IBM Operational Decision Manager
Version 8 Release 0

Getting Started Tutorials

IBM
## Contents

**Tutorials** ........................................... 1

Tutorial: Creating a copybook-based ruleset for zRule Execution Server for z/OS .................. 1

Before you start .................................. 2

Task 1: Importing an existing rule project ........ 4

Task 2: Generating a XOM from a COBOL copybook ............................................. 5

Task 3: Creating a BOM from the COBOL XOM . 6

Task 4: Declaring ruleset parameters ............ 7

Task 5: Adding BOM methods and mapping them to the XOM .................................... 8

Task 6: Creating a ruleflow ....................... 10

Task 7: Writing action rules ....................... 11

Task 8: Preparing for rule execution .......... 13

Step 1: Create a RuleApp project .......... 13

Step 2: Deploy the XOM and RuleApp to zRule Execution Server for z/OS ........ 13

Step 3: View the deployed artifacts .......... 14

Task 9: Building your COBOL application ...... 14

Task 10: Executing the COBOL program ...... 15

Tutorial: Sharing a BOM between Java and COBOL applications ................................. 15

Before you start .................................. 16

Task 1: Importing an existing rule project .... 17

Task 2: Configuring the BOM for COBOL copybook generation .............................. 18

Task 3: Creating a RuleApp project ............ 19

Task 4: Deploying the XOM and RuleApp ........ 20

Task 5: Building your COBOL application ..... 20

Task 6: Executing the COBOL application ...... 21

Tutorial: Generating COBOL code from a copybook-based rule project ....................... 21

Before you start .................................. 22

Task 1: Importing an existing rule project .... 24

Task 2: Generating a XOM from a COBOL copybook ............................................. 25

Task 3: Creating a BOM from the COBOL XOM 25

Task 4: Declaring ruleset parameters .......... 26

Task 5: Implementing BOM methods in COBOL 27

Task 6: Creating a ruleflow ....................... 30

Task 7: Writing action rules ....................... 31

Task 8: Preparing for rule execution .......... 32

Decision Center tutorials .......................... 34

**Notices** ............................................. 35

**Trademarks** ......................................... 37

**Index** .............................................. 39
Tutorials

Develop your technical skills with tutorials for Operational Decision Manager for z/OS.

Tutorials installed with Decision Server Rules on a distributed platform also demonstrate some features that are applicable to the Rule Execution Server run time on z/OS. The Decision Server Rules tutorials are pre-configured to run on the sample server. However, the same user tasks are relevant when you deploy your rules to z/OS, and administer Rule Execution Server on WebSphere® Application Server for z/OS.

Therefore, the following tutorials and tasks can be of interest to you.

**Note:** You can install the sample server on a distributed platform and follow the tutorials on that computer. You can deploy the tutorial applications on an instance of WebSphere Application Server for z/OS and follow the tasks that apply to both environments, or you can simply read the documentation to understand the tasks and find out what steps are involved.

<table>
<thead>
<tr>
<th>Tutorial</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing RuleApps</td>
<td>Task 1 (Steps 2-5), Task 3, and Task 4</td>
</tr>
<tr>
<td>Executing a hosted transparent decision service on Java</td>
<td>Tasks 2-7</td>
</tr>
<tr>
<td>Debugging a remote Rule Execution Server application</td>
<td>Tasks 1-4</td>
</tr>
</tbody>
</table>

The following tutorials are specific to Decision Server for z/OS.

**Tutorial: Creating a copybook-based ruleset for zRule Execution Server for z/OS**

In Rule Designer, you can use a COBOL copybook as the basis for your XOM, and deploy your ruleset to zRule Execution Server for z/OS so that you can execute it from a COBOL application.

**What you will learn**

A fictitious loan company uses a large COBOL application running on z/OS to validate loan applications. In this tutorial, the company decides to use business rules to test customer eligibility for their loans. In this tutorial, you learn how to perform these tasks:

- Import an existing rule project named reszminiloan
  - If you already have a rule project named reszminiloan in your workspace, either switch to a different workspace or choose a different name.
- Create a XOM from the sample COBOL copybook provided with this tutorial
- Create a BOM based on the XOM
- Add two rule packages: eligibility and validation
Add methods to the BOM so that you can specify conditions and actions in your rules, and map them to the XOM.

Add a ruleflow to control rule execution.

Write a set of rules.

Deploy your rules for rule execution in a COBOL application on z/OS.

**Time required**

This tutorial takes about one hour to complete.

**Before you start**

In this topic, find out what you need to know before you start the tutorial.

Rule Designer provides features that you can use to abstract COBOL data structures to a Business Object Model (BOM). With a BOM, you can write rules in more natural language than you can using the COBOL structures and naming directly.

A fictitious loan company uses a large COBOL application running on z/OS to validate loan applications. In this tutorial, the company decides to use business rules to test customer eligibility for their loans.

The company wants to share rules contained in the COBOL application with other business applications, and manage the changes made to these rules. The company decides to migrate the business logic to a BRMS by prioritizing the most commonly shared rules. The rules that are chosen define customer eligibility. This set of rules calculates the income verification and minimum credit amount. During execution, the rules pass a message to the main program to notify the result of the initial validation.

In a real life scenario, you identify or create a COBOL copybook containing the COBOL data items required for the business rules. You then use this copybook as a basis for creating your execution object model (XOM). In this tutorial, you use the sample COBOL copybook provided, HBRLDAT1.cpy.

The HBRLDAT1.cpy copybook contains the following data items:

```plaintext
01 Borrower.
   05 name PIC X(20).
   05 creditScore PIC 9(10).
   05 yearlyIncome PIC 9(10).
   05 age PIC 9(3).
      88 teenager VALUE 0 THRU 17.
      88 adult VALUE 18 THRU 60.
      88 retired VALUE 61 THRU 150.
01 Loan.
   05 amount PIC 9(10).
   05 yearlyInterestRate PIC 99.
   05 yearlyRepayment PIC 9(10).
   05 effectDate PIC X(8).
   05 approved PIC X.
   05 messageCount PIC 9(2).
   05 messages PIC X(60) OCCURS 0 TO 99 TIMES DEPENDING ON messageCount.
```

From this copybook, you will generate a ruleset to be deployed on zRule Execution Server for z/OS.
What you will learn

A fictitious loan company uses a large COBOL application running on z/OS to validate loan applications. In this tutorial, the company decides to use business rules to test customer eligibility for their loans. In this tutorial, you learn how to perform these tasks:

- Import an existing rule project named reszminiloan
  
  If you already have a rule project named reszminiloan in your workspace, either switch to a different workspace or choose a different name.
- Create a XOM from the sample COBOL copybook provided with this tutorial
- Create a BOM based on the XOM
- Add two rule packages: eligibility and validation
- Add methods to the BOM so that you can specify conditions and actions in your rules, and map them to the XOM
- Add a ruleflow to control rule execution
- Write a set of rules
- Deploy your rules for rule execution in a COBOL application on z/OS

Time required

This tutorial takes about one hour to complete.

Software requirements

To use this tutorial you must have Rule Designer installed on Windows or Linux. This tutorial uses Rule Designer on Windows.

You must also have an instance of zRule Execution Server for z/OS installed and configured on z/OS.

Note:

You can also write and manage rules using Decision Center. However, this tutorial does not include tasks using Decision Center.

What you need to know

To follow this tutorial, you must be familiar with certain aspects relating to your installation:

- You must have a basic understanding of the Enterprise COBOL programming language, know how to make calls from a COBOL program, and know how to build a COBOL application.
- You must know where to deploy your RuleApps. Your zRule Execution Server for z/OS system administrator has configured the location where your RuleApps and XOMs must be deployed, either on the file system or in a DB2® database. If you do not know the location, contact your system administrator.
- You must also be familiar with Rule Designer and understand some basic Business Rule Management System (BRMS) terms and concepts, such as:
  - Execution object model (XOM)
  - Business object model (BOM)
You can find all these terms in the Glossary.

For information about special considerations for rule execution in COBOL applications, see COBOL considerations for rule application development.

Important:

This tutorial is only available in American English. The rule project containing the rules to import in this tutorial is only provided in the en_US locale. If en_US is not your default locale, you can use the samples console shortcut:

From the Start menu, click All Programs > IBM > <package_group> > Sample Server > Samples Console (en_US).

Directory structure

All the files for this tutorial are contained in the following directory: <InstallDir>/studio/tutorials/cobol/cobolstartresz, where <InstallDir> is the Decision Server installation directory.

This tutorial provides a sample COBOL copybook named HBRLDAT1.cpy for you to use. The copybook is stored in:

- <InstallDir>/studio/tutorials/cobol/cobolstartresz/start/reszminiloan-execution/copy/HBRLDAT1.cpy.

Start this tutorial: "Task 1: Importing an existing rule project."

