BEA WebLogic Portal™

Using Rules in Portal Applications

Version 8.1 with Service Pack 3
Document Revised: July 2004

Contents

Using Rules in Portal Applications

Using Rules in Portal Applications

WebLogic Portal gives you a powerful set of tools with which to personalize the user experience in your portal applications. You have dynamic, pinpoint control over the content each user sees, any automatic e-mail messages each receives, and, in a commerce application, what kind of discounts each user receives. To achieve these personalization results, you create user segments, content selectors, and campaigns in the WebLogic Workshop designers provided by WebLogic Portal. Developing personalization with these tools is easy and involves very little Java coding (only using JSP tags). Once this type of personalization is developed, portal administrators can use the WebLogic Administration Portal to modify the behavior of the personalization with no coding at all.

There may be times, however, when you want even more power and flexibility in the personalization you develop. For example, you may want to use personalization to control each user's path through a Page Flow or use runtime information as dynamic input to conditional logic in your code. This type of advanced personalization development is the focus of this guide. (Procedures for creating personalization with user segments, content selectors, and campaigns can be found in the WebLogic Workshop help system.)

To build this type of advanced personalization, you must have a working knowledge of XML and schemas (an advanced version of DTDs), and you must have an intermediate understanding of Java development.

This guide includes the following sections:

- Overview
- Using the Rules Service

Overview

This guide shows you how to use the WebLogic Portal Rules Service to develop advanced personalization.

You can access the Rules Service using two components: the rules controls and the RulesManager EJB. The rules controls (the Rules Executor control and the Rules Manager control), which you can use in Page Flows or Web services, provide a convenient way to add rules functionality to your code without having to write code. For example, using a drag-and-drop interface, you can add a rules control to a Page Flow, select the methods in the control you want to use, and configure the control in the WebLogic Workshop Property Editor.

The rules controls, which delegate calls to the underlying RulesManager EJB, are the preferred way to interact with the Rules Service. However, if you want to use the Rules Service somewhere besides a Page Flow or Web service, you can use the RulesManager EJB directly in your code to access the Rules Service.

This overview section includes the following topics:

- Personalization Components
- Rules Overview

Personalization Components

The following table describes the personalization tools provided by WebLogic Portal. While user segments, campaigns, content selectors, and personalization JSP tags are not the focus of this guide, they are described to highlight the increased programmatic power you have in directly accessing the Rules Service. The Input Objects and Action columns show the increased flexibility and power you have with the rules controls and the RulesManager EJB.

Table 1 WebLogic Portal personalization components

Component	Description	Input Objects	Action (if the input objects match the rules criteria)
User Segments	Dynamically assign users to a grouping, or segment, when the users meet specific conditions. Segment rules are created in WebLogic Workshop with the User Segment Designer. Rules are modifiable in the WebLogic Administration Portal.	Segment rules can be defined with user profile properties, HTTP session/request properties, and date/time values and ranges.	One action: If all conditions evaluate to true, the user is considered a member of the segment. Segments can be used in campaigns, content selectors, and in the <pz:div> JSP tag.
Campaigns	Trigger personalized actions to occur for users who meet specific conditions or perform specific actions. Campaign rules are created in WebLogic Workshop with the Campaign Designer. Rules are modifiable in the WebLogic Administration Portal.	Campaign rules can be defined with user segments, user profile properties, HTTP session/request properties, event characteristics, date/time values and ranges, shopping cart/catalog conditions, and random sampling.	Up to three types of actions: Show a single personalized content item, automatically send a predefined e-mail, provide a discount.
Content Selectors	Show specific content items to users who meet specific conditions. Content selector rules are created in WebLogic Workshop with the Content Selector Designer. Rules are modifiable in the WebLogic Administration Portal.	Content selector rules can be defined with user segments, user profile properties, HTTP session/request properties, and date/time values and ranges.	One action: Show one or more personalized content items.

Table 1 WebLogic Portal personalization components (Continued)

Component	Description	Input Objects	Action (if the input objects match the rules criteria)
Personalization (Interaction Management) JSP Tags	Some of these tags are used to render the results of segment, campaign, and content selector rules.	Displayed content is based user segment, campaign, or content selector rules.	One action: Show personalized content items.
Rules Controls	The Rules Executor control lets you evaluate any input objects (such as a user's profile properties) against a predefined set of rules (rule set). If the rules evaluate to "true" based on the input objects, any predefined action(s) can be triggered (such as assigning the user to a certain classification). The Rules Manager control lets you look up information about rule sets. The rules controls serve as an interface to the RulesManager EJB. You add and configure rules controls in your Page Flows or Web services with a graphical user interface in WebLogic Workshop. You create rules by hand in XML.	**Unlimited types of input objects: You can use the rules you create in XML to evaluate any object put into working memory (discussed later).**	**Unlimited types of actions: Can filter objects in working memory (discussed later) and perform any action defined in the rule set XML.**
RulesManager EJB	Provides the same capabilities as the rules controls in areas of your application other than Page Flows and Web services. Using the RulesManager EJB to access the Rules Service involves Java coding.		

