Creating and deploying business rules
Using WebSphere Integration Developer and WebSphere Process Server

Skill Level: Intermediate

Neil Kolban (kolban@us.ibm.com)
IT Specialist
IBM

11 Oct 2006

This tutorial shows you how to use WebSphere® Integration Developer to create and deploy a solution that uses business rules, and then test that solution in WebSphere Process Server. At the conclusion of this tutorial, you will have a working set of business rules deployed.

Section 1. Before you start

This tutorial illustrates the functions of WebSphere Process Server business rules. Business rules externalize simple algorithms and values that are used to affect the operation of a business process. These algorithms and values will change over time. They should not be embedded in code, which would make them difficult to change.

In this scenario, you are responsible for implementing a Frequent Flyer tracking system for a major airline. Whenever a passenger flies with this airline, they are awarded Frequent Flyer miles based on the actual miles flown. These miles are used to calculate redeemable awards. In addition, the passenger is given extra bonus miles based on their status in the Frequent Flyer program. There are three privileged statuses:

• Gold
• Silver
• Platinum

For each of these, a different bonus mileage is applied:
• Gold = 25%
• Silver = 50%
• Platinum = 100%

Our business leaders selectively add new status values as well as change the bonus award values from time to time. These are the variables in our business rules. Your task is to develop a service component that both implements and externalizes the requirement to quickly change the rules as the market changes. When completed, the service component looks like the diagram in Figure 1.

![Figure 1. Logical function of rules](image)

**Section 2. Building empty rules**

Before describing the details of the rules, you must first create the artifacts in WebSphere Integration Developer that will be used to hold their content. Within WebSphere Integration Developer, you first create a module (also called a project) that will be used to house all the other components that you will construct for our solution.

Within the module, you can then create an interface definition that describes how the business rules will look from a caller's perspective. The interface has an operation that you can use to calculate the mileage bonus.

With the interface definition created, you then construct both a Rule Set and Rule Group that will be used to reference the Rule Set. The Rule Group will expose the interface definition.

**Creating the module**

Now let's create a new module to hold our artifacts.

1. From the Business Integration view, select **New => Module** to launch WebSphere Integration Developer to create a new module with an empty workspace.

**Figure 2. Business Integration view shows how to create a module**
2. Enter MileageBonus in the New Module Name field. Every module within a WebSphere Integration Developer workspace must have a unique name.

**Figure 3. New Module wizard displays MileageBonus in Module Name field**

---

**Section 3. Creating the interface definition**

Now, let's create a new interface definition. Our business rule has an interface which, when called, returns the mileage bonus.
1. Right-click on the Interfaces folder contained within the Business Integration view to create the new Interface definition.

2. From the menu, select **New => Interface**.

   **Figure 4. Selecting New => Interface from the Business Integration view**

3. Interfaces have names associated with them. Enter **CalculateMileageBonus** in the Name field on the New Interface Wizard. Once entered, press **Finish** to complete the task.

   **Figure 5. Enter the name of the new interface**

4. In this example, the interface exposes a single operation called
calculateMileageBonus. When called, it requires two parameters. One parameter is the passenger's status (one of none, gold, silver, or platinum) and the other is the base mileage of the flight. The return from the operation is the mileage to be associated with the frequent flyer points adjusted, based upon the bonus business rules in effect. Add the operation and parameters to achieve the definition shown in Figure 6 by adding a two way operation called calculateMileageBonus as well as the illustrated parameters.

**Figure 6. Enter the details of the interface’s operation**

5. Now that the interface has been fully described, save and close the editor. Clicking on the **Close** button on the tab prompts you to save your work as well as close the editor. You will see the new interface definition in the Business Integration view.

**Constructing a rule set**

A rule set is a description of a set of rules to be executed. It is now time for you to start the construction of these rules.

1. A rule set is a set of business rules that are executed to calculate and return some business values. You need to define a new rule set at this point. This is the core part of the exercise. From the Business Integration view, right click on the Rules folder and from the menu, select **New => Ruleset**.

**Figure 7. Create a new ruleset from the Business Integration view**
2. Define a name for the new rule set, such as `MileageBonusRuleSet` in the Name field of the New Rule Set wizard. When completed, press the Next button to move on to the next step.