Task 1: Importing an existing rule project

To focus on the main purpose of the tutorial, you import an existing rule project rather than create one.

This tutorial provides an existing rule project, reszminiloan, which has a COBOL copybook from which you want to generate a ruleset to be deployed on zRule Execution Server for z/OS. If you already have a rule project with the same name, switch to a new workspace before starting this task.

To import an existing rule project:

1. If necessary, switch to a new workspace:
   a. In Rule Designer, on the File menu, click Switch Workspace > Other.
   b. On the Workspace Launcher page, in the Workspace field, change the name of the current workspace to a new workspace name.
   c. Click OK.
2. To import a rule project from the samples console, click Window > Open Perspective > Other.
3. In the Open Perspective dialog, click Samples Console, and then click OK.
The Samples and Tutorials view is displayed.

4. In this perspective, expand the Rule Designer section if necessary, and then click Tutorials > Cobol > Creating a copybook-based rule project for zRules Execution Server for Z.

5. Under start, click Import projects.

The reszminiloan rule project is added to the list of projects in the Rule Explorer.

Note:

By default, the rules in this tutorial are executed using the classic rule engine. If you want to use the decision engine instead, see Choosing a build mode and Decision engine.

**Task 2: Generating a XOM from a COBOL copybook**

To execute your rules in a COBOL application, you use a COBOL copybook as the basis for your XOM.

A COBOL XOM provides the necessary COBOL-to-Java mapping so that you can execute your rules in a COBOL application.

To generate a COBOL XOM:

1. In the Design part of the Rule Project Map, click Import XOM and then select COBOL Execution Object Model.

   Note: You use the Rule Project Map to view the status of goals and tasks in a rule project. To display it, select Window > Show View > Other > Rule Designer > Rule Project Map.

   Alternatively, right-click your rule project and then select Properties > COBOL Management > COBOL Execution Object Model.

2. On the Properties page, in the COBOL Execution Object Model pane, click Add.

3. On the Import COBOL XOM page, in the Execution Object Model Name field, type a name for the XOM. For example, rezminiloan-xom.

4. Click Add.

5. On the Select Cobol Copybook page, click File System and then navigate to the HBRLDAT1 copybook.

   See "Before you start" on page 2 for the copybook location.

6. Select HBRLDAT1.cpy, and then click Open.

7. In the Package name field, type a package name, for example, xom. Click OK.

8. On the Import COBOL XOM page, click Next to display a summary of the default Java types and class attributes derived from each COBOL item in the copybook.

9. Change approved from type String to boolean, using the type converter:
   a. Expand the Loan item.
   b. Right-click the row that contains the approved data item and then click Add converter.
   c. On the Configure Converter Settings page, select the boolean converter.
   d. For the boolean True value, type T and for the False value, type F.
   e. Click OK.
Attention:

If you change the default mappings and then subsequently reimport the XOM, you lose your configured values.

10. Click Finish to close the page and create the COBOL XOM.
    You see the new XOM on the Properties page in the COBOL Execution Object Model pane.

11. Click OK to close the Properties window.

Notice the following addition in the Rule Explorer view: a rezminiloan Java XOM project.

Task 3: Creating a BOM from the COBOL XOM

You create a BOM in Rule Designer based on your COBOL XOM.

In this task, you create a BOM in Rule Designer, based on the COBOL XOM you created in the previous task.

To create a BOM from the COBOL XOM:

1. In the Design part of the Rule Project map, click Create BOM.
   Alternatively, in the Rule Explorer view, select the bom folder in your rule project and then, on the File menu, click New > BOM Entry.

2. In the New BOM Entry wizard, in the Name field, accept the default name for the BOM entry: model.

3. Make sure that the option Create a BOM entry from a XOM is selected and then click Next.

4. On the New BOM Entry screen, in the Choose a XOM entry field, click Browse XOM, select platform:/rezminiloan, and then click OK.

5. Under Select classes, select the xom package.
   When you select the package, you automatically select all the classes it contains.

6. Click Next.
   On the BOM Verbalization page, select the All Methods check box.
   This option makes sure that all the methods are verbalized, in addition to the elements you already selected.

7. Click Finish.
   In the Rule Explorer view, the bom folder contains a new BOM entry.

8. In the Rule Explorer, double-click bom > model to open the BOM Editor.

9. In the BOM Editor, expand the xom package.
   You see that the BOM has two classes to match the Borrower and Loan classes in the XOM.

10. Double-click the Borrower class to view the default verbalization: borrower.
    The default verbalization for the Loan class is loan.

Note:

The default verbalization is in American English. If you are working in a localized version of Rule Designer, you can verbalize the Borrower and Loan classes in the language of your locale, using the Edit Term dialog.
To display the Edit Term dialog, in the Rule Explorer double-click the class or member you want to modify. In the Verbalization section of the BOM Editor, click Edit.

**Task 4: Declaring ruleset parameters**

Ruleset parameters define the data that is sent between the COBOL calling program and the RuleApp.

Ruleset parameters provide the means to exchange data between your COBOL application and the RuleApp. You define ruleset parameters by name, type, and direction.

In this tutorial, you want to make a decision on the status of a loan, so you create ruleset parameters for the Borrower and Loan classes. You must use the IN direction for the borrower parameter. The value of the borrower parameter is provided as input from the COBOL client application on execution. The direction for the loan parameter must be IN_OUT. The value of the loan parameter is set by the IN value passed by the client and then updated by the engine on the way out. The updated value is returned to the client.

**Note:**

You cannot use the OUT parameter direction with zRule Execution Server for z/OS.

To declare ruleset parameters:

1. In the **Design** part of the Rule Project map, click **Define parameters**.
   - Alternatively, in the Rule Explorer, right-click your rule project and then click **Properties**.
2. In the type filter pane of the Properties dialog, select **Ruleset Parameters**.
3. On the Ruleset Parameters page, select the **Enable type check for COBOL XOM** check box.
   - This selection filters the ruleset parameter types you can use for a rule project that has a COBOL XOM.
4. To define the borrower parameter, click **Add** and then change the default values as follows:
   a. In the **Name** column, type borrower.
   b. In the **Type** column, click the ... button to display the Types dialog, and then double-click the **Borrower** type in the **Matching types** box.
   - The xom.Borrower type is displayed in the **Type** column.
   c. In the **Direction** column, select the IN direction.
   d. In the **Verbalization** column, type the borrower.
5. To define the loan parameter, click **Add**.
   - Your ruleset parameter settings look like this:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Direction</th>
<th>Default Value</th>
<th>Verbalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>borrower</td>
<td>xom.Borrower</td>
<td>IN</td>
<td></td>
<td>the borrower</td>
</tr>
<tr>
<td>loan</td>
<td>xom.Loan</td>
<td>IN_OUT</td>
<td></td>
<td>the loan</td>
</tr>
</tbody>
</table>
```

   a. In the **Name** column, type loan.
   b. In the **Type** column, click the ... button to display the Types dialog, and then double-click the **Loan** type in the **Matching types** box.
The xom.Loan type is displayed in the **Type** column.

- In the **Direction** column, use the default **IN_OUT** direction.
- In the **Verbalization** column, type the loan.

6. Click **OK** to save your ruleset parameters.

**Task 5: Adding BOM methods and mapping them to the XOM**

You add BOM methods to a BOM class to help verbalize your rules. To keep the XOM synchronized with the BOM, you must map the new BOM methods to the XOM.

You use methods to specify conditions and actions in your rules. You create methods in Rule Designer.

When you add methods to the BOM, you create members in the BOM that are not defined in the XOM. This mismatch creates errors in your rule project. To avoid this mismatch, you map the new methods to the XOM, using the BOM Editor.

**Note:**

You must not make any changes directly in the XOM.

In this task, you add the following methods, verbalize them, and then apply BOM to XOM mapping.

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loan</td>
<td>public void addMessage(String msg)</td>
<td>Defines what is needed to pass information from the rules.</td>
</tr>
<tr>
<td>Loan</td>
<td>public void reject</td>
<td>Identifies whether the loan application was rejected.</td>
</tr>
<tr>
<td>Borrower</td>
<td>public long computeIncomeAfterTax</td>
<td>Computes the income of the borrower, after tax.</td>
</tr>
</tbody>
</table>

To add BOM methods and map them to the XOM:

1. Start by adding the addMessage method. In the Outline view, expand the model package and then double-click the Loan class.
2. On the Class page of the BOM editor, in the **Members** section, click **New**.
3. On the Member page of the New Member dialog, make the following entries.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Select <strong>Method</strong></td>
</tr>
<tr>
<td>Name</td>
<td>Type <strong>addMessage</strong></td>
</tr>
<tr>
<td>Method type</td>
<td>Type <strong>void</strong></td>
</tr>
</tbody>
</table>

4. On the same page, click **Add** and then complete the Method Argument dialog as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Type <strong>msg</strong></td>
</tr>
<tr>
<td>Type</td>
<td>Type <strong>java.lang.String</strong></td>
</tr>
</tbody>
</table>

5. Click **OK** to close the Method Argument dialog, and then click **Finish**.
On the Class page of the BOM editor, the Members list now includes the addMessage method.

6. Double-click the addMessage method, and in the Member Verbalization section of the BOM Editor, click Create to view the default verbalization:
   add {0} to the messages of {this}
   You are not going to change this verbalization.

7. Scroll down to the BOM to XOM mapping section of the BOM Editor and then click the arrow next to the Body field to open the BOM to XOM Mapping editor.

8. Type the following Java code:
   ```java
this.messages.add(msg);
   ```
   Save your work.

10. Add the reject method to the Loan class and then save it:
    On the Member page of the New Member dialog, make the following entries.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Select Method</td>
</tr>
<tr>
<td>Name</td>
<td>Type reject</td>
</tr>
<tr>
<td>Method type</td>
<td>Type void</td>
</tr>
</tbody>
</table>

You do not add parameters for this method.

The following table shows you what to enter for the verbalization and the BOM to XOM mapping.

<table>
<thead>
<tr>
<th>Verbalization</th>
<th>BOM to XOM mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept the default verbalization, reject {this}</td>
<td>this.approved = false;</td>
</tr>
</tbody>
</table>

11. Add the computeIncomeAfterTax method to the Borrower class and then save it:
    On the Member page of the New Member dialog, make the following entries.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Select Method</td>
</tr>
<tr>
<td>Name</td>
<td>Type computeIncomeAfterTax</td>
</tr>
<tr>
<td>Method type</td>
<td>Type long</td>
</tr>
</tbody>
</table>

You do not add parameters to this method.

The following table shows you what to enter for the verbalization and the BOM to XOM mapping.

<table>
<thead>
<tr>
<th>Verbalization</th>
<th>BOM to XOM mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change the verbalization to the income after tax of {this}</td>
<td>if(this.yearlyIncome &lt; 24000) return this.yearlyIncome; else return Math.round(this.yearlyIncome * 0.9);</td>
</tr>
</tbody>
</table>

12. In the Outline view, in the model package, expand the Borrower class to view the computeIncomeAfterTax method, and then expand the Loan class to view the addMessage and reject methods you added.
Task 6: Creating a ruleflow

You use ruleflows to control the execution flow of the rules.

Before writing the rules, you orchestrate how your rules will be executed. You control rule execution using ruleflows. It is not essential to use ruleflows, but doing so makes rule execution more efficient.

When defining the flow of execution, you organize your rules into packages that contain related rules. Before you create a ruleflow, therefore, you create the relevant packages. You then treat these rule packages as tasks in the ruleflow. In this task, you create a package related to validation, and another related to eligibility.

To create a ruleflow:
1. Start by creating two rule packages for storing your rules, one named eligibility and the other named validation.
   a. In Rule Designer, in the Orchestrate part of the Rule Project Map, click Add rule package.
      Alternatively, you can right-click your rule project rules folder in the Rule Explorer and then click New > Rule Package.
   b. In the New Rule Package wizard, in the Package field, type validation and then click Finish.
      The new validation rule package displays in the Rule Explorer.
   c. Create another package and name it eligibility.
      Your rule project now contains two packages for storing your rules.
2. To create the ruleflow, in the Rule Explorer, select your rule project and then click New Ruleflow on the Rule Designer toolbar.
3. In the New Ruleflow dialog, in the Name field, type mainflow and then click Finish.
   The Ruleflow Editor opens, displaying the Diagram page.
4. Click the start node in the Ruleflow palette and then click the Diagram page.
5. Click the Create a ruletask icon in the palette and then click the Diagram page.
   A new rule task is created.
6. Click the new rule task and then make the following entries on the Properties tab:
   • Change the ID to validation
   • For Rule Execution > Algorithm, select Sequential
7. On the Properties tab, click Rule Selection and then click Edit.
8. In the Select Rules dialog, use the arrow button to move the validation package to the side pane and then click OK.
9. Create a second rule task named eligibility and then set its properties as for the validation task.
10. Move the eligibility package to the side pane of the Select Rules dialog and then click OK.
11. Add an end node to the ruleflow.

12. Using the Create a transition icon in the palette, create the following transitions:
   - From the start point to the validation task
   - From the validation task to the eligibility task
   - From the eligibility task to the end point

13. Align the ruleflow by clicking the Layout All Nodes icon.
14. Save the ruleflow.
   Your ruleflow looks like this.

```
Task 7: Writing action rules

You write rules using dedicated editors and the Business Action Language (BAL).

In this task, you write five action rules and put them into the relevant package. You can write rules in Rule Designer, or in Decision Center. If you use Decision Center, make sure that you synchronize with Rule Designer before you do so.

You create the following rules in Rule Designer to the eligibility and validation packages:

<table>
<thead>
<tr>
<th>Package name</th>
<th>Rule name</th>
</tr>
</thead>
<tbody>
<tr>
<td>eligibility</td>
<td>MinimumCreditScore</td>
</tr>
<tr>
<td></td>
<td>MinimumAmount</td>
</tr>
<tr>
<td></td>
<td>MinimumNetIncome</td>
</tr>
<tr>
<td>validation</td>
<td>MaximumAgeReq</td>
</tr>
<tr>
<td></td>
<td>MaximumAmount</td>
</tr>
</tbody>
</table>
```
Note:

If you are working in a localized version of the product, make sure that you have verbalized the Borrower and Loan classes in the language for your locale before you start writing rules.

To create the action rules:

1. Create the MinimumCreditScore rule:
   a. In the rule project, right-click the eligibility package and then click **New > Action Rule**.
   b. In the **Name** field, type **MinimumCreditScore**.
   c. Click **Finish**.
      
      The new action rule displays in the Rule Explorer view and the Intellirule Editor opens. You can switch to the guided editor if you prefer.

2. Write the eligibility.MinimumCreditScore rule as follows:
   
   ```
   if
   the credit score of 'the borrower' is less than 200
   then
   add "Credit score below 200" to the messages of 'the loan';
   reject 'the loan';
   ```

3. In the eligibility package, add the following rules:

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinimumAmount</td>
<td>if the yearly repayment of 'the loan' is more than the yearly income of 'the borrower' * 0.3 then add &quot;Too big Debt-To-Income ratio&quot; to the messages of 'the loan'; reject 'the loan';</td>
</tr>
<tr>
<td>MinimumNetIncome</td>
<td>if the income after tax of 'the borrower' is less than 24000 then add &quot;The yearly income is lower than the basic request&quot; to the messages of 'the loan'; reject 'the loan';</td>
</tr>
</tbody>
</table>

4. In the validation package, add the following rules:

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaximumAgeReq</td>
<td>if the age of 'the borrower' is more than 65 then add &quot;The age exceeds the maximum.&quot; to the messages of 'the loan'; reject 'the loan';</td>
</tr>
<tr>
<td>MaximumAmount</td>
<td>if the amount of 'the loan' is more than 1000000 then add &quot;The loan cannot exceed 1000000&quot; to the messages of 'the loan'; reject 'the loan';</td>
</tr>
</tbody>
</table>

5. Save your work.
Task 8: Preparing for rule execution

To prepare for rule execution, you must create a RuleApp project and then deploy the RuleApp and the XOM to the zRule Execution Server for z/OS persistence on z/OS.

**Step 1: Create a RuleApp project**
You create a RuleApp project in Rule Designer.

You must create a RuleApp project to contain the rulesets that you want to execute.

To create a RuleApp project:
1. In Rule Designer, in the Deploy and Integrate part of the Rule Project Map, click Create RuleApp project.
   Alternatively, on the File menu, click New > Project, select RuleApp Project and then click Next.
2. In the New RuleApp Project wizard, in the Project name field, type this name for your rule project: reszminiloanApp.
3. Make sure that the Use default location check box is selected.
4. Click Next.
   Your rule project is shown as selected on the Rule Projects tab.

**Note:**
In this tutorial you include all the rules in the rule project. For large rule projects, you might want to use the ruleset extractor to extract only certain sets of rules. In this case, you click Add and then select the rule extractor option.

5. Click Finish.

The RuleApp project is created and is added to the list of projects in the Rule Explorer. The RuleApp project editor opens automatically.

Leave the RuleApp project editor open, as you use the Deploy option in the next step.

**Step 2: Deploy the XOM and RuleApp to zRule Execution Server for z/OS**
You deploy your artifacts so that you can execute the ruleset on z/OS.

To be able to execute your ruleset with zRule Execution Server for z/OS, you must deploy the COBOL XOM and your RuleApp to zRule Execution Server for z/OS.

To deploy the XOM and RuleApp:
1. In the RuleApp project editor, in the Deployment pane, click Deploy to deploy the RuleApp to one or more Rule Execution Server instances.
2. For the RuleApp, accept the default Increment RuleApp major version and then click Next.
3. Select Create a temporary Rule Execution Server configuration and enter the following details:
   - URL: http://your.server.address:PORT/res
4. Click Finish.

Your artifacts are deployed to zRule Execution Server for z/OS. You can now log in to the zRule Execution Server for z/OS console and view them.

**Step 3: View the deployed artifacts**

Use the zRule Execution Server for z/OS console to view the deployed RuleApp and XOM.

Log in to the zRule Execution Server for z/OS console and use the Navigator pane to view the deployed RuleApp and XOM.

To view your deployed artifacts:
1. Open the zRule Execution Server for z/OS console using the following URL:
   http://your.server.address:PORT/res
2. At the login prompt, enter the following login details:
   - Login: resAdmin
   - Password: resAdmin
3. Click the **Explorer** tab.
4. In the Navigator pane, click **Resources** to view the deployed XOM.
5. In the Navigator pane, click **RuleApps** to view the deployed RuleApp.

**Task 9: Building your COBOL application**

To execute your ruleset, you call the RuleApp from your COBOL application and pass the ruleset parameters to the rule engine running an instance of zRule Execution Server for z/OS.

This tutorial includes a code sample that you can use to call a RuleApp from a COBOL application. The following procedure explains each part of the client code and the steps to follow.

To build your COBOL application:
1. Copy in the zRule Execution Server for z/OS runtime copybooks:
   ```
   01 WS-REASON-CODES.
   COPY HBRC.
   COPY HBRWS.
   