Rules Overview

This guide focuses on using the WebLogic Portal rules controls and the RulesManager EJB to add personalization to your applications. The development of personalization with user segments, campaigns, content selectors, and personalization JSP tags, which involves an intuitive user interface that is more abstracted from the Rules Service architecture, is covered in the WebLogic Workshop help system.

The Rules Service works by reading objects you have put into working memory and evaluating those objects against a set of rules you have predefined in an XML file. (You can think of "working memory" as the place where objects are temporarily stored as the Rules Service is processing the rules.) If the objects in memory match the conditions defined in the rule set (which can be made up of multiple rules), the corresponding rule set actions are triggered. For example, if you put a user's credit score into working memory (from the user profile, from the return of a Web service calculation, or any other way), a rule in the rule set can be defined to say (in XML), "If the user has a credit score equal to or greater than 10, classify that user as a 'gold customer'." The results of that rule processing can be used in your applications any way you choose. For example, if you are developing a Page Flow, you can send a "gold customer" to the `gold.jsp` and send all other customers to another JSP.

- **Rules Controls** – WebLogic Portal provides two rules controls that you can use to invoke the Rules Service from a Page Flow or Web service. The Rules Executor control lets you evaluate objects in working memory against a rule or set of rules, filter the results, and perform actions if the rules evaluate to "true"; and the Rules Manager control provides methods for getting rule set information.

  Note: The Rules Manager control is most useful as a rules development debugging tool.

- **RulesManager EJB** – The RulesManager EJB is the interface into the Rules Service. The rules controls delegate calls to the RulesManager EJB. Use the RulesManager EJB if you want to use the Rules Service in code outside of a Page Flow or Web service.

The Rules Service Process

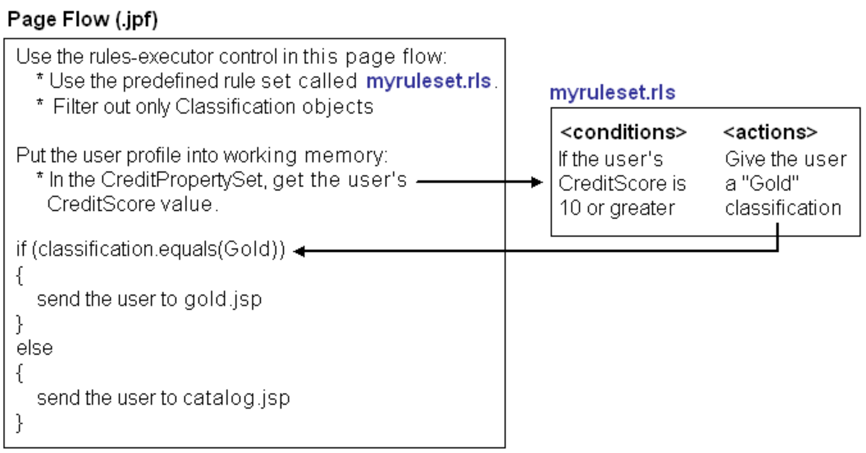
The Rules Service is based on the Rete algorithm, which is optimized for forward chaining reasoning. Following is the order for the rule evaluation process. In this process, the Rules Executor control is used as an example.

1. The Portal Rules Service is initialized, creating its "working memory."

2. The Rules Executor control will identify which rule set to use, which rules to evaluate (defaults to all), and optionally, whether to filter the results. These are all parameters that can be configured on the control.
3. The developer creates and adds objects to "working memory." Example objects might be the user's profile, the Request, and so on. These parameters are passed in as an argument to the rule control's `evaluate*()` method.
4. The Rules Service is invoked by the Rules Executor control and uses the following algorithm.
   a. Match: Evaluates the left hand side (LHS) of the rules to determine which are satisfied given the current contents of working memory.
   b. Conflict resolution: Selects one rule with a satisfied LHS. If no rules have satisfied the LHSs, halts the interpreter.
   c. Act: Performs the actions in the right hand side (RHS) of the selected rule.
   d. Go to step a.
5. The Rules Service fires repeatedly, executing rules according to the state of the input objects and rule conditions. Only one rule can be fired at a time. As conditions are met and rules are fired, more objects may be added to working memory for evaluation.
6. After the Rules Service has reached a state where no more rules will fire, it halts. In addition to the original input objects, new objects created as a result of the rule evaluation may also be in working memory.
7. Because the input objects are part of the results, you may choose to filter the results based on a class. For example, you can specify that only results of class `com.bea.p13n.usermgmt.profile.ProfileWrapper` are returned.
8. The objects are returned to the caller, who then decides what to do with the returned data. For example, the user may be directed to a new page, or the user's profile may have its properties updated.

Figure 1 provides a basic illustration of the previous process with the Rules Executor control used in a Page Flow. The returned results from the Rules Service process are used to determine the user's path through the Page Flow. In the figure, natural language is used instead of code for illustration purposes. (To see the actual parameterization and invocation of the control in a Page Flow, see Using the Control to Determine the User's Path in the Page Flow on page 20.)