**Figure 8. the name of the new rule set**

---

**Constructing a rule group**

Rule sets do not exist in isolation. You must contain them within a rule group. It is the rule group that exposes the interface and determines which set of rules should be executed.
1. A rule set cannot exist in a vacuum. You must associate it with a rule group. A rule group names the rules that will be executed at different times. You can create that rule group at the same time as the rule set. In the New Rule Set wizard, press the **New** button on the Rule Group row to create a new rule group.

*Figure 9. Creating a new Rule Group*

2. A new wizard is started that walks you through the creation of a new rule group. Enter **MileageBonusRuleGroup** in the Name field of the New Rule Group wizard. Once entered, press the **Next** button to continue.

*Figure 10. Naming the new Rule Group*
3. When a rule group is invoked at runtime, it is expected to execute business rules to derive an answer. The rule group is exposed to its callers through an interface, which must be selected in the Select an Interface dialog. Use the pull-down to select the Interface definition that you created earlier (CalculateMileageBonus). Once the interface has been selected, press the Finish button to conclude the creation of the new rule group.

Figure 11. Selecting the interface for the Rule Group
4. You can now complete the new rule set. It has been associated with a rule group as well as the interface and operation for the rule set. Press the **Finish** button to complete this task. After pressing the **Finish** button, you are brought into the Rule Set editor.

**Figure 12. Completing the creation of the Rule Set**
Section 4. Declaring the rules

At this stage, you have completed the basic setup of the environment for building our rules. You have created:

- An interface
- A rule set
- A rule group

In the next steps, you will populate the rule set to describe the business logic you wish to represent through these rules.

1. A template is a pattern for creating instances of new rules. You are going to create a rule for each of the different classes of flyer status so you need to create a template that can be used for each. Press the button as shown in Figure 13 to create a new definition of a template.

   **Figure 13. Creating a new definition of a template**
2. When the definition of the template has been started, you will define the parameters that will differentiate one instance of a template from another. You must add two parameters. Click the Add Parameters button twice as shown in Figure 14.

Figure 14. Adding template parameters

3. When the parameters are added, they initially have no names or data types associated with them. Rename them to parmFlyerStatus of type String and parmMileageBonus of type int. The first parameter is used to indicate the flyer's status, and the second is the bonus mileage percentage. Refer to Figure 15 when setting these values.

Figure 15. Setting variable names and data types
4. A presentation string is the text shown to end users when setting rule instances. Set the presentation string value as:

If the passenger is parmFlyerStatus then the bonus is parmMileageBonus%

Note that parmFlyerStatus and parmMileageBonus are variables and not plain text strings. Figure 16 shows the value having been entered. Note the location of the two parameters in the presentation string text.

**Figure 16. Setting the presentation string**

5. Most rules have an *If* condition associated with them. This is a boolean valued expression which, if true, causes the corresponding action to be executed. If false, it causes the action to be skipped. Now you want to set the logical condition that causes the action to be executed. Set the *If* expression to be true if the flyer’s status, as submitted in the interface’s operation call, matches the value supplied on the parameter of the template. The expression is:

flyerStatus = parmFlyerStatus

This is illustrated in Figure 17.

**Figure 17. Define the condition**
6. Variables within a rule set have scope across all the rules contained within the set. You will use a variable for the duration of the rules that will temporarily hold the bonus percentage value. This variable may be changed by an individual rule during processing and will be used to calculate the final response returned to the caller. Press the "+" button to create a new variable as shown in Figure 18.

Figure 18. Creating a new variable

7. You must now set the name and data type of the newly created variable. Change the name to `bonusPercentage` and the data type to `int`. The variable holds the percentage bonus awarded to this flyer based upon their status.

Figure 19. Setting the variable name and type

8. You are now ready to supply the action statement that is executed if the expression for the rule is true. The statement assigns the mileage bonus for this instance of the rule to the `bonusPercentage` variable you just created. The action statement is:

```
bonusPercentage = parmMileageBonus
```

Figure 20. Setting the bonus mileage
This completes the construction of the template. Take a few moments to look at what you have created.

9. Rules sets can contain actions without qualifying conditions. These actions are unconditionally always executed. They have no guard expression that determines whether or not they should run. You are going to create such an action. Press the create action button as shown in Figure 21.

