**1.0 Overview**

**Drools** is a Business Logic Integration platform (BLiP). It is written in Java. It is open source and backed by RedHat and JBoss. It is a Business Rules Management System (BRMS) solution. It provides a core Business Rules Engine (BRE), web authoring, and rules management application (DROOLS Workbench) and an Eclipse IDE plugin for core development. It is one of the best open source rule engines providing Complex Event Processing.

**Facts** are stored in working memory (like an in-memory database). Fact is a piece of data, such as **salary=3000** or, in an object-oriented rules engine, it may be a Java object with properties.

**Rules** are defined in a **knowledge base**. A **rule** consists of conditions (which typically depend on facts in the working memory) and actions which are executed when the conditions are true (similar to an "if-then" statement).

**2.0 Advantages of the Drools Rules Engine**

* **Reusability**: The rules are kept at a centralized location (separation of business logic from the rest of the system); it is easy to reuse the Drools rules engine.
* **Flexibility:** Changing the application or its model is not an easy task, but it is easier to change the rules than Java code.
* **Redeploying**: It is easy to redeploy modified rules without stopping or modifying the application logic.
* **Easier to Understand**: It is easier to understand the rules for new developers than code written in Java or any other languages.
* You can avoid a lot of if-else from your actual business logic by separating or implementing business rules separately.
* **Unifying:** The Drools platform brings unification of rules and processes. It is easy to call rules from a process or vice versa.
* Easy to implement complex logic using the Drools Rules Engine.

**3.0 Considerations for Using the Drools Rule Engine**

Let consider that Drools should be used when your business logic is very complex. Apart from complex business logic, you can use Drools in the below situations:

* Application business logic often changes.
* The business logic contains a lot of **If-Else** statements.
* The business logic is overly complex or time-consuming when defined procedurally.
* The business logic needs to be maintained by people who don’t (or shouldn’t) have access to the application itself (to recompile/redeploy it).
* Domain experts (or business analysts) are readily available, but are nontechnical.

**4.0 Business Use Cases**

Today we will implement a business use case such as a life insurance policy; depending on age and policy type, we will be providing a premium amount. Below is the logic for our business case:

* If age < 10 and policyType = "Policy1", then premiumAmount = 20000.
* If age is between 10 and 40 and policyType = "Policy1", then premiumAmount = 30000.
* If age > 40 and policyType = "Policy1", then premiumAmount = 40000.
* If age < 10 and policyType = "Policy2", then premiumAmount = 21000.
* If age is between 10 and 40 and policyType = "Policy2", then premiumAmount = 31000.
* If age > 40 and policyType = "Policy2", then premiumAmount = 41000.

Our flow will receive the below JSON message and we will use JSON data to pass through our rules engine.



Once the JSON message has passed through the rules engine and our expected output is shown below:



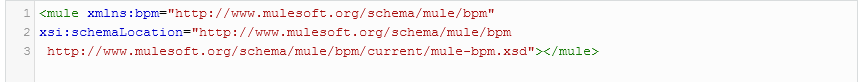
## 5.0 Implementing Drools With Mule ESB

Now, we will walk through how to integrate the Mule ESB and Java rules engine, namely Drools.

Create a Mule project and name the project droolsexample.xml; you can select Mule runtime 3.6 or higher.

## 5.1 Namespace and Schema

The first thing we need to add is the namespace and schema location to the Mule configuration file (e.g. droolsexample.xml).



## 5.2 Initialize BPM Rules

Add the below tag in droolsdemo.xml and it will tell the Mule flow there are rules defined in your flow and it will start Drool's working memory.