Figure 1 Using Rules to control a Page Flow


Page Flow (.jpf)
Use the rules-executor control in this page flow:
* Use the predefined rule set called myruleset.rls.
* Filter out only Classification objects
Put the user profile into working memory:
* In the CreditPropertySet, get the user's CreditScore value.
if (classification.equals(Gold))
{
send the user to gold.jsp
}
else
{
send the user to catalog.jsp
}
myruleset.rls
<conditions>
If the user's CreditScore is 10 or greater
<actions>
Give the user a "Gold" classification

Why is the Rules Service better than if/then?

The Rules Service is more dynamic than a simple conditional in your code. Once a Web application or some other application component has been hard-coded with condition statements (if/then), there is no way to change that without recompiling the code and re-deploying the application. In comparison, rules can be changed and loaded as the Portal server is running. This means the administrator may get the business logic from domain experts, formulate a rule to reflect that logic, and load the rule into the application without ever having to stop the server.

Using the Rules Service

This section shows you how to develop personalization using the rules controls and RulesManager EJB. The following steps are involved:

1. Create a Rule Set
2. Deploy a Rule Set
3. Add Objects to Working Memory
4. Invoke the Rules Service to Evaluate Objects
5. Filter the Results
6. Use the Results in Your Applications

1. Create a Rule Set

Rule sets are sets of instructions written in XML that the Rules Service uses to evaluate objects in working memory. A rule set says, in essence, “if something in working memory meets these conditions, then do this.”

WebLogic Portal does not currently have a rules editor, so you must create rule sets by hand. Rule sets must conform to particular schema. (The rule set schemas are located in `p13n_ejb.jar` in your application root directory.) The language of the rules is actually a usage of the WebLogic Portal expressions package, extended to meet additional requirements for the Rules Service. (See the `com.bea.p13n.expression.operator.*` packages in WebLogic Portal Javadoc for descriptions of the expressions you can use.)

Rule set XML files, which must end in `.rls`, contain the following required base elements:

- <rule-set> - Includes all references to schema used in the rule set
- <rule> - Contains the definition of the rule, which consists of one or more conditions and one or more actions. A rule set can have more than one <rule>.
- <conditions> and <actions> - Each <rule> contains its own if/then clauses that consist of one or more <conditions> (if) and one or more <actions> (then). The Rules Service evaluates the objects in working memory against the conditions. If an object meets a condition, the related actions are executed.

Following is a simple rule set example that says, “If the string ‘Make an Integer 10’ is in working memory, add the object ‘10’ to working memory.”

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Your Name (Your
Company) -->

<rule-set xmlns="http://www.bea.com/servers/p13n/xsd/rules/core/2.1.1"
xmlns:exp="http://www.bea.com/servers/p13n/xsd/expression/expressions/2.1.
1"
xmlns:literal="http://www.bea.com/servers/p13n/xsd/expression/literal/1.0.
1"
xmlns:string="http://www.bea.com/servers/p13n/xsd/expression/string/1.0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/p13n/xsd/rules/core/2.1.1

