Red Hat Process Automation Manager 7.8

Designing a decision service using PMML models

Last Updated: 2020-07-29

Red Hat Process Automation Manager 7.8 Designing a decision service using PMML models

Red Hat Customer Content Services
brms-docs@redhat.com

Legal Notice

Abstract

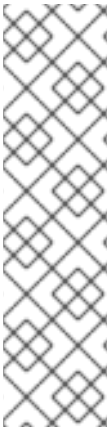
This document describes how to implement Predictive Model Markup Language (PMML) models in your decision services in Red Hat Process Automation Manager 7.8.

Table of Contents

PREFACE

As a business rules developer, you can use Predictive Model Markup Language (PMML) to define statistical or data-mining models that you can integrate with your decision services in Red Hat Process Automation Manager. Red Hat Process Automation Manager includes consumer conformance support of PMML 4.2.1 for Regression, Scorecard, Tree, and Mining models. Red Hat Process Automation Manager does not include a built-in PMML model editor, but you can use an XML or PMML-specific authoring tool to create PMML models and then integrate them with your Red Hat Process Automation Manager projects.

For more information about PMML, see the DMG PMML specification.



NOTE

You can also design your decision service using Decision Model and Notation (DMN) models and include your PMML models as part of your DMN service. For information about DMN support in Red Hat Process Automation Manager 7.8, see the following resources:

- *Getting started with decision services* (step-by-step tutorial with a DMN decision service example)
- *Designing a decision service using DMN models* (overview of DMN support and capabilities in Red Hat Process Automation Manager)

CHAPTER 1. PREDICTIVE MODEL MARKUP LANGUAGE (PMML)

Predictive Model Markup Language (PMML) is an XML-based standard established by the Data Mining Group (DMG) for defining statistical and data-mining models. PMML models can be shared between PMML-compliant platforms and across organizations so that business analysts and developers are unified in designing, analyzing, and implementing PMML-based assets and services.

For more information about the background and applications of PMML, see the DMG PMML specification.

1.1. PMML CONFORMANCE LEVELS

The PMML specification defines producer and consumer conformance levels in a software implementation to ensure that PMML models are created and integrated reliably. For the formal definitions of each conformance level, see the DMG PMML conformance page.

The following list summarizes the PMML conformance levels:

Producer conformance

A tool or application is producer conforming if it generates valid PMML documents for at least one type of model. Satisfying PMML producer conformance requirements ensures that a model definition document is syntactically correct and defines a model instance that is consistent with semantic criteria that are defined in model specifications.

Consumer conformance

An application is consumer conforming if it accepts valid PMML documents for at least one type of model. Satisfying consumer conformance requirements ensures that a PMML model created according to producer conformance can be integrated and used as defined. For example, if an application is consumer conforming for Regression model types, then valid PMML documents defining models of this type produced by different conforming producers would be interchangeable in the application.

Red Hat Process Automation Manager includes consumer conformance support for the following PMML 4.2.1 model types:

- Regression models
- Scorecard models
- Tree models
- Mining models (with sub-types **modelChain**, **selectAll**, and **selectFirst**)

For a list of all PMML model types, including those not supported in Red Hat Process Automation Manager, see the DMG PMML specification.

CHAPTER 2. PMML MODEL EXAMPLES

PMML defines an XML schema that enables PMML models to be used between different PMML-compliant platforms. The PMML specification enables multiple software platforms to work with the same file for authoring, testing, and production execution, assuming producer and consumer conformance are met.

The following are examples of PMML Regression, Scorecard, Tree, and Mining models. These examples illustrate the supported types of models that you can integrate with your decision services in Red Hat Process Automation Manager.

For more PMML examples, see the DMG PMML Sample Files page.