   HBRC contains constants that you can use in your application.
   HBRWS contains the working storage structure used for calling HBRCCONN, HBRRULE, and HBRCdisc.
   
   2. Add the ruleset parameters:
   MOVE LENGTH OF Borrower to HBRA-RA-DATA-LENGTH(1)
   MOVE "borrower" to HBRA-RA-PARAMETER-NAME(1)
   SET HBRA-RA-DATA-ADDRESS(1) to address of Borrower
   MOVE LENGTH OF Loan TO HBRA-RA-DATA-LENGTH(2)
   COMPUTE WS-MAX-TABLE-LEN = LENGTH OF Messages * 99
   ADD WS-MAX-TABLE-LEN TO HBRA-RA-DATA-LENGTH(2)
   MOVE "loan" to HBRA-RA-PARAMETER-NAME(2)
   SET HBRA-RA-DATA-ADDRESS(2) to address of Loan
   
   3. Specify the ruleset to call:
   MOVE ">your_rule_project_name>" to HBRA-CONN-RULEAPP-PATH
   
   4. Connect to zRule Execution Server for z/OS:
   ```
call "HBRCONN" using
   HBRA-CONN-AREA
5. Call the ruleset and check the return codes:
   call "HBRRULE" using
      HBRA-CONN-AREA
      IF HBRA-CONN-COMPLETION-CODE = HBR-CC-OK THEN
        DISPLAY "Successful call"
      ELSE
        DISPLAY "Error "
        HBRA-CONN-COMPLETION-CODE
        HBRA-CONN-REASON-CODE
   END-IF
6. Disconnect from zRule Execution Server for z/OS:
   call "HBRDISC" using
      HBRA-CONN-AREA

You can find a sample program for the COBOL application in
<InstallDir>/studio/tutorials/cobol/cobolstartresz/answer/reszminiloan-
execution/cobol/MINITUT1.cbl, where <InstallDir> is the Decision Server
installation directory.

Task 10: Executing the COBOL program
You execute the COBOL program using JCL.

You can find a prebuilt version of the tutorial with your Decision Server
installation on z/OS:
- **++HBRWORKDS++.SHBRJCL(HBRMTUT1)**, the JCL used to execute the COBOL program
- **++HBRWORKDS++.SHBRCOBS(HBRMTUT1)**, the COBOL program used to invoke the
  rule execution
- **++HBRWORKDS++.SHBRCOBC(HBLRDLAT1)**, the COBOL copybook

To execute the program:
1. Submit the sample JCL.
2. Check the output.
   The output should look something like this:
   
   **************EXECUTION RESULT ***************
   Borrower Name : John
   Loan Approved?: F
   Reject messages:
   The age exceeds the maximum.
   ****************************

You have now completed the tutorial.

Tutorial: Sharing a BOM between Java and COBOL applications

This tutorial shows you how to use Rule Designer to configure a Java BOM for
COBOL structures so that you can deploy your ruleset to zRule Execution Server
for z/OS and execute it from a COBOL application.

What you will learn

In this tutorial, you develop the following skills:
- Importing a Java-based rule project into your workspace
- Enabling the business object model (BOM) for COBOL copybook generation
• Generating a COBOL copybook from the BOM
• Customizing the default Java to COBOL mapping
• Deploying your rules for rule execution in a COBOL application on z/OS.

Time required

This tutorial should take approximately 60 minutes to finish.

Before you start

Find out what you need to know before you start the tutorial.

Note: This tutorial is available only in American English. The rule project containing the rules to import in this tutorial is only provided in the en_US locale. If en_US is not your default locale, you can use the samples console shortcut to change the locale. From the Start menu, click All Programs > IBM > package_group > Sample Server > Samples Console (en_US).

The default package group is Operational Decision Manager 8.5.1.

A fictitious loan company wants to use business rules to validate loan applications and execute them in either a Java environment, or using a COBOL application running on z/OS.

The company uses business rules to define customer eligibility for loan applications. The rule projects that the company currently uses contain BOMs based on a Java XOM. They deploy the rules to Rule Execution Server on WebSphere Application Server for z/OS, to execute in a Java environment.

The company now wants to share the Java rule projects with COBOL applications running on z/OS, and manage the changes made to these rules. To share rules with COBOL applications, they must add the necessary COBOL structures to the BOM and generate a COBOL copybook. With these structures in the rule project, the company deploys the RuleApps to zRule Execution Server for z/OS so that the COBOL application can call the rulesets and execute the rules.

What you will learn

In this tutorial, you develop the following skills:
• Importing a Java-based rule project into your workspace
• Enabling the business object model (BOM) for COBOL copybook generation
• Generating a COBOL copybook from the BOM
• Customizing the default Java to COBOL mapping
• Deploying your rules for rule execution in a COBOL application on z/OS.

Time required

This tutorial should take approximately 60 minutes to finish.

Software requirements

You must have Rule Designer installed on Windows or Linux. This tutorial uses Rule Designer on Windows. You must also have an instance of zRule Execution Server for z/OS installed and configured on z/OS.
All the files for this tutorial are contained in the following directory: `<InstallDir>/studio/tutorials/cobol/bomstartrez`, where `<InstallDir>` is the Decision Server for z/OS installation directory. This tutorial provides a sample rule project named SharingZMiniloan for you to use. The rule project is stored in the following directory: `/studio/tutorials/cobol/bomstartrez/start`.

**What you need to know**

To follow this tutorial, you must be familiar with certain aspects relating to your installation:

- You must have a basic understanding of the Enterprise COBOL programming language, know how to make calls from a COBOL program, and know how to build a COBOL application.
- You must know where to deploy your RuleApps. Your zRule Execution Server for z/OS system administrator has configured the location where your RuleApps and XOMs must be deployed, either on the file system or in a DB2 database. If you do not know the location, contact your system administrator.
- You must also be familiar with Rule Designer and understand some basic Business Rule Management System (BRMS) terms and concepts such as:
  - Execution object model (XOM)
  - Business object model (BOM)
  - RuleApp
  - RuleApp archive
  - Ruleset
  - Ruleset parameter.

You can find these terms in the Glossary.

If you want to share rules between Java and COBOL applications, it is important to understand how Java structures map to COBOL structures, and any limitations that apply. For more information, see Guidelines for sharing a BOM and Handling data type differences between Java and COBOL in the distributed platform part of this information center.

**Directory structure**

All the files for this tutorial are contained in the following directory: `<InstallDir>/studio/tutorials/cobol/bomstartrez`, where `<InstallDir>` is the Decision Server installation directory.

This tutorial provides a sample rule project named SharingZMiniloan for you to use. The rule project is stored in the following directory:

- `<InstallDir>/studio/tutorials/cobol/bomstartrez/start`.

Start this tutorial: “Task 1: Importing an existing rule project.”

**Task 1: Importing an existing rule project**

To focus on the main purpose of the tutorial, you import an existing rule project rather than create one.

This tutorial provides an existing rule project, SharingZMiniloan, which has a BOM generated from a Java XOM. If you already have a rule project with the same name, switch to a new workspace before starting this task.
1. If necessary, switch to a new workspace:
   a. In Rule Designer, on the File menu, click Switch Workspace > Other.
   b. On the Workspace Launcher page, in the Workspace field, change the name of the current workspace to a new workspace name.
   c. Click OK.
2. To import a rule project from the samples console, click Window > Open Perspective > Other.
3. In the Open Perspective dialog, click Samples Console, and then click OK. The Samples and Tutorials view displays.
4. Expand the Rule Designer section if necessary, and then click Tutorials > COBOL > Creating a BOM-based ruleset for zRule Execution Server for z/OS.
5. Under start, click Import projects.

The SharingZMiniloan rule project is added to the list of projects in the Rule Explorer.

**Task 2: Configuring the BOM for COBOL copybook generation**

Before you can use a Java BOM to generate a COBOL copybook, you must make your BOM entries compatible with COBOL code. If they are not compatible, the copybook you generate from the BOM is invalid.

To configure the BOM for copybook generation:
1. In Rule Explorer, right-click the rule project SharingZMiniloan and then select Properties > COBOL Management > COBOL Enabled BOM.
2. On the COBOL Enabled BOM page, click Add.
3. In the Select BOM entry dialog, click model, and then click OK.

The table under "Configure type setting for COBOL items" shows the proposed mapping between the Java structures in the BOM and the COBOL structures:

<table>
<thead>
<tr>
<th>Name</th>
<th>Java Type</th>
<th>COBOL Picture / Date Format</th>
<th>Generated COBOL Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>xom.Borrower</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- age</td>
<td>int</td>
<td>$9(10)</td>
<td>age PIC $9(10).</td>
</tr>
<tr>
<td>- creditScore</td>
<td>long</td>
<td>$9(18)</td>
<td>creditScore PIC $9(18).</td>
</tr>
<tr>
<td>- name</td>
<td>java.lang.String</td>
<td>X(20)</td>
<td>name PIC X(20) VALUE SPACE.</td>
</tr>
<tr>
<td>- yearlyIncome</td>