rules-core-2_1_1.xsd" is-complete="true">

    <rule is-complete="true">
```

```
        <name>Add Integer</name>
        <description>Test Rule</description>
        <conditions>
            <exp:equal-to>
                <exp:variable>
                    <exp:type-alias>java.lang.String</exp:type-alias>
                </exp:variable>
                <literal:string>Make an Integer 10</literal:string>
            </exp:equal-to>
        </conditions>
        <actions>
            <add-object>
                <exp:type-alias>java.lang.Integer</exp:type-alias>
                <exp:arguments>
                    <literal:string>10</literal:string>
                </exp:arguments>
            </add-object>
        </actions>
    </rule>
</rule-set>
```

Unless you are adept at reading schemas and manually constructing valid XML according to the schema's rules, use an XML editor such as XMLSpy (which can be installed from the WebLogic Platform product CD). An XML editor reads a schema, as well as all the schemas the schema imports, and shows you elements and attributes that can be added to an XML document at any given location, showing you available elements and attributes and helping you create a valid XML rule set.

Note: Perhaps the easiest way to create a rule set is to start with an existing one and modify it. There are several example rule sets on the dev2dev site at http://dev2dev.bea.com/products/wlportal81/articles/portalrulesservice.jsp.

Use the following guidelines to create a rule set (or modify an existing rule set) with an XML editor.

Before you begin creating the rule set in XML, write out the rule in natural language to understand all its pieces (conditions and actions) and types of data. What is being put into working memory? What conditions do you want the objects to meet, and what should happen (actions) when objects in working memory meet the conditions?

1. Extract the following schemas from the `p13n_ejb.jar` file in your portal application root **into the same directory in which you will create your rule set**:

   lib/schema/expression*.xsd
   lib/schema/rules*.xsd

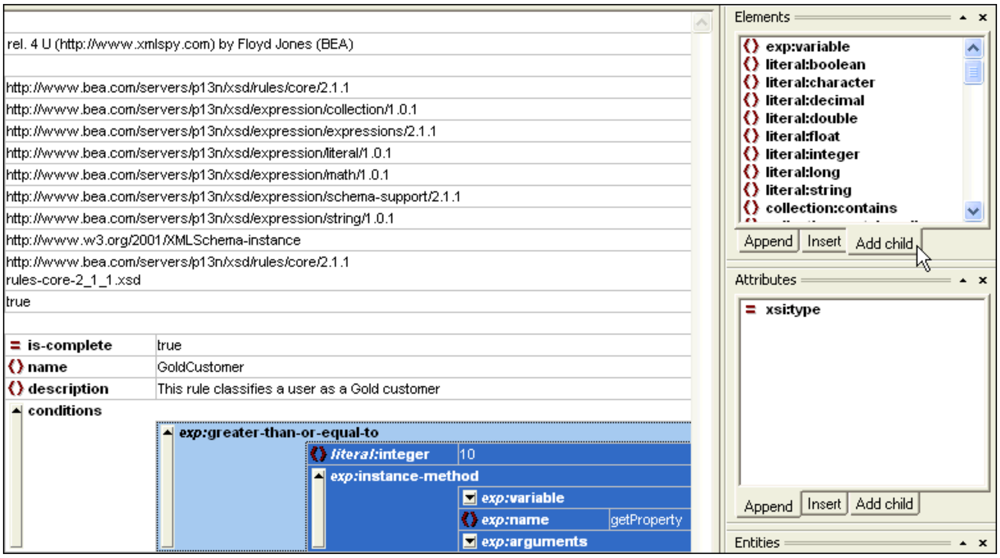
   Start a new document (or open an existing document) in your XML editor and associate the document with the rules-core-2_1_1.xsd schema. This schema also includes imports of other schemas, especially expression schemas, helpful in building rule sets.

   In a new XML document, your XML editor should automatically insert the XML header, automatically import any schemas listed for import, and insert required base elements, such as <rule-set>, <rule>, <name>, <conditions>, and <actions>.

2. Select the <conditions> and <rules> elements and begin building (or modifying) the conditions and rules.

   The following figure shows a rule set being built in XMLSpy. With the <exp:greater-than-or-equal-to> element selected, the Elements pane shows which elements can be added as children. The Attributes pane shows the attributes that can be set on the element.

Figure 2 Building a rule with an XML editor



```
rel. 4 U (http://www.xmlspy.com) by Floyd Jones (BEA)

http://www.bea.com/servers/p13n/xsd/rules/core/2.1.1
http://www.bea.com/servers/p13n/xsd/expression/collection/1.0.1
http://www.bea.com/servers/p13n/xsd/expression/expressions/2.1.1
http://www.bea.com/servers/p13n/xsd/expression/literal/1.0.1
http://www.bea.com/servers/p13n/xsd/expression/math/1.0.1
http://www.bea.com/servers/p13n/xsd/expression/schema-support/2.1.1
http://www.bea.com/servers/p13n/xsd/expression/string/1.0.1
http://www.w3.org/2001/XMLSchema-instance
http://www.bea.com/servers/p13n/xsd/rules/core/2.1.1
rules-core-2_1_1.xsd
true

is-complete    true
name           GoldCustomer
description    This rule classifies a user as a Gold customer
conditions
    exp:greater-than-or-equal-to
        literal:integer    10
        exp:instance-method
            exp:variable
            exp:name       getProperty
            exp:arguments
```

3. After you finish building the rule set, use the XML editor's features to first check for well-formedness and then for validation against the schema. (In XMLSpy, press F7 and F8 respectively.)

   See Working with Invalid Rule Sets for information on fixing invalid rule sets.

4. Save the rule set. (XMLSpy prompts you if you are trying to save an invalid rule set, but it lets you save it.)

5. Copy the rule set to your portal application's `META-INF/data` directory, or to on of its subdirectories. For example, create a `META-INF/data/rulesets` directory and save the rule set there.

Example: Using a Method in a Rule

The following illustration shows how rule set XML uses a method to retrieve a user profile property value so it can be evaluated by the Rules Service:

Table 2 How an XML rule uses a method

Condition

```
<exp:greater-than-or-equal-to>
    <literal:integer>10</literal:integer>
    <exp:instance-method>
        <exp:variable>
            <exp:type-alias>User</exp:type-alias>
        </exp:variable>
        <exp:name>getProperty</exp:name>
        <exp:arguments>
            <literal:string>CreditPropertySet</literal:string>
            <literal:string>CreditScore</literal:string>
        </exp:arguments>
    </exp:instance-method>
</exp:greater-than-or-equal-to>
```

Method the condition maps to

The following take an Object of type `User`:

```
ProfileWrapper pw = SessionHelper.getProfile(request);
Object value = pw.getProperty("CreditPropertySet", "CreditScore");
```

Details about the mapping between the XML and the method:

- The <exp:type-alias> identifies the type of object the method will work on.

  For a list of object type mappings defined in the `parser-mapping-type.properties` file in `p13n_ejb.jar`, see Type Mappings on page 15.

- The <exp:instance-method> indicates a method, and the <exp:name> provides the name of the method.

Note: In order to invoke methods from a rule, the appropriate classes must be imported in the calling code.

- The <exp:argument> includes two <literal:string> elements that provide the String arguments to the method.
- The <literal:integer> identifies a value that the Rules Service will use. The evaluation of the object in working memory (in this case the user profile CreditScore value) determines whether or not the rule's action will be fired.
- The result is that the rule's condition (in XML) retrieves the value of the user's CreditScore. If the value is greater than or equal to 10, in this example, the associated actions are fired.

Working with Invalid Rule Sets

If a rule set won't validate, you'll be shown the area that is invalid. Now you can do the following:

- Make sure you have imported all schemas referenced in the `.rls` file. Those schemas are listed at the top of the *.rls file. The schemas must be in the same directory as the `.rls` file.
- Make sure the XML sections defined in your `.rls` are valid according to the schema. You can open the schema in XMLSpy to check your `.rls` against the schema definitions. In XMLSpy, you can view the schema in design view for a graphical representation of the schema. You can view the `.rls` in text view and enhanced grid view.

  Note: There is no guarantee that a rule set validated in an XML editor will be validated in the Rules Service.

2. Deploy a Rule Set

After you create a rule set and store it in your application's `META-INF/data` directory (or in a subdirectory you create, such as `META-INF/data/rulesets`), the rule set is automatically deployed if the server is running. If the server is not running, the rule set is automatically deployed at server startup. (`META-INF/data` is called the DataSync directory, which also contains the user segments, content selectors, campaigns, and other application metadata you have created.) Rule sets must be deployed to the DataSync directory, and rule set filenames must have an `.rls` extension to be picked up by the Rules Service.

When you modify a rule set in the DataSync directory, the rule set is automatically refreshed on the running server.

3. Add Objects to Working Memory

Rule sets are useless unless they have objects in working memory to evaluate.For example, a rule set may contain a rule that has the following condition: "If the user's credit score is greater than 10." This implies there is either the credit score input, or there is a way to get at the credit score. We could add a credit score to working memory in one of two ways:

Adding a Credit Score to Working Memory from an Integer

You could simply provide a credit score value in your code, either directly (as shown in the following example) or indirectly, for example, through a form input value.

```
Integer value = new Integer(10);
```

`Object [] inputObjects = { value };` (This is a required argument to the `evaluate*()` methods.)

You could then create a rule condition that evaluates the "value" Integer.

Adding a Credit Score to Working Memory from a User Profile

Retrieving a credit score from a user profile is much more flexible and dynamic.

First, you would use code to retrieve the user profile and put it into working memory:

```
ProfileWrapper pw = SessionHelper.getProfile(request);
```

`Object [] inputObjects = { pw };` (This is a required argument to the `evaluate*()` methods.)

Then, in your rule set, you would create a condition that uses a method (getProperty) that retrieves a specific property (CreditScore) from a specific property set (CreditPropertySet). Note that this is the same example used previously in Table 2.

```
<exp:greater-than-or-equal-to>
    <literal:integer>10</literal:integer>
    <exp:instance-method>
        <exp:variable>
            <exp:type-alias>User</exp:type-alias>
        </exp:variable>
        <exp:name>getProperty</exp:name>
        <exp:arguments>
            <literal:string>CreditPropertySet</literal:string>
            <literal:string>CreditScore</literal:string>
        </exp:arguments>
    </exp:instance-method>
</exp:greater-than-or-equal-to>
```

This condition checks to see if the retrieved CreditScore is greater than or equal to the <literal:integer> value of 10.

Note: The `User` type is actually an alias for an object of class ProfileWrapper. This mapping of `User` to “ProfileWrapper,” along with the mappings of other well-known types, are defined in the `parser-mapping-type.properties` file in `p13n_ejb.jar`, shown in the following section.

Type Mappings

The following object type mappings are from the `parser-mapping-type.properties` file in `p13n_ejb.jar`:

Mappings for <type-alias> Tags

`User=``com.bea.p13n.usermgmt.profile.ProfileWrapper`

`Classifier=``com.bea.p13n.user.Classification`

`Capability=``com.bea.p13n.entitlements.common.Capability`

`Role=``com.bea.p13n.entitlements.common.Role`

`Context=``com.bea.p13n.rules.internal.engine.Context`

`Email=``com.bea.campaign.rules.MailActionDef`

`Placeholders=``com.bea.campaign.rules.AddAdToPlaceholderActionDef`

`Discount=``com.bea.commerce.ebusiness.campaign.AddUserDiscountActionDef`

`EndScenario=``com.bea.campaign.rules.EndScenarioActionDef`

`CatalogQuery=``com.beasys.commerce.ebusiness.catalog.rules.CatalogQueryWrapper`

`ContentQueryAdvice=``com.bea.p13n.content.advislets.ContentQueryAdvice`

