

Novell exteNd Director

5.0

www.novell.com

RULES GUIDE



Novell®

Legal Notices

Copyright © 2003 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher. This manual, and any portion thereof, may not be copied without the express written permission of Novell, Inc.

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Copyright © 2000, 2001, 2002, 2003 SilverStream Software, LLC. All rights reserved.

Title to the Software and its documentation, and patents, copyrights and all other property rights applicable thereto, shall at all times remain solely and exclusively with SilverStream and its licensors, and you shall not take any action inconsistent with such title. The Software is protected by copyright laws and international treaty provisions. You shall not remove any copyright notices or other proprietary notices from the Software or its documentation, and you must reproduce such notices on all copies or extracts of the Software or its documentation. You do not acquire any rights of ownership in the Software.

Patent pending.

Novell, Inc.
1800 South Novell Place
Provo, UT 85606

www.novell.com

exteNd Director *Rules Guide*
December 2003

Online Documentation: To access the online documentation for this and other Novell products, and to get updates, see www.novell.com/documentation.

Novell Trademarks

ConsoleOne is a registered trademark of Novell, Inc.

eDirectory is a trademark of Novell, Inc.

GroupWise is a registered trademark of Novell, Inc.

exteNd is a trademark of Novell, Inc.

exteNd Composer is a trademark of Novell, Inc.

exteNd Director is a trademark of Novell, Inc.

iChain is a registered trademark of Novell, Inc.

jBroker is a trademark of Novell, Inc.

NetWare is a registered trademark of Novell, Inc.

Novell is a registered trademark of Novell, Inc.

Novell eGuide is a trademark of Novell, Inc.

SilverStream Trademarks

SilverStream is a registered trademark of SilverStream Software, LLC.

Third-Party Trademarks

Acrobat, Adaptive Server, Adobe, AIX, Autonomy, BEA, Cloudscape, DRE, Dreamweaver, EJB, HP-UX, IBM, Informix, iPlanet, JASS, Java, JavaBeans, JavaMail, JavaServer Pages, JDBC, JNDI, JSP, J2EE, Linux, Macromedia, Microsoft, MySQL, Navigator, Netscape, Netscape Certificate Server, Netscape Directory Server, Oracle, PowerPoint, RSA, RSS, SPARC, SQL, SQL Server, Sun, Sybase, Symantec, UNIX, VeriSign, Windows, Windows NT

All third-party trademarks are the property of their respective owners.

Third-Party Software Legal Notices

The Apache Software License, Version 1.1

Copyright (c) 2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.
5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Autonomy

Copyright ©1996-2000 Autonomy, Inc.

Castor

Copyright 2000-2002 (C) Intalio Inc. All Rights Reserved.

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name "ExoLab" must not be used to endorse or promote products derived from this Software without prior written permission of Intalio Inc. For written permission, please contact info@exolab.org.
4. Products derived from this Software may not be called "Castor" nor may "Castor" appear in their names without prior written permission of Intalio Inc. Exolab, Castor and Intalio are trademarks of Intalio Inc.
5. Due credit should be given to the ExoLab Project (<http://www.exolab.org/>).

THIS SOFTWARE IS PROVIDED BY INTALIO AND CONTRIBUTORS ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL INTALIO OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Indiana University Extreme! Lab Software License

Version 1.1.1

Copyright (c) 2002 Extreme! Lab, Indiana University. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Indiana University Extreme! Lab (<http://www.extreme.indiana.edu/>)."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Indiana University" and "Indiana University Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact <http://www.extreme.indiana.edu/>.
5. Products derived from this software may not use "Indiana University" name nor may "Indiana University" appear in their name, without prior written permission of the Indiana University.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

JDOM.JAR

Copyright (C) 2000-2002 Brett McLaughlin & Jason Hunter. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.
3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jdom.org.
4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management (pm@jdom.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the JDOM Project (<http://www.jdom.org/>)."

Alternatively, the acknowledgment may be graphical using the logos available at <http://www.jdom.org/images/logos>.

THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Phaos

This Software is derived in part from the SSLava™ Toolkit, which is Copyright ©1996-1998 by Phaos Technology Corporation. All Rights Reserved. Customer is prohibited from accessing the functionality of the Phaos software.

Sun

Sun Microsystems, Inc.

Sun, Sun Microsystems, the Sun Logo Sun, the Sun logo, Sun Microsystems, JavaBeans, Enterprise JavaBeans, JavaServer Pages, Java Naming and Directory Interface, JDK, JDBC, Java, HotJava, HotJava Views, Visual Java, Solaris, NEO, Joe, Netra, NFS, ONC, ONC+, OpenWindows, PC-NFS, SNM, SunNet Manager, Solaris sunburst design, Solstice, SunCore, SolarNet, SunWeb, Sun Workstation, The Network Is The Computer, ToolTalk, Ultra, Ultracomputing, Ultraserver, Where The Network Is Going, SunWorkShop, XView, Java WorkShop, the Java Coffee Cup logo, Visual Java, and NetBeans are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

W3C

W3C® SOFTWARE NOTICE AND LICENSE

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.

3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

Contents

About This Guide	13
PART I CONCEPTS	15
1 About Rules in exteNd Director	17
Rules and the Rule subsystem	17
The Rule subsystem	17
What a rule is	17
exteNd Director features for rules	19
The Rule API	20
Why use rules?	20
Planning a rules-based application	21
When to use rules	21
Design guidelines	22
2 How You Use Rules	23
How rules work	23
Basic process	23
Rule application components	24
Working with conditions and actions	25
Using whiteboard values	26
Accessing scoped paths	28
Accessing and firing rules	28
Methods for firing rules	29
Firing rules from the context object	29
Firing rules from a rule manager	30
Firing rules from a JSP page	30
Firing temporary rules	31
Handling the result of a rule	32
Context methods for accessing HTTP response values	32
Context methods for accessing the whiteboard	33
Examples of handling return values	33
Using pipelines	35
Benefits of pipelines	36
How pipelines work	36
Validating a pipeline	37
3 Developing Custom Conditions and Actions	39
About custom conditions and actions	39
Designing a condition or action	40
Defining logic	41
Defining logic for a condition	41
Defining logic for an action	43

Defining a condition or action rule descriptor	44
Defining properties	45
Defining JavaBeans	45
Defining runtime properties.	46
Using generic property panels	49
Creating a custom property panel.	50
Writing a BeanInfo class.	51
Using resource bundles	53
PART II TOOLS.	55
4 Rule and Macro Editors	57
Accessing the Rule Editor	57
About the rule tree view	59
Naming a rule	59
Using conditions	59
Editing and deleting conditions.	62
Deactivating a condition	62
Using actions.	62
Editing and deleting actions	64
Deactivating an action	64
Using cases.	65
Adding case descriptions	66
Using other case commands	66
Testing, editing, and saving rules	67
Testing rules.	67
Saving and editing rules	67
Working with condition and action macros	68
Using condition macros	68
Using action macros	70
5 Condition and Action Wizards	73
Using the Condition Wizard and Action Wizard.	73
Using Java templates to define custom conditions and actions	75
Condition template	75
Action template.	76
About the template methods.	77
Using condition and action properties	79
Deploying custom conditions and actions	80
Compiling the condition or action source code	80
Deploying support files	80
6 Pipeline and Binding Editors	81
Basic steps of setting up a pipeline	81
Creating and editing a pipeline	82
Creating a pipeline	82
Editing a pipeline	84