Example PMML Regression model

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.dmg.org/PMML-4_2">
  <Header copyright="JBoss"/>
  <DataDictionary numberOfFields="5">
    <DataField dataType="double" name="fld1" optype="continuous"/>
    <DataField dataType="double" name="fld2" optype="continuous"/>
    <DataField dataType="string" name="fld3" optype="categorical">
      <Value value="x"/>
      <Value value="y"/>
    </DataField>
    <DataField dataType="double" name="fld4" optype="continuous"/>
    <DataField dataType="double" name="fld5" optype="continuous"/>
  </DataDictionary>
  <RegressionModel algorithmName="linearRegression" functionName="regression"
modelName="LinReg" normalizationMethod="logit" targetFieldName="fld4">
    <MiningSchema>
      <MiningField name="fld1"/>
      <MiningField name="fld2"/>
      <MiningField name="fld3"/>
      <MiningField name="fld4" usageType="predicted"/>
      <MiningField name="fld5" usageType="target"/>
    </MiningSchema>
    <RegressionTable intercept="0.5">
      <NumericPredictor coefficient="5" exponent="2" name="fld1"/>
      <NumericPredictor coefficient="2" exponent="1" name="fld2"/>
      <CategoricalPredictor coefficient="-3" name="fld3" value="x"/>
      <CategoricalPredictor coefficient="3" name="fld3" value="y"/>
      <PredictorTerm coefficient="0.4">
        <FieldRef field="fld1"/>
        <FieldRef field="fld2"/>
      </PredictorTerm>
    </RegressionTable>
  </RegressionModel>
</PMML>
```

Example PMML Scorecard model

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns="http://www.dmg.org/PMML-4_2">
  <Header copyright="JBoss"/>
  <DataDictionary numberOfFields="4">
    <DataField name="param1" optype="continuous" dataType="double"/>
    <DataField name="param2" optype="continuous" dataType="double"/>
    <DataField name="overallScore" optype="continuous" dataType="double" />
    <DataField name="finalscore" optype="continuous" dataType="double" />
  </DataDictionary>
  <Scorecard modelName="ScorecardCompoundPredicate" useReasonCodes="true"
isScorable="true" functionName="regression"    baselineScore="15" initialScore="0.8"
reasonCodeAlgorithm="pointsAbove">
    <MiningSchema>
      <MiningField name="param1" usageType="active" invalidValueTreatment="asMissing">
      </MiningField>
      <MiningField name="param2" usageType="active" invalidValueTreatment="asMissing">
      </MiningField>
      <MiningField name="overallScore" usageType="target"/>
      <MiningField name="finalscore" usageType="predicted"/>
    </MiningSchema>
    <Characteristics>
      <Characteristic name="ch1" baselineScore="50" reasonCode="reasonCh1">
        <Attribute partialScore="20">
          <SimplePredicate field="param1" operator="lessThan" value="20"/>
        </Attribute>
        <Attribute partialScore="100">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param1" operator="greaterOrEqual" value="20"/>
            <SimplePredicate field="param2" operator="lessOrEqual" value="25"/>
          </CompoundPredicate>
        </Attribute>
        <Attribute partialScore="200">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param1" operator="greaterOrEqual" value="20"/>
            <SimplePredicate field="param2" operator="greaterThan" value="25"/>
          </CompoundPredicate>
        </Attribute>
      </Characteristic>
      <Characteristic name="ch2" reasonCode="reasonCh2">
        <Attribute partialScore="10">
          <CompoundPredicate booleanOperator="or">
            <SimplePredicate field="param2" operator="lessOrEqual" value="-5"/>
            <SimplePredicate field="param2" operator="greaterOrEqual" value="50"/>
          </CompoundPredicate>
        </Attribute>
        <Attribute partialScore="20">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param2" operator="greaterThan" value="-5"/>
            <SimplePredicate field="param2" operator="lessThan" value="50"/>
          </CompoundPredicate>
        </Attribute>
      </Characteristic>
    </Characteristics>
  </Scorecard>
</PMML>
```

Example PMML Tree model

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.dmg.org/PMML-4_2">
 <Header copyright="JBOSS"/>
 <DataDictionary numberOfFields="5">
  <DataField dataType="double" name="fld1" optype="continuous"/>
  <DataField dataType="double" name="fld2" optype="continuous"/>
  <DataField dataType="string" name="fld3" optype="categorical">
   <Value value="true"/>
   <Value value="false"/>
  </DataField>
  <DataField dataType="string" name="fld4" optype="categorical">
   <Value value="optA"/>
   <Value value="optB"/>
   <Value value="optC"/>
  </DataField>
  <DataField dataType="string" name="fld5" optype="categorical">
   <Value value="tgtX"/>
   <Value value="tgtY"/>
   <Value value="tgtZ"/>
  </DataField>
 </DataDictionary>
 <TreeModel functionName="classification" modelName="TreeTest">
  <MiningSchema>
   <MiningField name="fld1"/>
   <MiningField name="fld2"/>
   <MiningField name="fld3"/>
   <MiningField name="fld4"/>
   <MiningField name="fld5" usageType="predicted"/>
  </MiningSchema>
  <Node score="tgtX">
   <True/>
   <Node score="tgtX">
    <SimplePredicate field="fld4" operator="equal" value="optA"/>
    <Node score="tgtX">
     <CompoundPredicate booleanOperator="surrogate">
      <SimplePredicate field="fld1" operator="lessThan" value="30.0"/>
      <SimplePredicate field="fld2" operator="greaterThan" value="20.0"/>
     </CompoundPredicate>
     <Node score="tgtX">
      <SimplePredicate field="fld2" operator="lessThan" value="40.0"/>
     </Node>
     <Node score="tgtZ">
      <SimplePredicate field="fld2" operator="greaterOrEqual" value="10.0"/>
     </Node>
    </Node>
    <Node score="tgtZ">
     <CompoundPredicate booleanOperator="or">
      <SimplePredicate field="fld1" operator="greaterOrEqual" value="60.0"/>
      <SimplePredicate field="fld1" operator="lessOrEqual" value="70.0"/>
     </CompoundPredicate>
     <Node score="tgtZ">
      <SimpleSetPredicate booleanOperator="isNotIn" field="fld4">
       <Array type="string">optA optB</Array>
      </SimpleSetPredicate>
     </Node>
```

