Tutorial: Using the Determinations Engine API

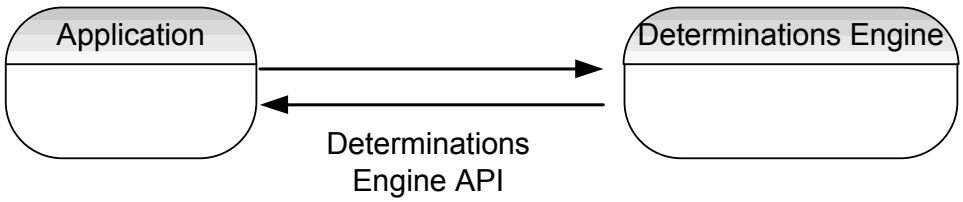
Author: Frank Hampshire
Last Updated: 25 November 2009

Introduction

The Determinations Engine provides a simple API (Application Programming Interface) so that it can be used, within an application. By direct access to the Determinations Engine, a systems integrator can use the API to access and use rules written in Oracle Policy Modeling directly in their application.

The Determinations Engine API is designed for direct, easy integration from an existing Java or C# .NET application. The direct nature of the API makes it simple as well as being fast and efficient.

This tutorial is a quick walkthrough the Java API, and will demonstrate how to integrate a rulebase (SimpleBenefits) with a very simple command line application.



Requirements to follow the tutorial

This tutorial discusses programming in Java. You should be familiar with java, and be able to set up and run a program in an IDE of your choice (Eclipse or Netbeans for example).

The following Software is required to follow this tutorial.

- A Java development environment/IDE with the ability to compile and run Java 1.5 (or later) code
- Oracle Policy Modeling 10.0
- Oralce Policy Automation Runtime - Java

Documentation on the Determinations Engine API

Full documentation for the Determinations Engine is available in the *Oracle Policy Automation Runtime - Java* Help.

1 Build the SimpleBenefits rulebase

The first thing to do is to have a look at the SimpleBenefits rulebase in Oracle Policy Modeling. This is a very simple example rulebase which determines 3 goals:

```
Is the claimant eligible for the low income allowance?
What is the claimant's low income allowance amount?
Is the claimant eligible for the teenage child allowance?
```

These goals are attributes of the global entity. Eligibility for the low income allowance is based on the information on the global attribute only. Eligibility for the teenage child allowance is based on the claimant's children and their age.

To build this rulebase, choose *Build* from the Build menu. Once this is done, you can find the rulebase archive file, SimpleBenefits.zip, in the output directory. This is the file that the Determinations Engine will use.

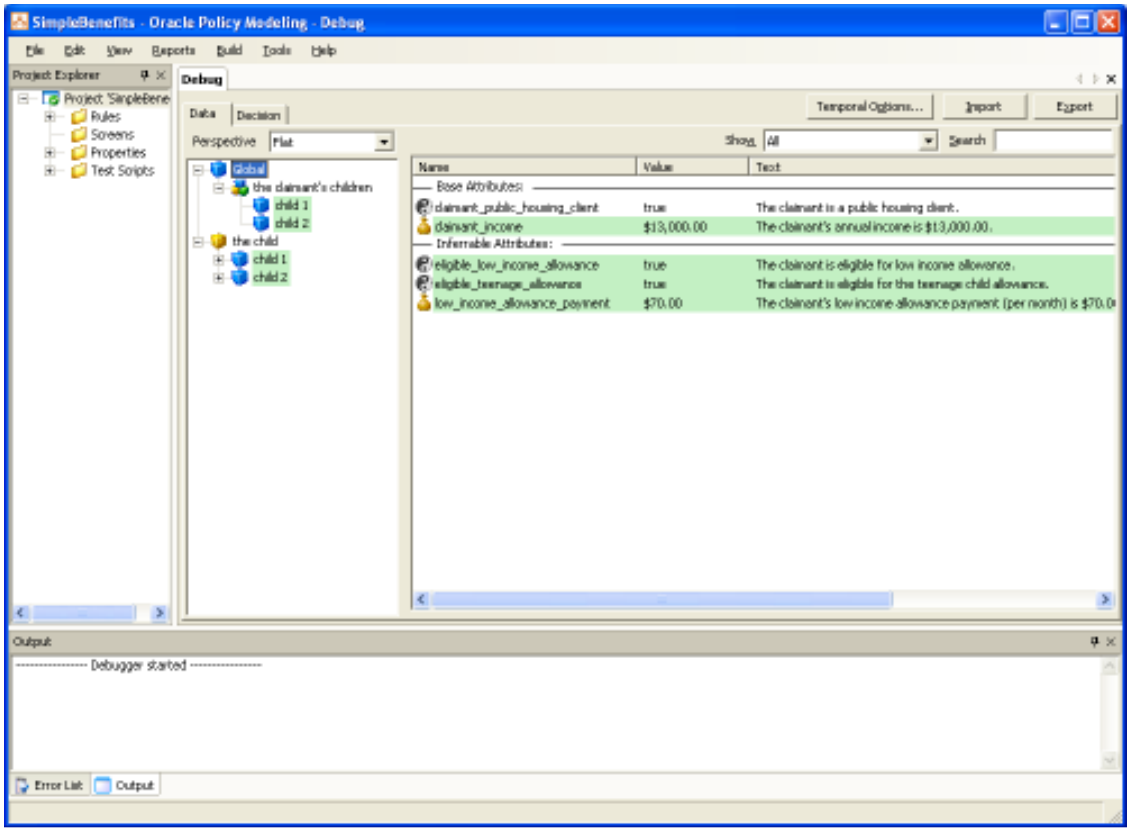
You can use the Debugger in Oracle Policy Modeling to test this rulebase. If you want to reproduce the session that we will use in the tutorial, you can do it by these steps:

1. Choose *Build and Debug* from the Build Menu
2. Choose *Without Screens* as the Debug Mode. The Debugger should start
3. Set *the claimant is a public housing client* To true (global entity)
4. Set *the claimant's annual income* to 13,000 on the (global entity)

5. Add two child entities and add them as targets to the relationship *the claimants children*
6. Set *the child's age* attribute on each child to 16 (child 1) and 8 (child 2)

If you look at the inferred attributes you should be able to see the following values:

The claimant is eligible for low income allowance (true)
The claimant is eligible for the teenage child allowance (true)
The claimant's low income allowance payment (per month) = $70.00



This is essentially what we are going to do via the Engine API, but instead of using the Debugger, we will do this directly in Java.

2 Write a program to use the Determinations Engine

Once we have built the rulebase, we can create a program that uses it. The program in this tutorial is a simple command line application that creates a Determinations Engine session, setting information about a claimant and his/her children using the API. The programs prints the eligibility information for the claimant to the screen

The program in this tutorial will consist of a single class with a *main* method.

There are several steps to writing a program that uses the Determinations Engine API

2.1 Add the Determinations Engine library, and third party libraries to the project's classpath

There are several jar files needed to run the Determinations Engine. These represent the Determinations Engine itself, and the third party libraries that the Determinations Engine depends on. They are all included in the OPA Runtime Java Engine. The jar files for 10.0 are as follows:

determinations-engine.jar
jdom-1.0.jar
jsr173_api.jar
log4j-1.2.15.jar
sjsxp.jar

2.2 Import the generated java client code

To use the Engine API in a program, you must import it. A single line import is all that is needed to include the engine API.

```
import com.oracle.determinations.engine.*;
```

2.3 Create the Engine and load the rulebase

Because the program will execute through the *main* method of our single class, all the code to uses the engine will go there.

The first thing you need to do is create an instance of the Engine. The engine represents the whole runtime and is responsible for managing rulebase loading and management and also session creation.

Once we have an instance of the engine we need to load the SimpleBenefits rulebase. In the code below you can see the path to the SimpleBenefits rulebase archive.

```
Engine engine = EngineFactory.localEngine();

Rulebase simpleBenefits = engine.getRulebase(
    "C:/SimpleBenefits/Rulebase/SimpleBenefits/"
    +"Development/output/SimpleBenefits.zip");
```

Note: Once the archive is built it can be put anywhere and does not need to remain in the project's output directory. The Determinations Engine, Web Determinations and the Determinations Server all use this rulebase archive.

2.4 Create the session, entities, attributes and relationships that will be needed

The next step is to create a session which will hold all the information on the claimant. The engine is responsible for creating and managing sessions. A session holds the data in the form of attributes, entities and relationships. When you run the Debugger or Web Determinations from Oracle Policy Modeling, you are creating and adding data to a *Session*.