```
ShoppingCartFacade=com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart
RulesFacade
```

The previous example, and the example in Table 2 on page 12 shows the `<type-alias>` element with a `User` type.

Mappings for <variable> Tags

```
user=com.bea.p13n.usermgmt.profile.ProfileWrapper

request=com.bea.p13n.http.Request

session=com.bea.p13n.http.Session

event=com.bea.p13n.events.Event

randomNumber=java.lang.Number

classification=com.bea.p13n.user.Classification

date=com.bea.p13n.xml.schema.Date

time=com.bea.p13n.xml.schema.Time

timeInstant=com.bea.p13n.xml.schema.TimeInstant

role=com.bea.p13n.entitlements.common.Role

resource=java.lang.String

shoppingCart=com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart
```

4. Invoke the Rules Service to Evaluate Objects

After you have created a rule set and you have objects in working memory, you can invoke the Rules Service to evaluate the objects in working memory with the rule(s) you created.

This section provides an example that shows you how to invoke the Rules Service with the Rules Executor control in a Page Flow.

Prerequisites for the Example

In the example throughout this section, it is assumed a rule set already exists in the `/data/rulesets` directory that classifies users as "GoldCardMembers" or "SilverCardMembers" by reading a user's profile, in a similar way to the example used in Adding a Credit Score to Working Memory from a User Profile on page 14. The Page Flow example in this section shows the user profile being added to working memory. Since the user's profile is needed in this example, it is assumed the Profile control has already been added to an existing Page Flow to enable the getting and setting of user profile properties.

This sample also uses the User Login control for authentication so that the Page Flow knows which user profile to retrieve.

For details on creating a Page Flow, see the Guide to Building Page Flows in the WebLogic Workshop online help system. For details on adding a Portal control to a Page Flow, see Adding Portal Controls to Page Flows in the WebLogic Workshop online help system.

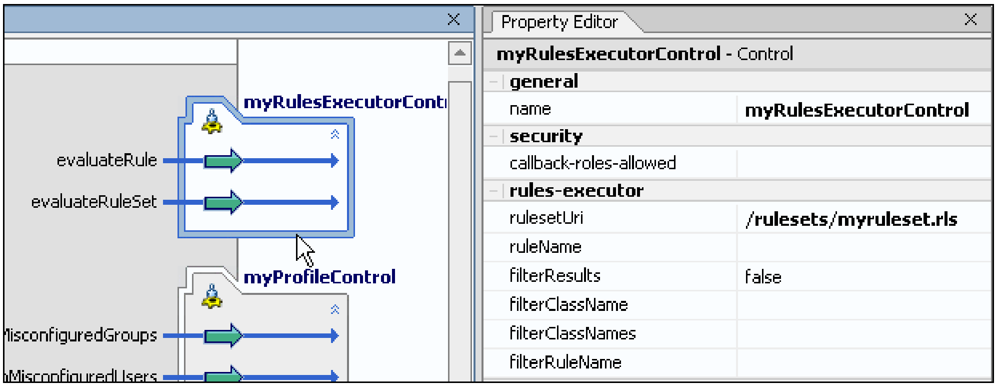
Insert the Control in the Page Flow

When you insert a control in a Page Flow (a `.jpf` file in WebLogic Workshop), all the actions (methods) that are part of that control are available for use. The Rules Executor control has two actions to choose from:

- `evaluateRule` – Lets you evaluate the objects in working memory against a single rule in a rule set.
- `evaluateRuleSet` – Lets you evaluate the objects in working memory against all rules in a rule set.

When you are looking at a Page Flow in Action View, you can select a control you have inserted (by selecting the control's border) and set properties on that control in the Property Editor, as shown in Figure 3.

Figure 3 Setting properties on a control



myRulesExecutorControl - Control

general	
name	myRulesExecutorControl
security	
callback-roles-allowed	
rules-executor	
rulesetUri	/rulesets/myruleset.rls
ruleName	
filterResults	false
filterClassName	
filterClassNames	
filterRuleName	

The Property Editor is a convenient way to send arguments to the RulesManager EJB (which interfaces directly with the Rules Service) without writing Java code.

For example, Table 3 shows how the Rules Executor control properties shown in Figure 3 map to method and constructor arguments in the RulesManager EJB.

Table 3 How control properties map to method and constructor arguments

Rules Executor control properties	**RulesManager EJB methods (see com.bea.p13n.rules.manager)**
rulesetUri ruleName filterResults filterClassName filterClassNames	evaluateRule(String **ruleSetUri**, String **ruleName**, Object[] inputObjects)
	evaluateRule(String **ruleSetUri**, String **ruleName**, Object[] inputObjects, ObjectFilter **filter**)
	evaluateRuleSet(String **ruleSetUri**, Object[] inputObjects)
	evaluateRuleSet(String **ruleSetUri**, Object[] inputObjects, ObjectFilter **filter**)

filterRuleName	**Filter constructors (see com.bea.p13n.rules.manager.RuleResultClassFilter)**
	RuleResultClassFilter(String **ruleName**, Class targetClass)
	RuleResultClassFilter(String **ruleName**, Class targetClassArray)