<td>long</td>
<td>$9(18)</td>
<td>yearlyIncome PIC $9(18).</td>
</tr>
<tr>
<td>xom.Loan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- amount</td>
<td>long</td>
<td>$9(18)</td>
<td>amount PIC $9(18).</td>
</tr>
<tr>
<td>- approved</td>
<td>boolean</td>
<td>X</td>
<td>approved PIC X.88 BoolValue Value 'T'.</td>
</tr>
<tr>
<td>- effectDate</td>
<td>java.util.Date</td>
<td>9(8) [yyyyMMdd]</td>
<td>effectDate PIC 9(8).</td>
</tr>
<tr>
<td>- messages</td>
<td>java.util.List</td>
<td>X(20)</td>
<td>messages PIC X(20) OCCURS 10 TIMES.</td>
</tr>
<tr>
<td>- yearlyInterestRate</td>
<td>short</td>
<td>$9(5)</td>
<td>yearlyInterestRate PIC $9(5).</td>
</tr>
<tr>
<td>Name</td>
<td>Java Type</td>
<td>COBOL Picture / Date Format</td>
<td>Generated COBOL Item</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>-----------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>yearlyRepayment</td>
<td>long</td>
<td>S9(18)</td>
<td>yearlyRepayment PIC S9(18)</td>
</tr>
</tbody>
</table>

Change the default length for messages from X(20) to X(60).

4. Click Finish.

The COBOL enabled BOM is generated.

5. Back on the COBOL Enabled BOM page, click Manage.

6. On the Copybook Generation page, specify the name and path of the copybook to be generated.
   a. Click the default copybook name. A button with three dots appears.
   b. Click the button to display the Select COBOL Copybook page.
   c. Click Browse, and then specify the following copybook name and path:
      /SharingZMiniloan-execution/copy/HBRLDAT2.cpy in place of the default name and path.

7. Click OK.

8. Under the Top Level Object column of the Copybook Generation page, accept the Java types xom.Borrower and xom.Loan as the top level data items in the copybook.

9. Under the Item Name column, accept the names borrower and loan corresponding to the Java types.
   When you generate the COBOL copybook, these are the names given to the top level data items.

10. Click Next to move to a copybook preview.

11. Click Finish.

12. Click OK to close the Properties for SharingZMiniloan dialog.

**Task 3: Creating a RuleApp project**

You create a RuleApp project to contain the rulesets that you want to execute.

1. In Rule Designer, in the Deploy and Integrate part of the Rule Project Map, click Create RuleApp project.

2. In the New RuleApp Project wizard, in the Project name field, type this name for your rule project: SharingZMiniloan-App.

3. Make sure that the Use default location check box is selected.

4. Click Next.
   Your rule project is shown as selected on the Rule Projects tab.

   **Note:**

   In this tutorial, you include all the rules in the rule project. For large rule projects, you might want to use the ruleset extractor to extract only certain sets of rules. In this case, you click Add and then select the rule extractor option.

5. Click Finish.

The RuleApp project is created and is added to the list of projects in the Rule Explorer. The RuleApp project editor opens automatically.

Leave the RuleApp project editor open, as you use the Deploy option in the next step.
**Task 4: Deploying the XOM and RuleApp**

You deploy your artifacts so that you can execute the ruleset on z/OS.

To be able to execute your ruleset with zRule Execution Server for z/OS, you must deploy the COBOL XOM and your RuleApp to zRule Execution Server for z/OS.

1. In the RuleApp project editor, in the **Deployment** pane, click **Deploy a RuleApp to one or more Rule Execution Servers**.
2. For the RuleApp, accept the default **Increment RuleApp major version** and then click **Next**.
3. Select **Create a temporary Rule Execution Server configuration** and enter the following details:
   - **URL:** http://your.server.address:PORT/res
   - **Login:** resAdmin
   - **Password:** resAdmin
4. Click **Finish**.

Under the **Console** tab, you can check whether your artifacts have been successfully deployed to Rule Execution Server. If so, you can log in to the Rule Execution Server console and view them.

**Task 5: Building your COBOL application**

To execute your ruleset, you call the RuleApp from your COBOL application and pass the ruleset parameters to the rule engine that is running inside an instance of zRule Execution Server for z/OS.

This tutorial includes a code sample that you can use to call a RuleApp from a COBOL application. The following procedure explains each part of the client code and the steps to follow.

You can find the sample COBOL application in `<InstallDir>/studio/tutorials/cobol/bomstartrez/answer/SharingZMiniloan-execution/cobol/HBRMTUT2.COBOL` where `<InstallDir>` is the Decision Server installation directory.

1. Copy in the zRule Execution Server for z/OS runtime copybooks:
   ```
   01 WS-REASON-CODES.
   COPY HBRC.
   COPY HBRWS.
   
   HBRC contains constants that you can use in your application.
   HBRWS contains the working storage structure used for calling HBRCNN, HBRRULE, and HBRDISC.
   ```
2. Add the ruleset parameters:
   ```
   MOVE LENGTH OF Borrower to HBRA-RA-DATA-LENGTH(1)
   MOVE "Borrower" to HBRA-RA-PARAMETER-NAME(1)
   SET HBRA-RA-DATA-ADDRESS(1) to address of Borrower
   MOVE LENGTH OF Loan to HBRA-RA-DATA-LENGTH(2)
   MOVE "Loan" to HBRA-RA-PARAMETER-NAME(2)
   SET HBRA-RA-DATA-ADDRESS(2) to address of Loan
   ```
3. Specify the ruleset to call:
   ```
   MOVE 'your_rule_project_name' to HBRA-CONN-RULEAPP-PATH
   ```
4. Connect to zRule Execution Server for z/OS:
   ```
   call 'HBRCNN' using
   HBRA-CONN-AREA
   ```
5. Call the ruleset and check the return codes:
call 'HBRRULE' using 
    HBRA-CONN-AREA 
    IF HBRA-CONN-COMPLETION-CODE = HBR-CC-OK THEN 
    DISPLAY 'Successful call'
    ELSE 
    DISPLAY 'Error ' 
        HBRA-CONN-COMPLETION-CODE 
        HBRA-CONN-REASON-CODE 
    END-if 
6. Disconnect from zRule Execution Server for z/OS: 
call 'HBRDISC' using 
    HBRA-CONN-AREA 

**Task 6: Executing the COBOL application**

You execute the COBOL application using JCL.

You can find a prebuilt version of the tutorial with your Decision Server installation on z/OS:

- `++HBRWORKDS++.SHBRJCL(HBRMTUT2)`, the JCL used to execute the COBOL program
- `++HBRWORKDS++.SHBRCOBS(HBRMTUT2)`, the COBOL program used to invoke the rule execution
- `++HBRWORKDS++.SHBRCOBC(HBRLDAT2)`, the COBOL copybook

where `++HBRWORKDS++` represents the high-level qualifier assigned to the working data sets.

1. Submit the sample JCL.
2. Check the output.
   The output should look something like this:
   ```
   ******************EXECUTION RESULT ******************
   Borrower Name: John
   Loan Approved?: F
   Reject messages:
   The age exceeds the maximum.
   *******************************************
   ```

You have now completed the tutorial.

---

**Tutorial: Generating COBOL code from a copybook-based rule project**

In Rule Designer, you can write rules against native COBOL data structures, and generate COBOL code for rule execution.

**Note:** The COBOL code generation feature is deprecated in V8.5.1.

In V8.5.1, the COBOL code generation user interface is not displayed in Rule Designer by default. To complete this tutorial, you must enable the interface manually. To display the COBOL code generation user interface in Rule Designer, add the following parameter in the file `<InstallDir>/eclipse.ini`:

- `Dcobol.code.gen=true`

**What you will learn**

In this tutorial, you learn how to perform these tasks:

- Import an existing rule project named codegenminiloan
If you already have a rule project named codegenminiloan in your workspace, either switch to a different workspace or choose a different name.

- Add two rule packages: eligibility and validation
- Create a XOM from the sample COBOL copybook provided with this tutorial
- Create a BOM based on the XOM
- Add methods to the BOM so that you can specify conditions and actions in your rules, and implement them in COBOL
- Add a ruleflow to control rule execution
- Write a set of rules
- Generate COBOL code from the rule project, in preparation for rule execution in your Enterprise COBOL application

**Time required**

This tutorial takes about one hour to complete.

**Before you start**

In this topic, find out what you need to know before you start the tutorial.

Rule Designer provides features to bridge the gap between two distinct programming languages. COBOL is a structured programming language and Java is an object-oriented programming language.

A fictitious loan company uses a large COBOL application running on z/OS to validate loan applications. In this tutorial, the company decides to use business rules to test customer eligibility for their loans.

The company wants to share rules contained in the COBOL application with other business applications, and manage the changes made to these rules. The company decides to migrate the business logic to a BRMS by prioritizing the most commonly shared rules. The rules that are chosen define customer eligibility. This set of rules calculates the income verification and minimum credit amount. During execution, the rules pass a message to the main program to notify the result of the initial validation.

In a real life scenario, you identify or create a COBOL copybook containing the COBOL data items required for the business rules. You then use this copybook as a basis for creating your execution object model (XOM). In this tutorial, you use the sample COBOL copybook provided, miniloan.cpy.

The miniloan.cpy copybook contains the following data items:

```plaintext
01 Borrower.
   05 name PIC X(20).
   05 creditScore PIC 9(10).
   05 yearlyIncome PIC 9(10).
   05 age PIC 9(3).
     88 teenager VALUE 0 THRU 17.
     88 adult VALUE 18 THRU 60.
     88 retired VALUE 61 THRU 150.
01 Loan.
   05 amount PIC 9(10).
   05 yearlyInterestRate PIC 99.
   05 yearlyRepayment PIC 9(10).
   05 effectDate PIC X(8).
   05 approved PIC X.
   05 messageCount PIC 9(2).
```
From this copybook, you will generate native COBOL code to execute rules within an Enterprise COBOL application.

Start this tutorial: “Task 1: Importing an existing rule project” on page 24.

What you will learn

In this tutorial, you learn how to perform these tasks:

- Import an existing rule project named codegenminiloan
  If you already have a rule project named codegenminiloan in your workspace, either switch to a different workspace or choose a different name.
- Add two rule packages: eligibility and validation
- Create a XOM from the sample COBOL copybook provided with this tutorial
- Create a BOM based on the XOM
- Add methods to the BOM so that you can specify conditions and actions in your rules, and implement them in COBOL
- Add a ruleflow to control rule execution
- Write a set of rules
- Generate COBOL code from the rule project, in preparation for rule execution in your Enterprise COBOL application

Time required

This tutorial takes about one hour to complete.

Software requirements

To use this tutorial you must have Rule Designer installed on Windows or Linux.

Note:

You can also write and manage rules using Decision Center. However, this tutorial does not include tasks using Decision Center.

What you need to know

To follow this tutorial, you must have a basic understanding of the Enterprise COBOL programming language, know how to build a COBOL copybook, and how to compile, link, and execute a COBOL program. You must also be familiar with Rule Designer and understand some basic Business Rule Management System (BRMS) terms and concepts, such as:

- Execution object model (XOM)
- Business object model (BOM)
- Verbalization
- Ruleset
- Ruleset parameter
- Business Action Language
- Ruleflow
You can find all these terms in the [Glossary](#).

For information about special considerations for rule execution in COBOL applications, see [COBOL considerations for rule application development](#).

This tutorial is only available in American English. The rule project containing the rules to import in this tutorial is only provided in the en_US locale. If en_US is not your default locale, you can use the samples console shortcut:

From the Start menu, click All Programs > IBM > <package_group> > Sample Server > Samples Console (en_US).

**Directory structure**

All the files for this tutorial are contained in the following directory:
<InstallDir>/studio/tutorials/cobol/cobolcodegen, where <InstallDir> is the Decision Server installation directory.

This tutorial provides a sample COBOL copybook named miniloan.cpy for you to use. The copybook is stored in:

<InstallDir>/studio/tutorials/cobol/cobolcodegen/start/codegenminiloan-execution/copy/miniloan.cpy

Start this tutorial: ["Task 1: Importing an existing rule project."](#)

**Task 1: Importing an existing rule project**

To focus on the main purpose of the tutorial, you import an existing rule project rather than create one.

This tutorial provides an existing rule project, named codegenminiloan, which has a copybook from which you want to generate COBOL code. If you already have a rule project with the same name, switch to a new workspace before starting this task.

To import an existing rule project:

1. If necessary, switch to a new workspace:
   a. In Rule Designer, on the File menu, click Switch Workspace > Other.
   b. On the Workspace Launcher page, in the Workspace field, change the name of the current workspace to a new workspace name.
   c. Click OK.
2. To import a rule project from the samples console, click Window > Open Perspective > Other.
3. In the Open Perspective dialog, click Samples Console, and then click OK. The Samples and Tutorials view is displayed.
4. In this perspective, expand the Rule Designer section if necessary, and then click Tutorials > COBOL > Generating COBOL code from a copybook-based rule project.
5. Under start, click Import projects.

The codegenminiloan rule project is added to the list of projects in the Rule Explorer.
Task 2: Generating a XOM from a COBOL copybook

To execute your rules in a COBOL application, you use a COBOL copybook as the basis for your XOM.

A COBOL XOM provides the necessary COBOL to Java mapping so that you can execute your rules in a COBOL application.

To generate a COBOL XOM:
1. In the Design part of the Rule Project Map, click Import XOM and then select COBOL Execution Object Model.
   Alternatively, right-click your rule project and then select Properties > COBOL Management > COBOL Execution Object Model.
   On the Properties page, in the COBOL Execution Object Model pane, click Add.
2. On the Import COBOL XOM page, in the Execution Object Model Name field, type a name for the XOM. For example, codegenminiloan-xom.
3. Click Add.
4. In the COBOL Copybook page, click File System and then navigate to the miniloan copybook.
   See “Before you start” on page 22 for the copybook location.
5. Select miniloan.cpy, and then click OK.
6. Click OK.
7. On the Import COBOL XOM page, click Next to display a summary of the default Java types and class attributes derived from each COBOL item in the copybook.
8. Change approved from type String to boolean, using the type converter:
   a. Expand the Loan item.
   b. Right-click the row that contains the approved data item and then click Add converter.
   c. On the Configure Converter Settings page, select the boolean converter.
   d. For the boolean True value, type T and for the False value, type F.
   e. Click OK.

Attention:

If you change the default mappings and then subsequently reimport the XOM, you lose your configured values.

9. Click Finish to close the Configure COBOL XOM Mapping page and create the COBOL XOM.
   You see the new XOM on the Properties page in the COBOL Execution Object Model window.
10. Click OK to close the Properties window.

Notice the following addition in the Rule Explorer view: a miniloan Java XOM project.

Task 3: Creating a BOM from the COBOL XOM

You create a BOM in Rule Designer based on your COBOL XOM.
In this task, you create a BOM in Rule Designer, based on the COBOL XOM you created in the previous task.

To create a BOM from the COBOL XOM:
1. In the Design part of the Rule Project map, click Create BOM.
   Alternatively, in the Rule Explorer view, select the bom folder in your rule project and then, on the File menu, click New > BOM Entry.
2. In the New BOM Entry wizard, in the Name field, accept the default name for the BOM entry: model.
3. Make sure that the option Create a BOM entry from a XOM is selected and then click Next.
4. On the New BOM Entry screen, in the Choose a XOM entry field, click Browse XOM, select codegenminiloan-xom, and then click OK.
5. Under Select classes, select the xom package.
   When you select the package, you automatically select all the classes it contains.
6. Click Next.
   On the BOM verbalization page, select the All Methods check box.
   This option makes sure that all the methods are verbalized, in addition to the elements you already selected.
7. Click Finish.
   In the Rule Explorer view, the bom folder contains a new BOM entry.
8. In the Rule Explorer, double-click bom > model to open the BOM Editor.
9. In the BOM Editor, expand the xom package.
   You see that the BOM has two classes to match the Borrower and Loan classes in the XOM.
10. Double-click the Borrower class to view the default verbalization: borrower.
    The default verbalization for the Loan class is loan.

   **Note:**

   The default verbalization is in American English. If you are working in a localized version of Rule Designer, you can verbalize the Borrower and Loan classes in the language of your locale, using the Edit Term dialog.

   To display the Edit Term dialog, in the Rule Explorer double-click the class or member you want to modify. In the Verbalization section of the BOM Editor, click Edit.

**Task 4: Declaring ruleset parameters**

Ruleset parameters define the data that is sent between the generated COBOL program and the COBOL calling program.

Ruleset parameters provide the means to exchange data between the generated COBOL program and the calling application. You define ruleset parameters by name, type, and direction.

In this tutorial, you make a decision on the status of a loan, so you create ruleset parameters for the Borrower and Loan classes. You must use the IN_OUT direction for both parameters. The value of the IN_OUT parameter is provided as input to the
calling application on execution. It can be modified by the calling application and provided as output at execution completion.

Note:

You can use only the IN_OUT parameter for COBOL code generation.

To declare ruleset parameters:
1. In the Design part of the Rule Project map, click Define parameters. Alternatively, in the Rule Explorer, right-click your rule project and then click Properties.
2. In the type filter pane of the Properties dialog, select Ruleset Parameters.