In this case, we will create a simple unmanaged and unsynchronized session. A managed session is stored in the engine, and will remain in the engine instance until it is explicitly deleted. Unmanaged sessions are not stored in the engine. A synchronized session inferences (resolves inferred attributes and relationships) every time its data changes. With an unsynchronized session inferencing is not done until the public `think()` method is called on the session.

```
Session session = engine.createSession(simpleBenefits,
        SessionManagementMode.UNMANAGED,
        SessionSynchronizationMode.UNSYNCHRONIZED);
```

We also need to get the entities, attributes and relationships that we are going to use. A new session always comes with an already created global entity instance. This is the only thing created automatically in a new session. The *child* entity, which is the other entity that we will use, can be got from the rulebase object.

```
EntityInstance global = session.getGlobalEntityInstance();
Entity globalEntity = global.getEntity();
Entity child = simpleBenefits.getEntity("child");
EntityInstance child1 = session.createEntityInstance(child);
EntityInstance child2 = session.createEntityInstance(child);
```

Once we have got the entities and created the instances, we can get any attributes and relationships we need.

```
Attribute claimant_income = globalEntity
    .getAttribute("claimant_income");
Attribute claimant_public_housing_client = globalEntity
    .getAttribute("claimant_public_housing_client");
Relationship claimantschildren = globalEntity
    .getRelationship("claimantschildren");

Attribute child_age = child.getAttribute("child_age");
```

Entities and Entity Instances

The difference between an entity and an entity instance in the Engine API can be a little confusing so it is worth spending a little time to understand them. An entity is a template

for an object in a rulebase session. The Entity has information about the attributes and relationships that object will have. An Entity Instance is an instantiation of the entity, or an actual object. So, for the child entity, we can create as many instances as we need. Each instance created will have its own attributes and be able to participate as source or target in relationships.

In programming terms you could consider an Entity to be a *class*, and an EntityInstance to be an *object*.

In the SimpleBenefits rulebase the child entity has a single attribute *the child's age.* We can create as many instances of the child as we need, in this case. We will create two child entities (children of the claimant).

```
EntityInstance child1 = session.createEntityInstance(child);
EntityInstance child2 = session.createEntityInstance(child);
```

2.5 Set the values of attributes and relationships

The next step is to set the attribute values that we will need to inference the results that we want. We do this by calling the `setValue` method on the Attribute interface and the `setInstance` method on the Relationship interface

We will set the income to 13000 and the public housing client to true.

```
claimant_income.setValue(global, 13000.00);
claimant_public_housing_client.setValue(global, true);
```

For each child, we will set their names to 16 and 8 respectively

```
child_age.setValue(child1, 16);
child_age.setValue(child2, 8);
```

Now we need to link the child instances to the global (claimant), by setting an instance of the `claimantschildren` relationship.

```
claimantschildren.setInstance(global,
    Arrays.asList(new EntityInstance[] {child1, child2}));
```

2.6 Think and check the results

Once we have set all the information to the session, we can tell it to think. When the think method is called all the rules that use data that has changed will fire and set new values for any inferred attributes or relationships. Because this is the first time that think has been called all the data counts as being changed.

The three values that we want to get answers for are also attributes which we can get from the global entity. Although, in this example, we are getting these attributes after the think method has been called on the session, we could have got the attributes at the beginning. With no data, the initial values for the attributes would be unknown but, after the think operation, the values would be updated to match any new inferred values.

```
session.think();

// outcomes
Attribute eligible_low_income_allowance =
    globalEntity.getAttribute("eligible_low_income_allowance");
Attribute low_income_allowance_payment =
    globalEntity.getAttribute("low_income_allowance_payment");
Attribute eligible_teenage_allowance =
globalEntity.getAttribute("eligible_teenage_allowance");

// print out the results
System.out.println("\n--- Results ----");

if (eligible_low_income_allowance.isUnknown(global)) {
    System.out.println("eligible_low_income_allowance is unknown");
}
else if (eligible_low_income_allowance.isUncertain(global)) {
    System.out.println("eligible_low_income_allowance is uncertain");
}
else  {
    System.out.println("eligible_low_income_allowance = "
        +eligible_low_income_allowance.getValue(global));
}

if (low_income_allowance_payment.isUnknown(global)) {
    System.out.println("low_income_allowance_payment is unknown");
}
else if (low_income_allowance_payment.isUncertain(global)) {
    System.out.println("low_income_allowance_payment is uncertain");
}
else  {
    System.out.println("low_income_allowance_payment = "
        +low_income_allowance_payment.getValue(global));
}

if (eligible_teenage_allowance.isUnknown(global)) {
    System.out.println("eligible_teenage_allowance is unknown");
}
else if (eligible_teenage_allowance.isUncertain(global)) {
    System.out.println("eligible_teenage_allowance is uncertain");
}
else  {
    System.out.println("eligible_teenage_allowance = "
        +eligible_teenage_allowance.getValue(global));
}
```