**Figure 21. Creating a new action**

10. You want to give the action a presentation string that will be used to describe its function, and also the code statement that should be executed. The statement initializes the `bonusPercentage` variable to zero. Since this is the first action in the rule set, it initializes the value in case no other rule is executed to set a specific value. The code statement is:

```
bonusPercentage = 0
```

**Figure 22. Setting the default bonus percentage**

11. You now want to add a rule and use the template that was previously created as the pattern for that rule. Press the create a rule from a template button as shown in Figure 23.

**Figure 23. Creating a new rule from a template**
12. In the newly created rule, you will notice that there is a place for parameters for that rule. Enter gold for the passenger's status and 25 for the discount amount. This means that for customers who have gold status, their bonus mileage will be 25%.

**Figure 24. Setting values for the rule parameters**

13. Repeat the previous steps to create two more rules from templates. For each rule, specify the values. For “silver” status, the bonus will be 50% and for “platinum” status, the bonus will be 100%.

**Figure 25. Outcome of having created a set of rules**

At the conclusion of this step, you will have defined the base set of rules that describes the current business decisions.

14. After executing the preceding rules, the bonusPercentage variable now contains the bonus value to be added to the base mileage. What remains is adding a final rule to calculate the final bonus value. Press the "add an action" button as shown in Figure 26.

**Figure 26. Adding the final calculation rule**

15. Change both the Presentation and Action values. The Action statement is:
calculatedMileage = baseMileage + (baseMileage * bonusPercentage / 100)

**Figure 27. Calculating the final mileage value**

The variable called `calculatedMileage` is the name of the output parameter that was defined in the interface definition. It will be the value returned to the original caller. This completes the creation of the rule set.

16. You can now save and close the Rule Set editor. Pressing the close button on the Rule Set editor prompts you to save the entries made and close the editor.

17. A rule group is a collection of rule sets that may be executed at different times. You have already created a rule set and also an empty rule group. You now need to define the rule set to be executed. In the Business Integration view, find the `MileageBonusRuleGroup` in the Rule Groups folder. Right click it and select **Open**. This opens the Rule Group editor. **Figure 28. Open the Rule Set**

18. A rule group is exposed to the caller as an interface. For each operation on that interface, you can have different rule sets defined. You want to select the `calculateMileageBonus` operation as the operation to be associated with a Rule Set. Click on the `calculateMileageBonus` operation contained within the `CalculateMileageBonus` interface. **Figure 29. Selecting the operation to associate with the rule set**
19. Execute the rule set at all times and not just at certain times. To achieve this, define your rule set as the default destination for rules. To the right of the Default Destination label is a pull-down from which you can select an existing rule set. Select the *MileageBonusRuleSet*.  
**Figure 30. Setting the default destination for the rule set**

This completes your configuration of the rule group.

20. You can now close the Rule Group editor. Clicking on the close button of the Rule Group editor prompts you to save and close your work. At this point, you have completed the configuration of both the rule group and the associated rule set.

---

**Section 5. Preparing for testing**

You have nearly completed your construction of the rules. What remains is preparing for, and then finally, testing your work. To do this, add your newly created business rules component to the module.
1. Open the Assembly Diagram editor. From within the Business Integration view, right click the **MileageBonus** assembly diagram and select **Open** from the menu.

*Figure 31. Open the Assembly Diagram*

2. Drag and drop the **MileageBonusRuleGroup** onto the Assembly Diagram. Note that it creates an SCA component icon on the diagram.

*Figure 32. Adding the Rules Group to the Assembly Diagram*

3. You don't need to add anything further in the Assembly Diagram to perform the tests. Save the diagram changes using the save (diskette) icon from the toolbar. Do not close the Assembly Diagram editor because you will use it again later.

4. To test the rules, you need to start the server. Switch to the Servers view, select WebSphere Process Server and start it. This takes a few minutes. To start the server (if it is not already running), select and press the Start icon (the green arrow) as shown in Figure 33.

*Figure 33. Starting an instance of WebSphere Process Server*

5. Once the server has started, add the project to the running server. Right click the server entry in the Servers view. In the popup menu, select the
Add and remove projects entry.