```
        </Node>
      </Node>
      <Node score="tgtY">
        <CompoundPredicate booleanOperator="or">
          <SimplePredicate field="fld4" operator="equal" value="optA"/>
          <SimplePredicate field="fld4" operator="equal" value="optC"/>
        </CompoundPredicate>
        <Node score="tgtY">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="fld1" operator="greaterThan" value="10.0"/>
            <SimplePredicate field="fld1" operator="lessThan" value="50.0"/>
            <SimplePredicate field="fld4" operator="equal" value="optA"/>
            <SimplePredicate field="fld2" operator="lessThan" value="100.0"/>
            <SimplePredicate field="fld3" operator="equal" value="false"/>
          </CompoundPredicate>
        </Node>
        <Node score="tgtZ">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="fld4" operator="equal" value="optC"/>
            <SimplePredicate field="fld2" operator="lessThan" value="30.0"/>
          </CompoundPredicate>
        </Node>
      </Node>
    </Node>
  </TreeModel>
</PMML>
```

Example PMML Mining model (modelChain)

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.dmg.org/PMML-4_2">
  <Header>
    <Application name="Drools-PMML" version="7.0.0-SNAPSHOT" />
  </Header>
  <DataDictionary numberOfFields="7">
    <DataField name="age" optype="continuous" dataType="double" />
    <DataField name="occupation" optype="categorical" dataType="string">
      <Value value="SKYDIVER" />
      <Value value="ASTRONAUT" />
      <Value value="PROGRAMMER" />
      <Value value="TEACHER" />
      <Value value="INSTRUCTOR" />
    </DataField>
    <DataField name="residenceState" optype="categorical" dataType="string">
      <Value value="AP" />
      <Value value="KN" />
      <Value value="TN" />
    </DataField>
    <DataField name="validLicense" optype="categorical" dataType="boolean" />
    <DataField name="overallScore" optype="continuous" dataType="double" />
    <DataField name="grade" optype="categorical" dataType="string">
      <Value value="A" />
      <Value value="B" />
      <Value value="C" />
      <Value value="D" />
```

```
      <Value value="F" />
    </DataField>
    <DataField name="qualificationLevel" optype="categorical" dataType="string">
      <Value value="Unqualified" />
      <Value value="Barely" />
      <Value value="Well" />
      <Value value="Over" />
    </DataField>
  </DataDictionary>
  <MiningModel modelName="SampleModelChainMine" functionName="classification">
    <MiningSchema>
      <MiningField name="age" />
      <MiningField name="occupation" />
      <MiningField name="residenceState" />
      <MiningField name="validLicense" />
      <MiningField name="overallScore" />
      <MiningField name="qualificationLevel" usageType="target"/>
    </MiningSchema>
    <Segmentation multipleModelMethod="modelChain">
      <Segment id="1">
        <True />
        <Scorecard modelName="Sample Score 1" useReasonCodes="true" isScorable="true"
functionName="regression"              baselineScore="0.0" initialScore="0.345">
          <MiningSchema>
            <MiningField name="age" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="occupation" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="residenceState" usageType="active" invalidValueTreatment="asMissing"
/>
            <MiningField name="validLicense" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="overallScore" usageType="predicted" />
          </MiningSchema>
          <Output>
            <OutputField name="calculatedScore" displayName="Final Score" dataType="double"
feature="predictedValue"                    targetField="overallScore" />
          </Output>
          <Characteristics>
            <Characteristic name="AgeScore" baselineScore="0.0" reasonCode="ABZ">
              <Extension name="cellRef" value="$B$8" />
              <Attribute partialScore="10.0">
                <Extension name="cellRef" value="$C$10" />
                <SimplePredicate field="age" operator="lessOrEqual" value="5" />
              </Attribute>
              <Attribute partialScore="30.0" reasonCode="CX1">
                <Extension name="cellRef" value="$C$11" />
                <CompoundPredicate booleanOperator="and">
                  <SimplePredicate field="age" operator="greaterOrEqual" value="5" />
                  <SimplePredicate field="age" operator="lessThan" value="12" />
                </CompoundPredicate>
              </Attribute>
              <Attribute partialScore="40.0" reasonCode="CX2">
                <Extension name="cellRef" value="$C$12" />
                <CompoundPredicate booleanOperator="and">
                  <SimplePredicate field="age" operator="greaterOrEqual" value="13" />
                  <SimplePredicate field="age" operator="lessThan" value="44" />
                </CompoundPredicate>
              </Attribute>
```

```
          <Attribute partialScore="25.0">
            <Extension name="cellRef" value="$C$13" />
            <SimplePredicate field="age" operator="greaterOrEqual" value="45" />
          </Attribute>
        </Characteristic>
        <Characteristic name="OccupationScore" baselineScore="0.0">
          <Extension name="cellRef" value="$B$16" />
          <Attribute partialScore="-10.0" reasonCode="CX2">
            <Extension name="description" value="skydiving is a risky occupation" />
            <Extension name="cellRef" value="$C$18" />
            <SimpleSetPredicate field="occupation" booleanOperator="isIn">
              <Array n="2" type="string">SKYDIVER ASTRONAUT</Array>
            </SimpleSetPredicate>
          </Attribute>
          <Attribute partialScore="10.0">
            <Extension name="cellRef" value="$C$19" />
            <SimpleSetPredicate field="occupation" booleanOperator="isIn">
              <Array n="2" type="string">TEACHER INSTRUCTOR</Array>
            </SimpleSetPredicate>
          </Attribute>
          <Attribute partialScore="5.0">
            <Extension name="cellRef" value="$C$20" />
            <SimplePredicate field="occupation" operator="equal" value="PROGRAMMER" />
          </Attribute>
        </Characteristic>
        <Characteristic name="ResidenceStateScore" baselineScore="0.0" reasonCode="RES">
          <Extension name="cellRef" value="$B$22" />
          <Attribute partialScore="-10.0">
            <Extension name="cellRef" value="$C$24" />
            <SimplePredicate field="residenceState" operator="equal" value="AP" />
          </Attribute>
          <Attribute partialScore="10.0">
            <Extension name="cellRef" value="$C$25" />
            <SimplePredicate field="residenceState" operator="equal" value="KN" />
          </Attribute>
          <Attribute partialScore="5.0">
            <Extension name="cellRef" value="$C$26" />
            <SimplePredicate field="residenceState" operator="equal" value="TN" />
          </Attribute>
        </Characteristic>
        <Characteristic name="ValidLicenseScore" baselineScore="0.0">
          <Extension name="cellRef" value="$B$28" />
          <Attribute partialScore="1.0" reasonCode="LX00">
            <Extension name="cellRef" value="$C$30" />
            <SimplePredicate field="validLicense" operator="equal" value="true" />
          </Attribute>
          <Attribute partialScore="-1.0" reasonCode="LX00">
            <Extension name="cellRef" value="$C$31" />
            <SimplePredicate field="validLicense" operator="equal" value="false" />
          </Attribute>
        </Characteristic>
      </Characteristics>
    </Scorecard>
  </Segment>
  <Segment id="2">
    <True />
```

```
      <TreeModel modelName="SampleTree" functionName="classification"
missingValueStrategy="lastPrediction" noTrueChildStrategy="returnLastPrediction">
        <MiningSchema>
          <MiningField name="age" usageType="active" />
          <MiningField name="validLicense" usageType="active" />
          <MiningField name="calculatedScore" usageType="active" />
          <MiningField name="qualificationLevel" usageType="predicted" />
        </MiningSchema>
        <Output>
          <OutputField name="qualification" displayName="Qualification Level" dataType="string"
feature="predictedValue"                    targetField="qualificationLevel" />
        </Output>
        <Node score="Well" id="1">
          <True/>
          <Node score="Barely" id="2">
            <CompoundPredicate booleanOperator="and">
              <SimplePredicate field="age" operator="greaterOrEqual" value="16" />
              <SimplePredicate field="validLicense" operator="equal" value="true" />
            </CompoundPredicate>
            <Node score="Barely" id="3">
              <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="50.0" />
            </Node>
            <Node score="Well" id="4">
              <CompoundPredicate booleanOperator="and">
                <SimplePredicate field="calculatedScore" operator="greaterThan" value="50.0" />
                <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="60.0" />
              </CompoundPredicate>
            </Node>
            <Node score="Over" id="5">
              <SimplePredicate field="calculatedScore" operator="greaterThan" value="60.0" />
            </Node>
          </Node>
          <Node score="Unqualified" id="6">
            <CompoundPredicate booleanOperator="surrogate">
              <SimplePredicate field="age" operator="lessThan" value="16" />
              <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="40.0" />
              <True />
            </CompoundPredicate>
          </Node>
        </Node>
      </TreeModel>
    </Segment>
  </Segmentation>
 </MiningModel>
</PMML>
```