2.7 Run the program

When you run the program, you should get the following output printed to standard out. From the response, we can see that all outcomes are known and that the claimant is eligible for both allowances and that the low income allowance payment is 70.0.

```
--- Creating local engine instance ---
Loading simple benefits rulebase
Creating a session
Setting claimant_income to 13000.00
Setting claimant_public_housing_client to true
Setting child_age on child1 to 16
Setting child_age on child2 to 8
Adding child1 and child2 to 'claimantschildren' relationship

--- Results ----
eligible_low_income_allowance = true
low_income_allowance_payment = 70.0
eligible_teenage_allowance = true
```

Appendix 1 - SimpleBenefitsEngineTest Class

```
import java.util.Arrays;
import com.oracle.determinations.engine.*;

public class SimpleBenefitsEngineTest
{


    public static void main(String[] args) {


        try {

            System.out.println("--- Creating local engine instance ---");
            Engine engine = EngineFactory.localEngine();

            System.out.println("Loading simple benefits rulebase");
            Rulebase simpleBenefits = engine.getRulebase(
                "C:/SimpleBenefits/Rulebase/SimpleBenefits/"
                +"Development/output/SimpleBenefits.zip");

            System.out.println("Creating a session");

            Session session = engine.createSession(simpleBenefits,
                SessionManagementMode.UNMANAGED,
                SessionSynchronizationMode.UNSYNCHRONIZED);

            // get the attributes entities that we are going to use
            // to set the values for assessing the claimants eligibility

            // we can just get the global entity instance, as every session
            // always has one
            EntityInstance global = session.getGlobalEntityInstance();
            Entity globalEntity = global.getEntity();
```

```
Entity child = simpleBenefits.getEntity("child");

Attribute claimant_income = globalEntity.getAttribute("claimant_income");
Attribute claimant_public_housing_client = globalEntity
    .getAttribute("claimant_public_housing_client");
Relationship claimantschildren = globalEntity
    .getRelationship("claimantschildren");

Attribute child_age = child.getAttribute("child_age");


System.out.println("Setting claimant_income to 13000.00");
claimant_income.setValue(global, 13000.00);

System.out.println("Setting claimant_public_housing_client to true");
claimant_public_housing_client.setValue(global, true);

// unlike the global entity instance, there can be many instances
//of the child entity. In this case, we will create two child
// instances.
EntityInstance child1 = session.createEntityInstance(child);
EntityInstance child2 = session.createEntityInstance(child);

// set the age for each child using the child_age attribute.
System.out.println("Setting child_age on child1 to 16");
child_age.setValue(child1, 16);
System.out.println("Setting child_age on child2 to 8");
child_age.setValue(child2, 8);

// this creates a relationship with the global as a source and
// child1 and child2 as targets. Because the claimantschildren
// relationship is a 1-to-many, the reverse relationship from each
// to the source will also be created.
System.out.println("Adding child1 and child2 to 'claimantschildren' relationship");
claimantschildren.setInstance(global,
    Arrays.asList(new EntityInstance[] {child1, child2}));

// This will fire any rule that relies on data that has changed.
```

```
// Because this is the first time it has been called, that should
// be everything.
session.think();

// outcomes
Attribute eligible_low_income_allowance = globalEntity
    .getAttribute("eligible_low_income_allowance");
Attribute low_income_allowance_payment = globalEntity
    .getAttribute("low_income_allowance_payment");
Attribute eligible_teenage_allowance = globalEntity
    .getAttribute("eligible_teenage_allowance");


// print out the results
System.out.println("\n--- Results ----");

if (eligible_low_income_allowance.isUnknown(global)) {
    System.out.println("eligible_low_income_allowance is unknown");
}
else if (eligible_low_income_allowance.isUncertain(global)) {
    System.out.println("eligible_low_income_allowance is uncertain");
}
else  {
    System.out.println("eligible_low_income_allowance = "
        +eligible_low_income_allowance.getValue(global));
}

if (low_income_allowance_payment.isUnknown(global)) {
    System.out.println("low_income_allowance_payment is unknown");
}
else if (low_income_allowance_payment.isUncertain(global)) {
    System.out.println("low_income_allowance_payment is uncertain");
}
else  {
    System.out.println("low_income_allowance_payment = "
        +low_income_allowance_payment.getValue(global));
}
```

```
            if (eligible_teenage_allowance.isUnknown(global)) {
                System.out.println("eligible_teenage_allowance is unknown");
            }
            else if (eligible_teenage_allowance.isUncertain(global)) {
                System.out.println("eligible_teenage_allowance is uncertain");
            }
            else  {
                System.out.println("eligible_teenage_allowance = "
                    +eligible_teenage_allowance.getValue(global));
            }

        }
        catch(Exception e) {
            System.out.println("An error occurred"+ e.getMessage());
            e.printStackTrace();
        }
    }
}
```