Notice the RulesManager EJB has the same methods contained in the Rules Executor control: `evaluateRule()` and `evaluateRuleSet()`. The difference is that the Rules Executor control methods take only one argument—for example, `evaluateRuleSet(Object[] inputObjects)`—and provide the rule and filter arguments through the control properties.

On the Rules Executor control, if you set the filterResults property on to "true," the EJB method with the `filter` argument is used, and the filtering properties you enter are automatically sent to that argument.

The filterClassName and filterClassNames properties are simply different options for populating the `filter` argument (with one or more types of filters). Set either filterClassName or filterClassNames on the control, not both.

Use the filterRuleName property to filter on the results of a specific rule in a rule set that has fired. If you use this property, the `RuleResultClassFilter` constructor is called. Notice that the constructor is overloaded to use either a single class filter (that you entered in the filterClassName) property or multiple class filters (that you entered in the filterClassNames) property. The result of using the filterRuleName property is that you not only filter the results of a specific rule that has fired, you can also filter on specific data types.

Following are more detailed definitions of the control properties.

- *rulesetUri* (required) URI of the rule set to use. This is relative to the application's `META-INF/data` directory. For example, if you created a rule set called `myruleset.rls` and stored it in a `data/rulesets` directory, the URI would be `/rulesets/myruleset.rls`. Use the Rules Manger control to list rule sets and rules.
- *ruleName* (optional) Name of the rule to use. The rule must be contained in the rule set specified in the *rulesetUri* property. If not specified, all the rules in the rule set will be evaluated. Defaults to null.
- *filterResults* (optional) Whether to filter the results after the rules have been evaluated. If false, all objects remaining in working memory will be returned. Defaults to false. For information on filtering, see 5. Filter the Results.
- *filterClassName* (optional) Enter class name (for example, `java.lang.String`) of the type of results to return. If left empty, results will not be filtered. Specify either this or the *filterClassNames*, but not both.
- *filterClassNames* (optional) Enter a comma-separated list of class names of the types of results to return (for example, `java.lang.String,java.lang.Integer`). If left empty, results will not be filtered. Specify either this or the *filterClassName*, but not both.
- *filterRuleName* (optional) Filter results of a specific rule that was fired. If this is left empty, results from all rules will be returned. Otherwise, results from only this rule will be returned. This filter is applied in conjunction with the Class filters, if those are specified.

Benefits of the Control

The previous discussion of how properties map to methods and constructors shows the benefits of using the Rules Executor control. By simply filling in property values:

- You do not have to write Java code.
- If you are filtering the results, the filter is constructed automatically for you.

For more information on the Rules Executor control, see the Rules Executor Control Javadoc at com.bea.p13n.controls.rules.RulesExecutorControl.

Using the Control to Determine the User's Path in the Page Flow

After you have added the Rules Executor control to the Page Flow and set the properties on the control, selected one of the control's `execute*` actions to use, and all other prerequisite details are in place (see Prerequisites for the Example), you can add code to the Page Flow that sends a user to a different page depending on the classification they receive from the rule evaluation process.

The following sample Page Flow code shows how a user is directed to a particular page based on the results from the Rules Service running. A similar sample is included in "The Portal Rules Service" on dev2dev at http://dev2dev.bea.com/products/wlportal81/articles/portalrulesservice.jsp (in the `examples/pageFlows/classifyAndFlow` subdirectory).