CHAPTER 3. PMML SUPPORT IN RED HAT PROCESS AUTOMATION MANAGER

Red Hat Process Automation Manager includes consumer conformance support for the following PMML 4.2.1 model types:

- Regression models
- Scorecard models
- Tree models
- Mining models (with sub-types **modelChain**, **selectAll**, and **selectFirst**)

For a list of all PMML model types, including those not supported in Red Hat Process Automation Manager, see the DMG PMML specification.

Red Hat Process Automation Manager does not include a built-in PMML model editor, but you can use an XML or PMML-specific authoring tool to create PMML models and then integrate the PMML models in your decision services in Red Hat Process Automation Manager. You can import PMML files into your project in Business Central (**Menu → Design → Projects → Import Asset**) or package the PMML files as part of your project knowledge JAR (KJAR) file without Business Central.

When you add a PMML file to a project in Red Hat Process Automation Manager, multiple assets are generated. Each type of PMML model generates a different set of assets, but all PMML model types generate at least the following set of assets:

- A DRL file that contains all of the rules associated with your PMML model
- At least two Java classes:
  - A data class that is used as the default object type for the model type
  - A **RuleUnit** class that is used to manage data sources and rule execution

If a PMML file has **MiningModel** as the root model, multiple instances of each of these files are generated.

For more information about including assets such as PMML files with your project packaging and deployment method, see *Packaging and deploying a Red Hat Process Automation Manager project* .

3.1. PMML NAMING CONVENTIONS IN RED HAT PROCESS AUTOMATION MANAGER

The following are naming conventions for generated PMML packages, classes, and rules:

- If no package name is given in a PMML model file, then the default package name **org.kie.pmml.pmml_4_2** is prefixed to the model name for the generated rules in the format **"org.kie.pmml.pmml_4_2"+modelName**.
- The package name for the generated **RuleUnit** Java class is the same as the package name for the generated rules.
- The name of the generated **RuleUnit** Java class is the model name with **RuleUnit** added to it in the format **modelName+"RuleUnit"**.

- Each PMML model has at least one data class that is generated. The package name for these classes is **org.kie.pmml.pmml_4_2.model**.
- The names of generated data classes are determined by the model type, prefixed with the model name:
  - Regression models: One data class named **modelName+"RegressionData"**
  - Scorecard models: One data class named **modelName+"ScoreCardData"**
  - Tree models: Two data classes, the first named **modelName+"TreeNode"** and the second named **modelName+"TreeToken"**
  - Mining models: One data class named **modelName+"MiningModelData"**

NOTE

The mining model also generates all of the rules and classes that are within each of its segments.

3.2. PMML EXTENSIONS IN RED HAT PROCESS AUTOMATION MANAGER

The PMML specification supports **Extension** elements that extend the content of a PMML model. You can use extensions at almost every level of a PMML model definition, and as the first and last child in the main element of a model for maximum flexibility. For more information about PMML extensions, see the DMG PMML Extension Mechanism.

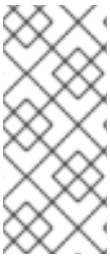
To optimize PMML integration, Red Hat Process Automation Manager supports the following additional PMML extensions:

- **modelPackage**: Designates a package name for the generated rules and Java classes. Include this extension in the **Header** section of the PMML model file.
- **adapter**: Designates the type of construct (**bean** or **trait**) that is used to contain input and output data for rules. Insert this extension in the **MiningSchema** or **Output** section (or both) of the PMML model file.
- **externalClass**: Used in conjunction with the **adapter** extension in defining a **MiningField** or **OutputField**. This extension contains a class with an attribute name that matches the name of the **MiningField** or **OutputField** element.

