Calendar.\DAY_OF\WEEK) == Calendar.SUNDAY)
    )
)
    outFields.put("authorised", false);
    outFields.put("message", "Sale not authorised by policy task "+subject.taskName+
    " for time "+df.format(timenow.get(Time()))+
    ". Alcohol can not be sold between "+df.format(midnight.get(Time()))+
    " and "+df.format(onepm.get(Time())) +" or on Sunday");
    return true;
else{
    outFields.put("authorised", true);
    outFields.put("message", "Sale authorised by policy task "+subject.taskName+
    " for time "+df.format(timenow.get(Time()));
    return true;
}

/*
This task checks if a sale request is for an item that is an alcoholic drink. 
If the local time is between 00:00:00 CET and 13:00:00 CET then the sale is not authorised. 
Also alcohol sales are not allowed on Sundays. Otherwise the sale is authorised. 
In this implementation we assume that items with item_ID between 1000 and 2000 are all alcoholic drinks :-)
*/
*/

LE

event create name=SALE_AUTH version=0.0.1 uuid=c45b0941-3f98-4080-a9cc-5b9753ed050b description="An event emitted by the Policy to indicate whether the sale of an item has been authorised" nameSpace=com.hyperm
    source="APEX" target="POS"
    parameter create name=SALE_AUTH version=0.0.1 parName=amount schemaName=price_type
    schemaVersion=0.0.1
    parameter create name=SALE_AUTH version=0.0.1 parName=assistant_ID schemaName=assistant_ID_type
    schemaVersion=0.0.1
    parameter create name=SALE_AUTH version=0.0.1 parName=authorised schemaName=authorised_type
    schemaVersion=0.0.1
    parameter create name=SALE_AUTH version=0.0.1 parName=branch_ID schemaName=branch_ID_type
    schemaVersion=0.0.1
    parameter create name=SALE_AUTH version=0.0.1 parName=item_ID schemaName=item_ID_type
    schemaVersion=0.0.1
    parameter create name=SALE_AUTH version=0.0.1 parName=message schemaName=message_type
    schemaVersion=0.0.1 optional=true
    parameter create name=SALE_AUTH version=0.0.1 parName=notes schemaName=notes_type
    schemaVersion=0.0.1 optional=true
    parameter create name=SALE_AUTH version=0.0.1 parName=quantity schemaName=quantity_type
    schemaVersion=0.0.1
    parameter create name=SALE_AUTH version=0.0.1 parName=sale_ID schemaName=sale_ID_type
    schemaVersion=0.0.1
    parameter create name=SALE_AUTH version=0.0.1 parName=time schemaName=timestamp_type
    schemaVersion=0.0.1
event create name=SALE_INPUT version=0.0.1 uuid=4f04aa98-e917-4f4a-a82a-c75ba5a99374 description="An event raised by the PoS system each time an item is scanned for purchase" nameSpace=com.hyperm source="POS" target="APEX"

event parameter create name=SALE_INPUT version=0.0.1 parName=amount schemaName=price_type schemaVersion=0.0.1

event parameter create name=SALE_INPUT version=0.0.1 parName=assistant_ID schemaName=assistant_ID_type schemaVersion=0.0.1

event parameter create name=SALE_INPUT version=0.0.1 parName=branch_ID schemaName=branch_ID_type schemaVersion=0.0.1

event parameter create name=SALE_INPUT version=0.0.1 parName=item_ID schemaName=item_ID_type schemaVersion=0.0.1

event parameter create name=SALE_INPUT version=0.0.1 parName=notes schemaName=notes_type schemaVersion=0.0.1 optional=true

event parameter create name=SALE_INPUT version=0.0.1 parName=quantity schemaName=quantity_type schemaVersion=0.0.1

event parameter create name=SALE_INPUT version=0.0.1 parName=sale_ID schemaName=sale_ID_type schemaVersion=0.0.1

event parameter create name=SALE_INPUT version=0.0.1 parName=time schemaName=timestamp_type schemaVersion=0.0.1

policy create name=MyFirstPolicy version=0.0.1 uuid=6c5e410f-489a-46ff-964e-982ce6e8b6d0 description="This is my first Apex policy. It checks if a sale should be authorised or not." template=FREEFORM
template=FREEFORM

firstState=BoozeAuthDecide

policy state create name=MyFirstPolicy version=0.0.1 stateName=BoozeAuthDecide triggerName=SALE_INPUT triggerVersion=0.0.1 defaultTaskName=MorningBoozeCheck defaultTaskVersion=0.0.1

policy state output create name=MyFirstPolicy version=0.0.1 stateName=BoozeAuthDecide outputName=MorningBoozeCheck_Output_Direct eventName=SALE_AUTH eventVersion=0.0.1 nextState=NULL

taskLocalName=MorningBoozeCheckAlt1 taskName=MorningBoozeCheckAlt1 taskVersion=0.0.1 outputType=DIRECT outputName=MorningBoozeCheck_Output_Direct

policy state taskref create name=MyFirstPolicy version=0.0.1 stateName=BoozeAuthDecide taskLocalName=MorningBoozeCheck taskName=MorningBoozeCheck taskVersion=0.0.1 outputType=DIRECT outputName=MorningBoozeCheck_Output_Direct

policy state selecttasklogic create name=MyFirstPolicy version=0.0.1 stateName=BoozeAuthDecide logicFlavour=JAVASCRIPT logic=LS

var returnValueType = Java.type("java.lang.Boolean");

var returnValue = new returnValueType(true);

executor.logger.info("Task Selection Execution: "+executor.subject.id+". Input Event: "+executor.inFields+". Output: "+returnValue);

branchid = executor.inFields.get("branch_ID");
taskorig = executor.subject.getTaskKey("MorningBoozeCheck");
taskalt = executor.subject.getTaskKey("MorningBoozeCheckAlt1");
taskdef = executor.subject.getDefaultTaskKey();

if(branchid >= 0 && branchid < 1000){
    taskorig.copyTo(executor.selectedTask);
}
else if (branchid >= 1000 && branchid < 2000){
    taskalt.copyTo(executor.selectedTask);
}
else{
    taskdef.copyTo(executor.selectedTask);
}

/*
This task selection logic selects task "MorningBoozeCheck" for branches with 0<=branch_id<1000 and selects task "MorningBoozeCheckAlt1" for branches with 1000<=branch_id<2000. Otherwise the default task is selected. In this case the default task is also "MorningBoozeCheck"
*/