3. On the Ruleset Parameters page, select the Enable type check for COBOL XOM check box.

This selection filters the ruleset parameter types you can use for a rule project that has a COBOL XOM.
4. To define the borrower parameter, click Add and then change the default values as follows:
   a. In the Name column, type borrower.
   b. In the Type column, click the ... button to display the Types dialog, and then double-click the Borrower type in the Matching types box.

   The xom.Borrower type is displayed in the Type column.
   c. In the Direction column, use the default IN-OUT direction.
   d. In the Verbalization column, type the borrower.
5. To define the loan parameter, click Add.

   Your ruleset parameter settings look like this:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Direction</th>
<th>Default Value</th>
<th>Verbalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>borrower</td>
<td>xom.Borrower</td>
<td>IN_OUT</td>
<td></td>
<td>the borrower</td>
</tr>
<tr>
<td>loan</td>
<td>xom.Loan</td>
<td>IN_OUT</td>
<td></td>
<td>the loan</td>
</tr>
</tbody>
</table>

   a. In the Name column, type loan.
   b. In the Type column, click the ... button to display the Types dialog, and then double-click the Loan type in the Matching types box.

   The xom.Loan type is displayed in the Type column.
   c. In the Direction column, keep the default IN_OUT direction.
   d. In the Verbalization column, type the loan.
6. Click OK to save.

**Task 5: Implementing BOM methods in COBOL**

You add BOM methods to a BOM class to help verbalize your rules. You then map them to the XOM to keep the XOM synchronized with the BOM, and implement the methods in COBOL so that the generated COBOL program can call the methods.

You use methods to specify conditions and actions in your rules. You create methods in Rule Designer. When you add methods to the BOM, you create members in the BOM that are not defined in the XOM. This mismatch creates errors in your rule project. To avoid this mismatch, you map the new methods to the XOM, using the BOM Editor.
Note:

You do not make any changes directly in the XOM.

You must also implement the methods in COBOL so that, when you generate the COBOL program to execute the rules, the generated COBOL program can call the methods.

In this task, you add the following methods to your classes, verbalize them, map them to the XOM, and then implement them in COBOL. The order in which you map the methods or implement them in COBOL is not important.

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loan</td>
<td>public void addMessage(String msg)</td>
<td>Defines what is needed to pass information from the rules.</td>
</tr>
<tr>
<td>Loan</td>
<td>public void reject</td>
<td>Identifies whether or not the loan application was rejected.</td>
</tr>
<tr>
<td>Borrower</td>
<td>public long computeIncomeAfterTax</td>
<td>Computes the net income of the borrower.</td>
</tr>
</tbody>
</table>

To implement COBOL methods:
1. Start by adding the addMessage method. In the Rule Explorer, expand the BOM model and then double-click the Loan class.
2. On the Class page of the BOM editor, in the Members section, click New.
3. On the Member page of the New Member dialog, make the following entries.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Select Method</td>
</tr>
<tr>
<td>Name</td>
<td>Type addMessage</td>
</tr>
<tr>
<td>Method type</td>
<td>Type void</td>
</tr>
</tbody>
</table>

4. In the New Member dialog, on the Member page, click Add and then complete the Method Argument dialog as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Type msg</td>
</tr>
<tr>
<td>Method type</td>
<td>Type java.lang.String</td>
</tr>
</tbody>
</table>

5. Click OK.
6. Click Finish to close the New Member dialog.
   On the Class page of the BOM editor, the Members list now includes the addMessage method.
7. Double-click the addMessage method, and in the Member Verbalization section of the BOM Editor, click Create to view the default verbalization: add {0} to the messages of {this}
   You are not going to change this verbalization.
8. Scroll down to the BOM to XOM mapping section of the BOM Editor and then type the following Java code:
   this.messages.add(msg);
9. In the COBOL Method section, click Create a mapped method.
   This option generates the COBOL data type definitions for the method and displays them in the Mapped Method table.

10. Change the default length for msg from X(20) to X(60).

11. In the COBOL Method Body Editor, type the following COBOL code to implement the method.
    

Computing messageCount = messageCount + 1
MOVE {msg} TO messages(messageCount)

12. Save your work.

13. Add the reject method to the Loan class and then save it:
    On the Member page of the New Member dialog, make the following entries.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Select Method</td>
</tr>
<tr>
<td>Name</td>
<td>Type reject</td>
</tr>
<tr>
<td>Method type</td>
<td>Type void</td>
</tr>
</tbody>
</table>

You do not add parameters to this method.

The following table shows you what to enter for the verbalization, BOM to XOM mapping, and COBOL implementation.

<table>
<thead>
<tr>
<th>Verbalization</th>
<th>BOM to XOM mapping</th>
<th>COBOL method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept the default verbalization: reject {this}.</td>
<td>Type the following code: this.approved = false;</td>
<td>Implement a mapped method: MOVE &quot;F&quot; TO approved approved</td>
</tr>
</tbody>
</table>

14. Add the computeIncomeAfterTax method to the Borrower class and then save it:
    On the Member page of the New Member dialog, make the following entries.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Select Method</td>
</tr>
<tr>
<td>Name</td>
<td>Type computeIncomeAfterTax</td>
</tr>
<tr>
<td>Method type</td>
<td>Type long</td>
</tr>
</tbody>
</table>

You do not add parameters to this method.

The following table shows you what to enter for the verbalization, BOM to XOM mapping, and COBOL implementation.

<table>
<thead>
<tr>
<th>Verbalization</th>
<th>BOM to XOM mapping</th>
<th>COBOL method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change the verbalization to the income after tax of {this}.</td>
<td>Type the following code: if(this.yearlyIncome &lt; 24000) return this.yearlyIncome; else return Math.round(this.yearlyIncome * 0.9);</td>
<td>Implement a mapped method: MOVE 0 TO {return} IF yearlyIncome &lt; 24000 MOVE yearlyIncome TO {return} ELSE COMPUTE {return} = yearlyIncome * 0.9 END-IF</td>
</tr>
</tbody>
</table>

15. Save your work.

16. In the Outline view, in the model package, expand the Borrower class to view the computeIncomeAfterTax method, and then expand the Loan class to view the addMessage and reject methods you added.
Task 6: Creating a ruleflow

You use ruleflows to control the execution flow of the rules.

Before writing the rules, you orchestrate how your rules will be executed. You control rule execution using ruleflows. It is not essential to use ruleflows, but doing so makes rule execution more efficient.

When defining the flow of execution, you organize your rules into packages that contain related rules. Before you create a ruleflow, therefore, you create the relevant packages. You then treat these rule packages as tasks in the ruleflow. In this task, you create a package related to validation, and another related to eligibility.

To create a ruleflow:

1. Start by creating two rule packages for storing your rules, one named eligibility and the other named validation.
   a. In Rule Designer, in the Orchestrate part of the Rule Project Map, click Add rule package.
      Alternatively, you can right-click your rule project rules folder in the Rule Explorer and then click New > Rule Package.
   b. In the New Rule Package wizard, in the Package field, type validation and then click Finish.
      The new validation rule package displays in the Rule Explorer.
   c. Create another package and name it eligibility.

   Your rule project now contains two packages for storing your rules.

2. To create the ruleflow, in the Rule Explorer, select your rule project and then click New Ruleflow on the Rule Designer toolbar.

3. In the New Ruleflow dialog, in the Name field, type mainflow and then click Finish.
   The Ruleflow Editor opens, displaying the Diagram page.

4. Click the start node in the Ruleflow palette and then click the Diagram page.

5. Click the Create a rule task icon in the palette and then click the Diagram page.
   A new rule task is created.

6. Click the new rule task and then make the following entries on the Properties tab:
   - Change the ID to validation
   - For Rule Execution > Algorithm, select Sequential

7. On the Properties tab, click Rule Selection and then click Edit.

8. In the Select Rules dialog, use the arrow button to move the validation package to the side pane and then click OK.

9. Create a second rule task named eligibility and then set its properties as for the validation task.

10. Move the eligibility package to the side pane of the Select Rules dialog and then click OK.
11. Add an end node to the ruleflow.

12. Using the Create a transition icon in the palette, create the following transitions:
   a. From the start point to the validation task