CHAPTER 4. PMML MODEL EXECUTION

You can import PMML files into your Red Hat Process Automation Manager project using Business Central (**Menu → Design → Projects → Import Asset**) or package the PMML files as part of your project knowledge JAR (KJAR) file without Business Central. After you implement your PMML files in your Red Hat Process Automation Manager project, you can execute the PMML-based decision service by embedding PMML calls directly in your Java application or by sending an **ApplyPmmlModelCommand** command to a configured KIE Server.

For more information about including PMML assets with your project packaging and deployment method, see *Packaging and deploying a Red Hat Process Automation Manager project* .



NOTE

You can also include a PMML model as part of a Decision Model and Notation (DMN) service in Business Central. When you include a PMML model within a DMN file, you can invoke that PMML model as a boxed function expression for a DMN decision node or business knowledge model node. For more information about including PMML models in a DMN service, see *Designing a decision service using DMN models* .

4.1. EMBEDDING A PMML CALL DIRECTLY IN A JAVA APPLICATION

A KIE container is local when the knowledge assets are either embedded directly into the calling program or are physically pulled in using Maven dependencies for the KJAR. You typically embed knowledge assets directly into a project if there is a tight relationship between the version of the code and the version of the PMML definition. Any changes to the decision take effect after you have intentionally updated and redeployed the application. A benefit of this approach is that proper operation does not rely on any external dependencies to the run time, which can be a limitation of locked-down environments.

Using Maven dependencies enables further flexibility because the specific version of the decision can dynamically change (for example, by using a system property), and it can be periodically scanned for updates and automatically updated. This introduces an external dependency on the deploy time of the service, but executes the decision locally, reducing reliance on an external service being available during run time.

Prerequisites

- A KJAR containing the PMML model to execute has been created. For more information about project packaging, see *Packaging and deploying a Red Hat Process Automation Manager project* .

Procedure

1. In your client application, add the following dependencies to the relevant classpath of your Java project:

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
```

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

The **<version>** is the Maven artifact version for Red Hat Process Automation Manager currently used in your project (for example, 7.39.0.Final-redhat-00005).

NOTE

Instead of specifying a Red Hat Process Automation Manager **<version>** for individual dependencies, consider adding the Red Hat Business Automation bill of materials (BOM) dependency to your project **pom.xml** file. The Red Hat Business Automation BOM applies to both Red Hat Decision Manager and Red Hat Process Automation Manager. When you add the BOM files, the correct versions of transitive dependencies from the provided Maven repositories are included in the project.

Example BOM dependency:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.8.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

For more information about the Red Hat Business Automation BOM, see What is the mapping between RHPAM product and maven library version?.

2. Create a KIE container from **classpath** or **ReleaseId**:

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

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

Alternative option:

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

KieContainer kieContainer = kieServices.getKieClasspathContainer();
```

3. Create an instance of the **PMMLRequestData** class, which applies your PMML model to a set of data:

```
public class PMMLRequestData {
   private String correlationId; (1)
   private String modelName; (2)
   private String source; (3)
   private List<ParameterInfo<?>> requestParams; (4)
   ...
}
```

(1) Identifies data that is associated with a particular request or result

(2) The name of the model that should be applied to the request data

(3) Used by internally generated **PMMLRequestData** objects to identify the segment that generated the request

(4) The default mechanism for sending input data points

4. Create an instance of the **PMML4Result** class, which holds the output information that is the result of applying the PMML-based rules to the input data:

```
public class PMML4Result {
   private String correlationId;
   private String segmentationId; (1)
   private String segmentId; (2)
   private int segmentIndex; (3)
   private String resultCode; (4)
   private Map<String, Object> resultVariables; (5)
   ...
}
```

(1) Used when the model type is **MiningModel**. The **segmentationId** is used to differentiate between multiple segmentations.

(2) Used in conjunction with the **segmentationId** to identify which segment generated the results.

(3) Used to maintain the order of segments.

(4) Used to determine whether the model was successfully applied, where **OK** indicates success.

(5) Contains the name of a resultant variable and its associated value.

In addition to the normal getter methods, the **PMML4Result** class also supports the following methods for directly retrieving the values for result variables:

```
public <T> Optional<T> getResultValue(String objName, String objField, Class<T> clazz,
Object...params)

public Object getResultValue(String objName, String objField, Object...params)
```

5. Create an instance of the **ParameterInfo** class, which serves as a wrapper for basic data type objects used as part of the **PMMLRequestData** class:

```
public class ParameterInfo<T> { (1)
    private String correlationId;
    private String name; (2)
    private String capitalizedName;
    private Class<T> type; (3)
    private T value; (4)
    ...
}
```

(1) The parameterized class to handle many different types

(2) The name of the variable that is expected as input for the model

(3) The class that is the actual type of the variable

(4) The actual value of the variable

6. Execute the PMML model based on the required PMML class instances that you have created:

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String correlationId,
                         String modelPkgName) {
    RuleUnitExecutor executor = RuleUnitExecutor.create().bind(kbase);
    PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
    PMML4Result resultHolder = new PMML4Result(correlationId);
    variables.entrySet().forEach( es -> {
        request.addRequestParam(es.getKey(), es.getValue());
    });

    DataSource<PMMLRequestData> requestData = executor.newDataSource("request");
    DataSource<PMML4Result> resultData = executor.newDataSource("results");
    DataSource<PMMLData> internalData = executor.newDataSource("pmmlData");

    requestData.insert(request);
    resultData.insert(resultHolder);