Figure 34. Adding projects to the server

6. In the Add and Remove Projects dialog, select the MileageBonusApp from the list of available projects and add it to the server by pressing the Add > button. To complete the task, press Finish. The Module, which contains your business rule, is added to the running WebSphere Process Server.

Figure 35. Selecting the projects to add to the server
Section 6. Running the tests

You are now ready to execute the tests to see the results of defining and adding your business rules.

1. Use the WebSphere Integration Developer test environment to execute tests against the rules. Open the Assembly Diagram (if not already open) and right-click the MileageBonusRuleGroup component. From the popup menu, select Test Component.

Figure 36. Start the WebSphere Integration Developer Test Client

2. In the test environment, enter values for both the flyer status (flyerStatus parameter) and base mileage (baseMileage parameter)
and press the **Continue** button to execute the test.

**Figure 37. Entering test values and running the test**

![Image of Assembly Diagram: MileageBonus with details](image)

3. If prompted for a server to run the test, select your running WebSphere Process Server test server instance and complete it by pressing the **Finish** button.

**Figure 38. Selecting the server upon which to run the tests**
4. The test client will execute the test and display the results. Examine the results and ensure that they look as expected. 
**Figure 39. Reviewing the results**

5. Re-test the solution with different values for both flyer status (gold, silver and platinum) and mileage values. You can enter new test data by selecting a new invocation from the buttons as shown in Figure 40.
**Figure 40. Running additional tests**
Section 7. Changing the rule values

The values of the rules are as they were entered from within the WebSphere Integration Developer editor. The WebSphere Process Server runtime environment is now managing a copy of these rules. You can imagine a production solution with these rules in place and, over time, you may wish to change these rules as the business changes. To make changes, WebSphere Process Server provides a Web-based administration tool called the Business Rules Manager. In this next part of the exercise, you will use this tool to make changes and re-test.

**Note:** By default, this tool is not installed in the server. To successfully execute these tests, you must follow the procedure documented in the online WebSphere Process Server Information Center.

1. The Business Rules Manager provides a Web-based console against which the values of the rules may be modified. From the Servers view, you can launch the console. In the Servers view, right-click the server definition. From the popup menu, select **Launch => Business Rules Manager**.

   **Figure 41. Launching the Business Rules Manager**

2. From within the Business Rules Manager console, expand the rule set and click on **Edit** against the MileageBonusRuleSet.

   **Figure 42. Editing a rule set**
3. Change the values of the rules (for example, by changing the bonus values) and then press the **Save** button.  
**Figure 43. Changing Rule Set values**

4. Although changes have been made through the console, they will not go into effect until after they have been published. Select the **Publish and Revert** link, and then the **Publish** button to commit the changes.  
**Figure 44. Publishing the rule set changes**

5. Repeat the previous steps, re-run some tests to validate that the changes have taken effect.

Congratulations, you have now completed the tutorial and have a working set of business rules.
Section 8. Conclusion

In this tutorial, you:

- Worked with the Rule Set and Rule Group editors.
- Created a rule set template.
- Created rules from the template.
- Added actions that were not conditional on expressions.
- Associated a rule set with a rule group.
- Tested the execution of a rule group.
- Used the Business Rules Manager to make changes.
- Tested the changes.

You have now designed, built, and tested a set of business rules. This is a powerful technique to externalize values for flexible changes, without requiring the operations staff to learn low level mechanical techniques to affect those changes.
Resources

Learn

• WebSphere Process Server Information Center
• WebSphere Business Process Management Information Center V6: Installing the business rules manager
• Resource page: WebSphere Integration Developer and WebSphere Process Server
• IBM Education Assistant for WebSphere Process Server and WebSphere Integration Developer Business Rules
• IBM WebSphere Developer Technical Journal: A guided tour of WebSphere Integration Developer, Part 6
• SOA programming model for implementing Web services, Part 9: Integrating rules with SOA

Discuss

• WebSphere Process Server forum
• WebSphere Integration Developer forum

About the author

Neil Kolban

Neil Kolban is a member of IBM's TechWorks organization. His goal is to make WebSphere Integration Developer and WebSphere Process Server as easy to understand as they are to use.