# Introduction to Drools

Last updated: May 11, 2024

Written by: baeldung (https://www.baeldung.com/author/baeldung)

Reviewed by: Michal Aibin (https://www.baeldung.com/editor/michal-author)

**Java IO (https://www.baeldung.com/category/java/java-io)**

**Drools (https://www.baeldung.com/tag/drools)** **reference**

**Rule Engine (https://www.baeldung.com/tag/rule-engine)**

## 1. Overview

Drools (https://www.drools.org/) is a Business Rule Management System (BRMS) solution. It provides a rule engine which processes facts and produces output as a result of rules and facts processing. Centralization of business logic makes it possible to introduce changes fast and cheap.

It also bridges the gap between the Business and Technical teams by providing a facility for writing the rules in a format which is easy to understand.

## 2. Maven Dependencies

To get started with Drools, we need to first add a couple of dependencies in our *pom.xml*:

```
<dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-ci</artifactId>
    <version>9.44.0.Final</version>
</dependency>
<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-decisiontables</artifactId>
    <version>9.44.0.Final</version>
</dependency>
```

The latest version of both dependencies is available on Maven Central Repository as kie-ci (https://mvnrepository.com/artifact/org.kie/kie-ci) and drools-decisiontables (https://mvnrepository.com/artifact/org.drools/drools-decisiontables).

## 3. Drools Basics

We are going to look at basic concepts of Drools:

- ***Facts*** – represents data that serves as input for rules
- ***Working Memory –*** a storage with *Facts,* where they are used for pattern matching and can be modified, inserted and removed
- ***Rule*** – represents a single rule which associates *Facts* with matching actions. It can be written in Drools Rule Language in the *.drl* files or as *Decision Table* in an excel spreadsheet
- ***Knowledge Session*** – it holds all the resources required for firing rules; all *Facts* are inserted into session, and then matching rules are fired
- ***Knowledge Base*** – represents the knowledge in the Drools ecosystem, it has the information about the resources where *Rules* are found, and also it creates the *Knowledge Session*
- ***Module –*** A module holds multiple Knowledge Bases which can hold different sessions

## 4. Java Configuration

To fire rules on a given data, we need to instantiate the framework-provided classes with information about the location of rule files and the *Facts:*

### 4.1. *KieServices*

The *KieServices* class provides access to all the Kie build and runtime facilities. It provides several factories, services, and utility methods. So, let's first get hold of a *KieServices* instance:

```
KieServices kieServices = KieServices.Factory.get();
```

Using the KieServices, we are going to create new instances of *KieFileSystem*, *KieBuilder*, and *KieContainer*.

### 4.2. *KieFileSystem*

First, we need to set the *KieFileSystem*; this is an in-memory file system provided by the framework. The following code provides the container to define the Drools resources like rules files and decision tables, programmatically:

```
private KieFileSystem getKieFileSystem() {
    KieFileSystem kieFileSystem = kieServices.newKieFileSystem();
    List<String> rules =
Arrays.asList("com/baeldung/drools/rules/BackwardChaining.drl",
"com/baeldung/drools/rules/SuggestApplicant.drl",
"com/baeldung/drools/rules/Product_rules.drl.xls");
    for (String rule : rules) {
        kieFileSystem.write(ResourceFactory.newClassPathResource(rule));
    }
    return kieFileSystem;
}
```

Here we are reading the files from *classpath* which is typically */src/main/resources* in case of a Maven project.