    List<String> possiblePackageNames = calculatePossiblePackageNames(modelName,
                                                                       modelPkgName);
    Class<? extends RuleUnit> ruleUnitClass = getStartingRuleUnit("RuleUnitIndicator",
                                                                  (InternalKnowledgeBase)kbase,
                                                                  possiblePackageNames);

    if (ruleUnitClass != null) {
        executor.run(ruleUnitClass);
        if ( "OK".equals(resultHolder.getResultCode()) ) {
          // extract result variables here
        }
    }
}
```

```
protected Class<? extends RuleUnit> getStartingRuleUnit(String startingRule,
InternalKnowledgeBase ikb, List<String> possiblePackages) {
    RuleUnitRegistry unitRegistry = ikb.getRuleUnitRegistry();
    Map<String,InternalKnowledgePackage> pkgs = ikb.getPackagesMap();
    RuleImpl ruleImpl = null;
    for (String pkgName: possiblePackages) {
      if (pkgs.containsKey(pkgName)) {
          InternalKnowledgePackage pkg = pkgs.get(pkgName);
          ruleImpl = pkg.getRule(startingRule);
          if (ruleImpl != null) {
              RuleUnitDescr descr = unitRegistry.getRuleUnitFor(ruleImpl).orElse(null);
              if (descr != null) {
                  return descr.getRuleUnitClass();
              }
          }
      }
    }
    return null;
}

protected List<String> calculatePossiblePackageNames(String modelId,
String...knownPackageNames) {
    List<String> packageNames = new ArrayList<>();
    String javaModelId = modelId.replaceAll("\\s","");
    if (knownPackageNames != null && knownPackageNames.length > 0) {
        for (String knownPkgName: knownPackageNames) {
            packageNames.add(knownPkgName + "." + javaModelId);
        }
    }
    String basePkgName = PMML4UnitImpl.DEFAULT_ROOT_PACKAGE+"."+javaModelId;
    packageNames.add(basePkgName);
    return packageNames;
}
```

Rules are executed by the **RuleUnitExecutor** class. The **RuleUnitExecutor** class creates KIE sessions and adds the required **DataSource** objects to those sessions, and then executes the rules based on the **RuleUnit** that is passed as a parameter to the **run()** method. The **calculatePossiblePackageNames** and the **getStartingRuleUnit** methods determine the fully qualified name of the **RuleUnit** class that is passed to the **run()** method.

To facilitate your PMML model execution, you can also use a **PMML4ExecutionHelper** class supported in Red Hat Process Automation Manager. For more information about the PMML helper class, see Section 4.1.1, “PMML execution helper class”.

4.1.1. PMML execution helper class

Red Hat Process Automation Manager provides a **PMML4ExecutionHelper** class that helps create the **PMMLRequestData** class required for PMML model execution and that helps execute rules using the **RuleUnitExecutor** class.

The following are examples of a PMML model execution without and with the **PMML4ExecutionHelper** class, as a comparison:

Example PMML model execution without using PMML4ExecutionHelper

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String correlationId,
                         String modelPkgName) {
    RuleUnitExecutor executor = RuleUnitExecutor.create().bind(kbase);
    PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
    PMML4Result resultHolder = new PMML4Result(correlationId);
    variables.entrySet().forEach( es -> {
        request.addRequestParam(es.getKey(), es.getValue());
    });

    DataSource<PMMLRequestData> requestData = executor.newDataSource("request");
    DataSource<PMML4Result> resultData = executor.newDataSource("results");
    DataSource<PMMLData> internalData = executor.newDataSource("pmmlData");

    requestData.insert(request);
    resultData.insert(resultHolder);

    List<String> possiblePackageNames = calculatePossiblePackageNames(modelName,
                                                    modelPkgName);
    Class<? extends RuleUnit> ruleUnitClass = getStartingRuleUnit("RuleUnitIndicator",
                                                (InternalKnowledgeBase)kbase,
                                                possiblePackageNames);

    if (ruleUnitClass != null) {
        executor.run(ruleUnitClass);
        if ( "OK".equals(resultHolder.getResultCode()) ) {
          // extract result variables here
        }
    }
}

protected Class<? extends RuleUnit> getStartingRuleUnit(String startingRule,
InternalKnowledgeBase ikb, List<String> possiblePackages) {
    RuleUnitRegistry unitRegistry = ikb.getRuleUnitRegistry();
    Map<String,InternalKnowledgePackage> pkgs = ikb.getPackagesMap();
    RuleImpl ruleImpl = null;
    for (String pkgName: possiblePackages) {
      if (pkgs.containsKey(pkgName)) {
          InternalKnowledgePackage pkg = pkgs.get(pkgName);
          ruleImpl = pkg.getRule(startingRule);
          if (ruleImpl != null) {
              RuleUnitDescr descr = unitRegistry.getRuleUnitFor(ruleImpl).orElse(null);
              if (descr != null) {
                  return descr.getRuleUnitClass();
              }
          }
      }
    }
    return null;
}

protected List<String> calculatePossiblePackageNames(String modelId,
String...knownPackageNames) {
    List<String> packageNames = new ArrayList<>();
```

```
    String javaModelId = modelId.replaceAll("\\s","");
    if (knownPackageNames != null && knownPackageNames.length > 0) {
        for (String knownPkgName: knownPackageNames) {
            packageNames.add(knownPkgName + "." + javaModelId);
        }
    }
    String basePkgName = PMML4UnitImpl.DEFAULT_ROOT_PACKAGE+"."+javaModelId;
    packageNames.add(basePkgName);
    return packageNames;
}
```