**Command :** <bpm:drools/>

## 5.3 Defining Mule Flow with Drools

We will be receiving the JSON message and it will be converted into a Java object using the JSON To Object transformer. Our Java class for Insurance will look like this:

Example :

package com.drools.example;

public class Insurance {

private String name;

private int age;

private String policyType;

private int premiumAmount;

public String getName()

{

return name;

}

public void setName(String name)

{

this.name=name;

}

public int getAge()

{

return age;

}

public void setAge(int age)

{

this.age=age;

}

public String getPolicyType()

{

return policyType;

}

public void setPolicyType(String policyType)

{

this.policyType=policyType;

}

public int getPremiumAmount()

{

return premiumAmount;

}

public void setPremiumAmount(int premiumAmount)

{

this.premiumAmount=premiumAmount;

}

}

Add file with .drl extension to your classpath (**src/main/resources**). This file will contain all rules in Plain Text, which can be easily modified and loaded at runtime if needed.

package com.drools.example;

import org.mule.MessageExchangePattern;

import com.drools.example.Insurance;

global org.mule.module.bpm.MessageService mule;

dialect "mvel"

declare Insurance

@role('event')

end

rule "Policy 1 And Infant"

lock-on-active when

$insurance: Insurance(age < 10 && policyType == "Policy1") then

modify($insurance) {

setPremiumAmount(20000)

}

end

rule "Policy 1 And Adult"

lock-on-active when

$insurance: Insurance((age >= 10 && age <= 40) && policyType == "Policy1") then

modify($insurance) {

setPremiumAmount(30000)

}

end

rule "Policy 1 And Older"

lock-on-active when

$insurance: Insurance(age > 40 && policyType == "Policy1") then

modify($insurance) {

setPremiumAmount(40000)

}

end

rule "Policy 2 And Infant"

lock-on-active when

$insurance: Insurance(age < 10 && policyType == "Policy2") then

modify($insurance) {

setPremiumAmount(21000)

}

end

rule "Policy 2 And Adult"

lock-on-active when

$insurance: Insurance((age >= 10 && age <= 40) && policyType == "Policy2") then

modify($insurance) {

setPremiumAmount(31000)

}

end

rule "Policy 2 And Older"

lock-on-active when

$insurance: Insurance(age > 40 && policyType == "Policy2") then

modify($insurance) {

setPremiumAmount(41000)

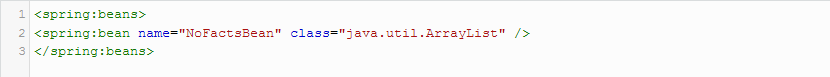
}

end

In the above rule file, we have defined import namespaces and packages statements. Then, we create the Insurance object to be used by our rule engine and followed by some rules with conditions that explained above.

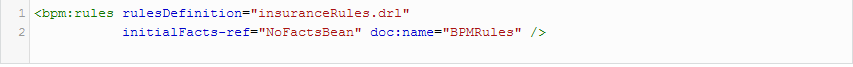
## 5.4 Defining Spring Beans

You need to define spring beans in your configuration file  (e.g. droolsexample.xml) as shown below



## 5.5 Call Rule Definition in Mule Flow

Rule defintions can be called in Mule Flow by defining the below statement in the configuration file (droolsexample.xml):