### 4.3. *KieBuilder*

(/)

Now, build the content of the KieFileSystem by passing it to KieBuilder and build all the defined rules using *kb.buildAll()*. This step compiles the rules and checks them for syntax errors:

```
KieBuilder kieBuilder = kieServices.newKieBuilder(kieFileSystem);
kieBuilder.buildAll();
```

### 4.4. *KieRepository*

The framework automatically adds the *KieModule* (resulting from the build) to *KieRepository*:

```
KieRepository kieRepository = kieServices.getRepository();
```

### 4.5. *KieContainer*

It is now possible to create a new *KieContainer* with this *KieModule* using its *ReleaseId*. In this case, Kie assigns a default *ReleaseId:*

```
ReleaseId krDefaultReleaseId = kieRepository.getDefaultReleaseId();
KieContainer kieContainer
  = kieServices.newKieContainer(krDefaultReleaseId);
```

### 4.6. *KieSession*

We can now obtain *KieSession* from the *KieContainer*. Our application interacts with the *KieSession*, which stores and executes on the runtime data:

```
KieSession kieSession = kieContainer.newKieSession();
```

Below is the full configuration:

```
public KieSession getKieSession() {
    KieBuilder kb = kieServices.newKieBuilder(getKieFileSystem());
    kb.buildAll();

    KieRepository kieRepository = kieServices.getRepository();
    ReleaseId krDefaultReleaseId = kieRepository.getDefaultReleaseId();
    KieContainer kieContainer =
kieServices.newKieContainer(krDefaultReleaseId);

    return kieContainer.newKieSession();
}
```

# 5. Implementing Rules

Now that we're done with the setup, let's have a look at a couple of options for creating rules.

We'll explore the rule implementation by an example of categorizing an applicant for a specific role, based on his current salary and number of years of experience he has.

## 5.1. Drools Rule File (*.drl*)

Simply put, the Drools rule file contains all business rules.

**A rule includes a *When-Then* construct**, here the *When* section lists the condition to be checked, and *Then* section lists the action to be taken if the condition is met:

```
package com.baeldung.drools.rules;

import com.baeldung.drools.model.Applicant;

global com.baeldung.drools.model.SuggestedRole suggestedRole;

dialect  "mvel"

rule "Suggest Manager Role"
    when
        Applicant(experienceInYears > 10)
        Applicant(currentSalary > 1000000 && currentSalary <=
         2500000)
    then
        suggestedRole.setRole("Manager");
end
```

This rule can be fired by inserting the *Applicant* and *SuggestedRole* facts in *KieSession:*

```
public SuggestedRole suggestARoleForApplicant(
    Applicant applicant,SuggestedRole suggestedRole){
    KieSession kieSession = kieContainer.newKieSession();
    kieSession.insert(applicant);
    kieSession.setGlobal("suggestedRole",suggestedRole);
    kieSession.fireAllRules();
    // ...
}
```

It tests two conditions on *Applicant* instance and then based on the fulfillment of both conditions it sets the *Role* field in the *SuggestedRole* object.

This can be verified by executing the test:

```
@Test
public void whenCriteriaMatching_ThenSuggestManagerRole(){
    Applicant applicant = new Applicant("David", 37, 1600000.0,11);
    SuggestedRole suggestedRole = new SuggestedRole();

    applicantService.suggestARoleForApplicant(applicant, suggestedRole);

    assertEquals("Manager", suggestedRole.getRole());
}
```

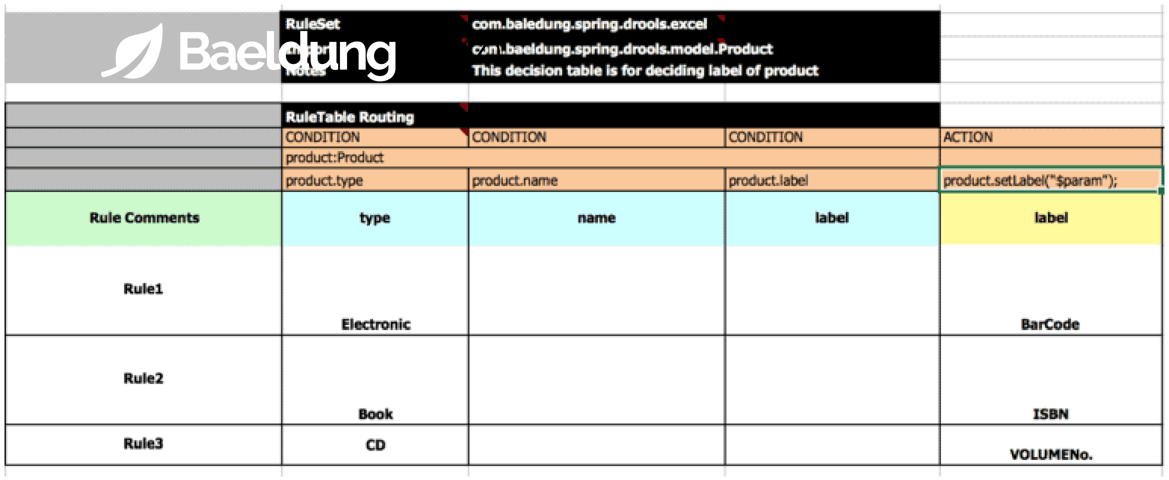
In this example, we've used few Drools provided keywords. Let's understand their use:

- ***package –*** this is the package name we specify in the *kmodule.xml,* the rule file is located inside this package
- ***import*** – this is similar to Java *import* statement, here we need to specify the classes which we are inserting in the *KnowledgeSession*
- ***global –*** this is used to define a global level variable for a session; this can be used to pass input parameter or to get an output parameter to summarize the information for a session
- ***dialect*** – a dialect specifies the syntax employed in the expressions in the condition section or action section. By default the dialect is Java. Drools also support dialect *mvel*; it is an expression language for Java-based applications. It supports the field and method/getter access
- ***rule*** – this defines a rule block with a rule name
- ***when*** – this specifies a rule condition, in this example the conditions which are checked are *Applicant* having *experienceInYears* more than ten years and *currentSalary* in a certain range
- ***then –*** this block executes the action when the conditions in the *when* block met. In this example, the *Applicant* role is set as Manager

## 5.2. Decision Tables

A decision table provides the capability of defining rules in a pre-formatted Excel spreadsheet. The advantage with Drools provided Decision Table is that they are easy to understand even for a non-technical person.

Also, it is useful when there are similar rules, but with different values, in this case, it is easier to add a new row on excel sheet in contrast to writing a new rule in *.drl* files. Let's see what the structure of a decision table with an example of applying the label on a product based on the product type:



| | | | | |
|---|---|---|---|---|
| RuleSet | com.baledung.spring.drools.excel | | | |
| Import | com.baeldung.spring.drools.model.Product | | | |
| Notes | This decision table is for deciding label of product | | | |
| | | | | |
| RuleTable Routing | | | | |
| CONDITION | CONDITION | CONDITION | ACTION | |
| product:Product | | | | |
| product.type | product.name | product.label | product.setLabel("$param"); | |
| Rule Comments | type | name | label | label |
| Rule1 | Electronic | | | BarCode |
| Rule2 | Book | | | ISBN |
| Rule3 | CD | | | VOLUMENo. |

(/wp-content/uploads/2017/05/Screen-Shot-2017-04-26-at-12.26.59-PM-2.png)
The Decision Table is grouped in different sections the top one is like a header section where we specify the *RuleSet* (i.e. package where rule files are located), *Import* (Java classes to be imported) and *Notes* (comments about the purpose of rules).

**The central section where we define rules is called *RuleTable* which groups the rules which are applied to the same domain object.**

In the next row, we have column types *CONDITION* and *ACTION*. Within these columns, we can access the properties of the domain object mentioned in one row and their values in the subsequent rows.

The mechanism to fire the rules is similar to what we have seen with *.drl* files.

We can verify the outcome of applying these rules by executing the test:

```
@Test
public void whenProductTypeElectronic_ThenLabelBarcode() {
    Product product = new Product("Microwave", "Electronic");
    product = productService.applyLabelToProduct(product);

    assertEquals("BarCode", product.getLabel());
}
```

## 6. Conclusion

In this quick article, we have explored making use of Drools as a business rule engine in our application. We've also seen the multiple ways by which we can write the rules in the Drools rule language as well as in easy to understand the language in spreadsheets.

As always, the complete code for this article is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/drools).