Example PMML model execution using PMML4ExecutionHelper

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String modelPkgName,
                         String correlationId) {
  PMML4ExecutionHelper helper = PMML4ExecutionHelperFactory.getExecutionHelper(modelName,
kbase);
  helper.addPossiblePackageName(modelPkgName);

  PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
  variables.entrySet().forEach(entry -> {
    request.addRequestParam(entry.getKey(), entry.getValue);
  });

  PMML4Result resultHolder = helper.submitRequest(request);
  if ("OK".equals(resultHolder.getResultCode)) {
    // extract result variables here
  }
}
```

When you use the **PMML4ExecutionHelper**, you do not need to specify the possible package names nor the **RuleUnit** class as you would in a typical PMML model execution.

To construct a **PMML4ExecutionHelper** class, you use the **PMML4ExecutionHelperFactory** class to determine how instances of **PMML4ExecutionHelper** are retrieved.

The following are the available **PMML4ExecutionHelperFactory** class methods for constructing a **PMML4ExecutionHelper** class:

PMML4ExecutionHelperFactory methods for PMML assets in a KIE base

Use these methods when PMML assets have already been compiled and are being used from an existing KIE base:

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, KieBase kbase)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, KieBase kbase,
boolean includeMiningDataSources)
```

PMML4ExecutionHelperFactory methods for PMML assets on the project classpath

Use these methods when PMML assets are on the project classpath. The **classPath** argument is the project classpath location of the PMML file:

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName,  String classPath,
KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName,String classPath,
KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

PMML4ExecutionHelperFactory methods for PMML assets in a byte array

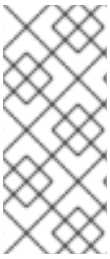
Use these methods when PMML assets are in the form of a byte array:

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, byte[] content,
KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, byte[] content,
KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

PMML4ExecutionHelperFactory methods for PMML assets in a Resource

Use these methods when PMML assets are in the form of an **org.kie.api.io.Resource** object:

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, Resource resource,
KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, Resource resource,
KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```



NOTE

The classpath, byte array, and resource **PMML4ExecutionHelperFactory** methods create a KIE container for the generated rules and Java classes. The container is used as the source of the KIE base that the **RuleUnitExecutor** uses. The container is not persisted. The **PMML4ExecutionHelperFactory** method for PMML assets that are already in a KIE base does not create a KIE container in this way.

4.2. EXECUTING A PMML MODEL USING KIE SERVER

You can execute PMML models that have been deployed to KIE Server by sending the **ApplyPmmlModelCommand** command to the configured KIE Server. When you use this command, a **PMMLRequestData** object is sent to the KIE Server and a **PMML4Result** result object is received as a reply. You can send PMML requests to KIE Server through the KIE Server REST API from a configured Java class or directly from a REST client.

Prerequisites

- KIE Server is installed and configured, including a known user name and credentials for a user with the **kie-server** role. For installation options, see *Planning a Red Hat Process Automation Manager installation*.
- A KIE container is deployed in KIE Server in the form of a KJAR that includes the PMML model. For more information about project packaging, see *Packaging and deploying a Red Hat Process Automation Manager project*.
- You have the container ID of the KIE container containing the PMML model.

Procedure

1. In your client application, add the following dependencies to the relevant classpath of your Java project:

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>

<!-- Required for the KIE Server Java client API -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

The **<version>** is the Maven artifact version for Red Hat Process Automation Manager currently used in your project (for example, 7.39.0.Final-redhat-00005).

NOTE

Instead of specifying a Red Hat Process Automation Manager **<version>** for individual dependencies, consider adding the Red Hat Business Automation bill of materials (BOM) dependency to your project **pom.xml** file. The Red Hat Business Automation BOM applies to both Red Hat Decision Manager and Red Hat Process Automation Manager. When you add the BOM files, the correct versions of transitive dependencies from the provided Maven repositories are included in the project.

Example BOM dependency:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.8.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

For more information about the Red Hat Business Automation BOM, see What is the mapping between RHPAM product and maven library version?.

2. Create a KIE container from **classpath** or **ReleaseId**:

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

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

Alternative option:

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

KieContainer kieContainer = kieServices.getKieClasspathContainer();
```

3. Create a class for sending requests to KIE Server and receiving responses:

```
public class ApplyScorecardModel {
  private static final ReleaseId releaseId =
        new ReleaseId("org.acme","my-kjar","1.0.0");
  private static final String containerId = "SampleModelContainer";
  private static KieCommands commandFactory;
  private static ClassLoader kjarClassLoader; (1)
  private RuleServicesClient serviceClient; (2)

  // Attributes specific to your class instance
  private String rankedFirstCode;
  private Double score;

  // Initialization of non-final static attributes
  static {
    commandFactory = KieServices.Factory.get().getCommands();
```

```
    // Specifications for kjarClassLoader, if used
    KieMavenRepository kmp = KieMavenRepository.getMavenRepository();
    File artifactFile = kmp.resolveArtifact(releaseId).getFile();
    if (artifactFile != null) {
      URL urls[] = new URL[1];
      try {
        urls[0] = artifactFile.toURI().toURL();
        classLoader = new KieURLClassLoader(urls,PMML4Result.class.getClassLoader());
      } catch (MalformedURLException e) {
        logger.error("Error getting classLoader for "+containerId);
        logger.error(e.getMessage());
      }
    } else {
      logger.warn("Did not find the artifact file for "+releaseId.toString());
    }
  }

  public ApplyScorecardModel(KieServicesConfiguration kieConfig) {
    KieServicesClient clientFactory = KieServicesFactory.newKieServicesClient(kieConfig);
    serviceClient = clientFactory.getServicesClient(RuleServicesClient.class);
  }
  ...
  // Getters and setters
  ...