```
public class Controller extends PageFlowController
{
    /**
    * @common:control
    */
    private com.bea.p13n.controls.login.UserLoginControl userLoginControl;
    /**
    * @common:control
    */
    private com.bea.p13n.controls.profile.ProfileControl myProfileControl;
// The Rules Executor control is added. Properties are configured
// in the Property Editor.
// This is all done in the Page Flow's Action View.
    /**
    * @common:control
   * @jc:rules-executor filterClassName="com.bea.p13n.user.Classification"
filterResults="true" rulesetUri="/rulesets/myruleset.rls"
    */
    private com.bea.p13n.controls.rules.RulesExecutorControl
    myRulesExecutorControl;
    /**
    * @jpf:action
    * @jpf:forward name="default" path="default.jsp"
    * @jpf:forward name="goldCard" path="goldCard.jsp"
    * @jpf:forward name="silverCard" path="silverCard.jsp"
```

```
     * @jpf:catch
type="com.bea.p13n.controls.exceptions.P13nControlException"
path="error.jsp"
     * @jpf:forward name="error" path="error.jsp"
     */
    protected Forward evaulateRuleSetAction(EvaluateRuleSetActionForm form)
     throws P13nControlException
     {
// Start with an empty list into which we add objects to populate
// the working memory of the Rules Service
        List wmObjects = new ArrayList();
        ProfileWrapper pw =
myProfileControl.getProfileFromRequest(this.getRequest());
        if ( pw == null)
        {
           throw new P13nControlException("Undable to retrieve profile from
            request. " + "Make sure PortalServletFilter is configured
            in web.xml for an anonymous user, " + "or that a user
            has logged in.");
        }
// This one will be the condition that fires the rule
        Integer value = new Integer(6);
        myProfileControl.setProperty(pw, "FooPropertySet", "CreditScore",
value);

        wmObjects.add(pw);
// Evaulate all rules in the rule set. Parameters have been declared on the
// control in the Page Flow Property Editor (in Action View).
        Iterator iter =
myRulesExecutorControl.evaluateRuleSet(wmObjects.toArray());
        List results = new ArrayList();
// Let's say we're looking for GoldCardMembers
        Classification goldCardMembers = new
Classification("GoldCardMembers");
        Classification silverCardMembers = new
Classification("SilverCardMembers");
// And we'll direct them to a certain page depending
// on how the rule evaluates
```

```
        Classification classification = (Classification)iter.next();
// Now you would do something with that,
// like show them a different page
        if (classification.equals(goldCardMembers))
        {
// Direct them to high-price stuff
            return new Forward("goldCard");
        }
        else if (classification.equals(silverCardMembers))
        {
// Direct them to lower-price stuff
            return new Forward("silverCard");
        }
// Otherwise, it defaults. Something went wrong.
// Check the rule conditions or turn off filtering on the control
// to see what's in working memory
        }
    }
    return new Forward("default");
}
```

If you want to use only a specific type of object in working memory after the Rules Service has stopped running, you can filter the objects in working memory. Filtering is set using Java types. On the Rules Executor control, you can set the filter type in the Property Editor. See 5. Filter the Results.

For more information on the Rules Executor control, see the WebLogic Portal Javadoc at com.bea.p13n.controls.rules.RulesExecutorControl. For more information on the RulesManager EJB, see the com.bea.p13n.rules.manager package.

Rules Examples

For more examples of using rules controls and the RulesManager EJB, see “The Portal Rules Service” on dev2dev at http://dev2dev.bea.com/products/wlportal81/articles/portalrulesservice.jsp.

5. Filter the Results

When you execute the Rules Service to evaluate objects in working memory, as described in 4. Invoke the Rules Service to Evaluate Objects, you can filter the objects in working memory when the Rules Service has stopped running to return only the objects of a specific type.

Objects exist in working memory as a result of either:

- The caller puts them there (in the inputObjects array).
- A new object is instantiated in one of the rules' actions.

When the Rules Service has halted, several objects may remain in working memory, including those the user initially added. For example, your rule may instantiate a new Classification object into working memory if the rule evaluates to true. Or a rule action might have updated the user's profile, so you'll need to retrieve the profile from working memory.

The API of the Rules Service, when executing a rule, will return an Iterator over the entire contents of working memory *unless you filter the results*. If you are looking only for Classification objects, then you may specify a filter that will return only Classification objects. This filter may be designed based on a single class name, multiple class names, or a given rule, as described in Insert the Control in the Page Flow.

Filtering with the Rules Executor control is easy, since the control implementation takes care of constructing the filter automatically. You need only specify whether to filter, and the filter class name(s) as control properties. The filter will be applied for you automatically by the control if specified.

Filtering with the RulesManager EJB

Filtering with the RulesManager EJB is a bit more cumbersome, as you must design the filter yourself:

```
String filterRuleName = null;

Class filterClass = com.bea.p13n.user.Classification.class;

ObjectFilter filter = new RuleResultClassFilter(filterRuleName,
filterClass);

Class [] filterClasses = { java.lang.String.class,
com.bea.p13n.usermgmt.profile.ProfileWrapper.class};

ObjectFilter filter = new RuleResultClassFilter(filterRuleName,
filterClasses);
```

The filter can then be used as part of the RulesManager EJB, as shown in the following example:

```
public Iterator evaluateRule(String ruleSetUri, String ruleName, Object[]
inputObjects, ObjectFilter filter)
```

If you've filtered the results as shown above, the Iterator should only contain results of the class type(s) you specified. So you can do something like this:

```
while (iter.hasNext())
{
Classification c = (Classification)iter.next();
     if (c.equals(silverCardMembers))
     {
          // do something
     }
}
```

6. Use the Results in Your Applications

Presumably, you had a good reason to invoke the Rules Service: to let it make decisions for you at runtime. Note the benefit of doing things this way rather than hard-coding logic (if/then) into your components.

Here are some examples of using rules and rule results:

- If the time is between 8-5, then direct them to pages relating to brokerage services, else direct them to pages related to investment research.
- If the user lives in Boulder and is female, then show them an advertisement for the Boulder Rock Club.
- If the user’s credit score is > 10, then update their profile (set a property) to classify them as a Gold Member.
- If the date is between December 1 and December 31, send them to the New Year’s promotional JSP.

Rules Service Samples

For samples of the rules controls and RulesManager EJB, besides the example provided in 4. Invoke the Rules Service to Evaluate Objects, see “The Portal Rules Service” on dev2dev at http://dev2dev.bea.com/products/wlportal81/articles/portalrulesservice.jsp.