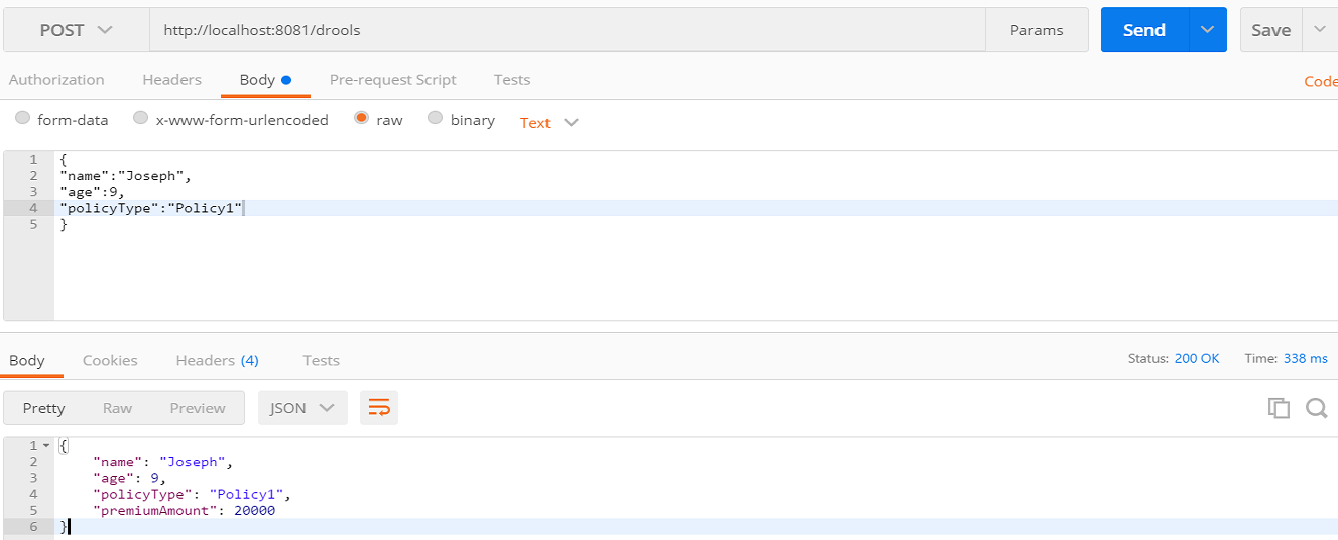
## 5.6 Mule Flow Code

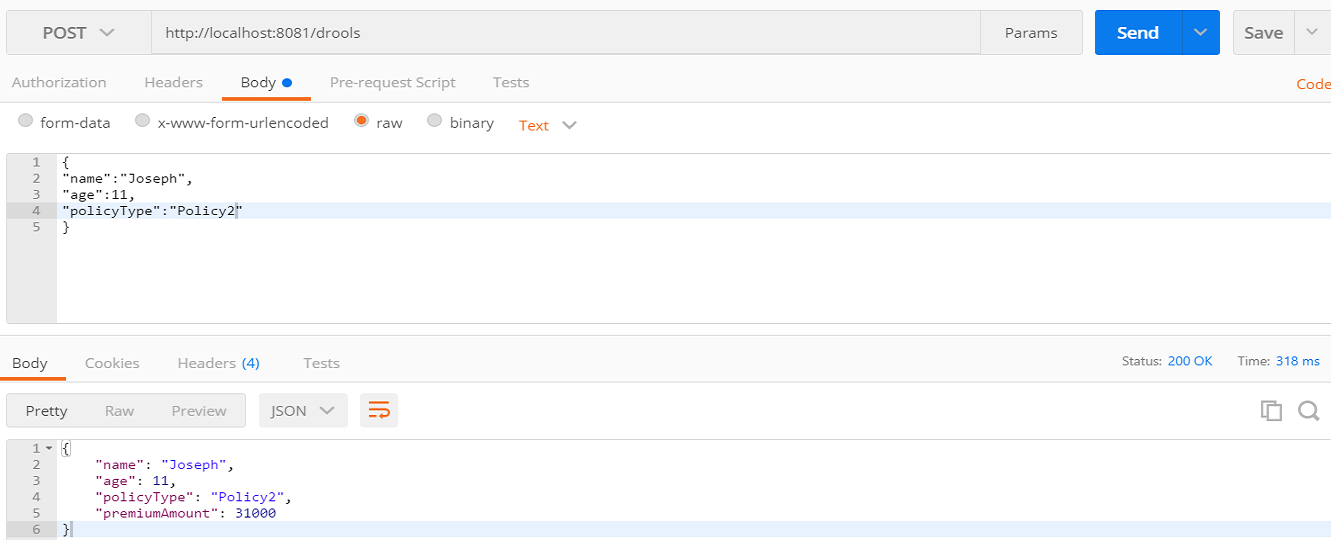
## 

## 

## 6.0 Testing

We will use Postman to test all the above scenarios, and hopefully, we will get the expected result back.





## Image title