  // Method for executing the PMML model on KIE Server
  public void applyModel(String occupation, int age) {
    PMMLRequestData input = new PMMLRequestData("1234","SampleModelName"); (3)
    input.addRequestParam(new ParameterInfo("1234","occupation",String.class,occupation));
    input.addRequestParam(new ParameterInfo("1234","age",Integer.class,age));

    CommandFactoryServiceImpl cf = (CommandFactoryServiceImpl)commandFactory;
    ApplyPmmlModelCommand command = (ApplyPmmlModelCommand)
cf.newApplyPmmlModel(request); (4)

    ServiceResponse<ExecutionResults> results =
        ruleClient.executeCommandsWithResults(CONTAINER_ID, command); (5)

    if (results != null) { (6)
      PMML4Result resultHolder = (PMML4Result)results.getResult().getValue("results");
      if (resultHolder != null && "OK".equals(resultHolder.getResultCode())) {
        this.score = resultHolder.getResultValue("ScoreCard","score",Double.class).get();
        Map<String,Object> rankingMap =
            (Map<String,Object>)resultHolder.getResultValue("ScoreCard","ranking");
        if (rankingMap != null && !rankingMap.isEmpty()) {
          this.rankedFirstCode = rankingMap.keySet().iterator().next();
        }
      }
    }
  }
}
```

(1) Defines the class loader if you did not include the KJAR in your client project dependencies

(2) Identifies the service client as defined in the configuration settings, including KIE Server REST API access credentials

3. Initializes a **PMMLRequestData** object
4. Creates an instance of the **ApplyPmmlModelCommand**
5. Sends the command using the service client
6. Retrieves the results of the executed PMML model

4. Execute the class instance to send the PMML invocation request to KIE Server. Alternatively, you can use JMS and REST interfaces to send the **ApplyPmmlModelCommand** command to KIE Server. For REST requests, you use the **ApplyPmmlModelCommand** command as a **POST** request to **http://SERVER:PORT/kie-server/services/rest/server/containers/instances/{containerId}** in JSON, JAXB, or XStream request format.

Example POST endpoint

```
http://localhost:8080/kie-server/services/rest/server/containers/instances/SampleModelContainer
```

Example JSON request body

```
{
  "commands": [ {
     "apply-pmml-model-command": {
       "outIdentifier": null,
       "packageName": null,
       "hasMining": false,
       "requestData": {
         "correlationId": "123",
         "modelName": "SimpleScorecard",
         "source": null,
         "requestParams": [
           {
             "correlationId": "123",
             "name": "param1",
             "type": "java.lang.Double",
             "value": "10.0"
           },
           {
             "correlationId": "123",
             "name": "param2",
             "type": "java.lang.Double",
             "value": "15.0"
           }
         ]
       }
     }
   }
  ]
}
```

Example curl request with endpoint and body

```
curl -X POST "http://localhost:8080/kie-server/services/rest/server/containers/instances/SampleModelContainer" -H "accept: application/json" -H "content-type: application/json" -d "{ \"commands\": [ { \"apply-pmml-model-command\": { \"outIdentifier\": null, \"packageName\": null, \"hasMining\": false, \"requestData\": { \"correlationId\": \"123\", \"modelName\": \"SimpleScorecard\", \"source\": null, \"requestParams\": [ { \"correlationId\": \"123\", \"name\": \"param1\", \"type\": \"java.lang.Double\", \"value\": \"10.0\" }, { \"correlationId\": \"123\", \"name\": \"param2\", \"type\": \"java.lang.Double\", \"value\": \"15.0\" } ] } } } ]}"
```

Example JSON response


```
{
  "results" : [ {
    "value" : {"org.kie.api.pmml.DoubleFieldOutput":{
  "value" : 40.8,
  "correlationId" : "123",
  "segmentationId" : null,
  "segmentId" : null,
  "name" : "OverallScore",
  "displayValue" : "OverallScore",
  "weight" : 1.0
}},
    "key" : "OverallScore"
  }, {
    "value" : {"org.kie.api.pmml.PMML4Result":{
  "resultVariables" : {
    "OverallScore" : {
      "value" : 40.8,
      "correlationId" : "123",
      "segmentationId" : null,
      "segmentId" : null,
      "name" : "OverallScore",
      "displayValue" : "OverallScore",
      "weight" : 1.0
    },
    "ScoreCard" : {
      "modelName" : "SimpleScorecard",
      "score" : 40.8,
      "holder" : {
        "modelName" : "SimpleScorecard",
        "correlationId" : "123",
        "voverallScore" : null,
        "moverallScore" : true,
        "vparam1" : 10.0,
        "mparam1" : false,
        "vparam2" : 15.0,
        "mparam2" : false
      },
      "enableRC" : true,
      "pointsBelow" : true,
      "ranking" : {
        "reasonCh1" : 5.0,
        "reasonCh2" : -6.0
      }
    }
```

```
  },
  "correlationId" : "123",
  "segmentationId" : null,
  "segmentId" : null,
  "segmentIndex" : 0,
  "resultCode" : "OK",
  "resultObjectName" : null
}},
    "key" : "results"
  } ],
  "facts" : [ ]
}
```

CHAPTER 5. ADDITIONAL RESOURCES

- PMML specification
- *Packaging and deploying a Red Hat Process Automation Manager project*
- *Interacting with Red Hat Process Automation Manager using KIE APIs*

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Wednesday, July 29, 2020.