   b. From the validation task to the eligibility task
   c. From the eligibility task to the end point

13. Align the ruleflow by clicking the Layout All Nodes icon.
14. Save the ruleflow.
   Your ruleflow looks like this.

![Ruleflow Diagram]

**Task 7: Writing action rules**

You write rules using dedicated editors and the Business Action Language (BAL).

In this task, you write five action rules and put them into the relevant package. You can write rules in Rule Designer, or in Decision Center. If you use Decision Center, make sure that you synchronize with Rule Designer before you do so.

You create the following rules in Rule Designer to the eligibility and validation packages:

<table>
<thead>
<tr>
<th>Package name</th>
<th>Rule name</th>
</tr>
</thead>
<tbody>
<tr>
<td>eligibility</td>
<td>MinimumCreditScore</td>
</tr>
<tr>
<td></td>
<td>MinimumAmount</td>
</tr>
<tr>
<td></td>
<td>MinimumNetIncome</td>
</tr>
<tr>
<td>validation</td>
<td>MaximumAgeReq</td>
</tr>
<tr>
<td></td>
<td>MaximumAmount</td>
</tr>
</tbody>
</table>

Tutorials 31
Note:

If you are working in a localized version of the product, make sure that you have verbalized the Borrower and Loan classes in the language for your locale before you start writing rules.

To create the action rules:
1. Create the MinimumCreditScore rule:
   a. In the rule project, right-click the eligibility package and then click New > Action Rule.
   b. In the Name field, type MinimumCreditScore.
   c. Click Finish.
      The new action rule displays in the Rule Explorer view and the Intellirule Editor opens. You can switch to the guided editor if you prefer.
2. Write the eligibility.MinimumCreditScore rule as follows:
   if the credit score of 'the borrower' is less than 200
   then add "Credit score below 200" to the messages of 'the loan';
   reject 'the loan';
3. In the eligibility package, add the following rules:

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
</tr>
</thead>
</table>
| MinimumAmount | if the yearly repayment of 'the loan' is more than the yearly income of 'the borrower' * 0.3
               then add "Too big Debt-To-Income ratio" to the messages of 'the loan';
               reject 'the loan';                                           |
| MinimumNetIncome | if the income after tax of 'the borrower' is less than 24000
                  then add "The yearly income is lower than the basic request" to the messages of 'the loan';
                  reject 'the loan';                                   |
4. In the validation package, add the following rules:

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
</tr>
</thead>
</table>
| MaximumAgeReq | if the age of 'the borrower' is more than 65
               then add "The age exceeds the maximum." to the messages of 'the loan';
               reject 'the loan';                                           |
| MaximumAmount | if the amount of 'the loan' is more than 1000000
                 then add "The loan cannot exceed 1000000" to the messages of 'the loan';
                 reject 'the loan';                                      |
5. Save your work.

Related information:
- Rule authoring for COBOL compatibility

Task 8: Preparing for rule execution

To prepare for rule execution, you generate the COBOL program for rule execution.
In this task, you generate the COBOL code that will be used to execute the rules within an Enterprise COBOL application. This task describes how to generate the code in Rule Designer, but you can use Decision Center, or you can use an Ant task. For information about generating the COBOL code using Decision Center or Ant, see Generating COBOL code and Generating COBOL code using Ant.

Note: These links point to the distributed platform part of this information center.

You can use the CICS® usage option to generate a COBOL program that can run with CICS channels and containers. You configure the CICS channel and container names in Rule Designer.

This task describes the basic steps for generating the COBOL code and does not cover the CICS usage option. For information about configuring the generated code for CICS, see Generating COBOL code in Rule Designer in the distributed platform part of this information center.

To generate the COBOL code in Rule Designer:

1. In the Rule Explorer, right-click the codegenminiloan project and then click COBOL Code Generation > Generate COBOL Code.
2. In the Generate COBOL Code wizard, in the COBOL Program ID field, type minirule.
3. In the Output file field, specify where you want to store the generated COBOL code:
   
   - To store the code in your workspace, click Workspace and then, on the Save As page, select the parent folder and type a filename.
   
   - To store the code in your file system, click File System, navigate to the required directory, type a filename in the Filename field and then click Open.
     For example: <WorkDir>/codegenminiloan/batch/minicbl.cbl, where <WorkDir> is your working directory.
4. Click Finish.
   
   The COBOL code generator generates a COBOL program containing your rule project rulesets, converted to COBOL code.
5. At the COBOL code generated successfully prompt, click OK.
6. Navigate to your output file and have a look at the code you have generated.
   
   This tutorial provides a file containing a sample COBOL program that was generated against the codegenminiloan data used in this tutorial.
7. If you want to compare your output file with the sample output file, open the sample file in:
   
   <InstallDir>/tutorials/cobol/cobolcodegen/answer/codegenminiloan-execution/cobol/minirule.cbl, where <InstallDir> is the Rule Designer installation directory.

Note:

The sample output file relates only to en_US locale. Omit this step if you are working in a different locale.
In a real life scenario, you pass the COBOL program to the z/OS application and compile it with the calling program. It is then ready for testing and execution. When you execute the program generated from this tutorial, you get the following result:

********** EXECUTION RESULT **********
Borrower Name : John
Loan Approved?: F
Reject messages:
The age exceeds the maximum.
********** EXECUTION RESULT **********

The main calling program for this tutorial is minimain.cbl. You can see this file in <InstallDir>/tutorials/cobol/cobolcodegen/answer/codegenminiloan-execution/cobol/minimain.cbl.

You have now completed the tutorial.

### Decision Center tutorials

Develop your skills with tutorials for Decision Center.

Tutorials installed with Decision Center on a distributed platform demonstrate features that are also applicable on z/OS. The Decision Center tutorials are pre-configured to run on the sample server. However, the same user tasks are relevant when you author and test your rules on z/OS, and administer Decision Center on WebSphere Application Server for z/OS.

Therefore, the following tutorials and tasks can be of interest to you.

**Note:** You can install the sample server on a distributed platform and follow the tutorials on that computer. You can deploy the tutorial applications on an instance of WebSphere Application Server for z/OS and follow the tasks that apply to both environments, or you can simply read the documentation to understand the tasks and to find out what steps are involved.

<table>
<thead>
<tr>
<th>Table 2. Tutorials under Decision Center that are also relevant on z/OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorial</td>
</tr>
<tr>
<td>Editing action rules and decision tables</td>
</tr>
<tr>
<td>Using smart folders and queries</td>
</tr>
<tr>
<td>Analyzing rules</td>
</tr>
<tr>
<td>Testing a set of rules</td>
</tr>
<tr>
<td>Managing releases</td>
</tr>
<tr>
<td>Implementing permissions</td>
</tr>
</tbody>
</table>
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England  SO21 2JN

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those
websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England  SO21 2JN

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or
imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_.

### Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml).

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Index

A
  action rule  11, 31

B
  BOM
    creating  6, 26
  BOM methods
    implementing  8
    implementing in COBOL  27
  BOM to XOM mapping  8, 27
  business object model (BOM)
    creating from COBOL XOM  6, 26

C
  COBOL application
    executing the ruleset  14, 20
  COBOL application execution  21
  COBOL code generation  33
  COBOL program execution  15
  COBOL XOM
    from COBOL copybook  5, 25

D
  declaring
    ruleset parameters  7, 26
  deploying artifacts  13, 20

E
  executing a COBOL application  21
  executing a COBOL program  15
  executing a ruleset  14, 20

G
  generating COBOL code  33

M
  marshaller
    deploying  20

R
  rule execution  13
    creating a RuleApp  13
    deploying artifacts  13
    viewing deployed artifacts  14
  rule package  11, 31
  RuleApp project
    deploying  20
  RuleApp, creating  13
  ruleflows
    creating  10, 30
  ruleset execution
    COBOL application  14, 20
  ruleset parameters
    declaring, in tutorial  7, 26
  viewing deployed artifacts  14

X
  XOM
    deploying  20

© Copyright IBM Corp. 2008, 2012