Binding rules to a user, group, or pipeline	85
Creating a rule binding	85
Editing a rule binding	86
PART III REFERENCE	87
7 Installed Actions	89
Accessing condition and action sources	89
Properties that support string templates	90
IvalueOf template	90
Scoped path support	90
Properties that support database drivers and URLs	90
Alphabetical list of actions	92
Add	93
Add Eraser	94
Calculate Age	94
Clear Request Data From Whiteboard	95
Create Collection Of Objects From SQL	95
Default	96
Delete Cookie	96
Deny Access	97
Display Component	97
Display Cookies	97
Display Request Headers	97
Display Whiteboard	98
Divide	98
Drop Cookie User ID	98
Fire Rule	99
Flush	99
Format Date	99
Get Cookie Value	100
Get User Property	100
Log User Off	100
Multiply	100
Query	101
Remove From Whiteboard	101
Return As Decimal Format	101
Return As Html Body	102
Return As Html Bold	103
Return As Html Break	103
Return As Html Checkbox	104
Return As Html File Upload	104
Return As Html Hidden Field	105
Return As Html JavaScript	106
Return As Html Option List	106
Return As Html Password	107
Return As Html Radio Button	108

Return As Html Reset Button	109
Return As Html Scripted Button	109
Return As Html Submit Button	110
Return As Html Table	111
Return As Html Text Area	112
Return As Html Text Field	112
Return As XML	113
Return Authentication Required	115
Return False	115
Return Response	115
Return Response With Default	115
Return True	116
Save Cookies To Whiteboard	116
Save Form Get Data To Whiteboard	116
Save Request Data To Whiteboard	116
Save To Whiteboard	117
Send Mailer SMTP	117
Set Component Parameter	118
Set Cookie Value	118
Set Date On Whiteboard	119
Set Expired	119
Set Next Activity	120
Set Pipeline Status	120
Set Response Header	120
Set Response Status	121
Set User Property	121
Set Workitem Priority	121
Set Workitem Value	122
SQL Hierarchy	122
SQL String	122
Stop Rule Processing	123
Subtract	123
8 Installed Conditions	125
Alphabetical list of conditions	125
Check Component Parameter	126
Check Date	126
Check Date Within Range	127
Check Day	127
Check For Cookie	128
Check Month	128
Check Request Data	128
Check Time	128
Check User	129
Check User Group	129
Check User Property	130
Check Whiteboard	130

Check Whiteboard Value	131
Check Whiteboard Value Is Empty	131
Check Workitem Value	131
Default	132
Is Form Get Data Available.	132
Is New Session.....	132
Save Cookies To Whiteboard	132
Save Form Get Data To Whiteboard	132
Save Request Data To Whiteboard	133
Set Action Off	133
Set Action On	133
Set Action On Or Off	133
PSQL Check For Column	133
SQL String	134
9 Rule JSP Tag Library	135
doAction	136
doCondition	137
conditionalRule	138
fireRule	139

About This Book

Purpose

This book shows how to use the Rule subsystem to create and use rules in your Novell® exteNd Director™ applications.

Audience

This book is for Java developers.

Prerequisites

This book assumes you are familiar with Java programming, XML, and developing Web applications.

Organization

Here's a summary of the contents of the book:

Part and chapter	Description
PART I: Concepts	
1 About Rules in exteNd Director	Provides an overview of the Rule subsystem and how to design rule-based applications
2 How You Use Rules	Describes how rules work and how to use and deploy rules and pipelines in exteNd Director applications
3 Developing Custom Conditions and Actions	Describes how to write your own classes for special-purpose conditions and actions

Part and chapter	Description
PART II: Tools	
4 Rule and Macro Editors	Describes how to use the exteNd Director Rule Editor and the Condition and Action Macro Editors
5 Condition and Action Wizards	Describes how to write custom conditions and actions using the exteNd Director Condition Wizard and Action Wizards
6 Pipeline and Binding Editors	Describes how to use the exteNd Director Pipeline Editor and Pipeline Binders
PART III: Reference	
7 Installed Actions	Describes how to use the installed actions for creating rules in the exteNd Director Rule Editor
8 Installed Conditions	Describes how to use the installed conditions for creating rules in the exteNd Director Rule Editor
9 Rule JSP Tag Library	Describes how to use the custom JSP tags to call methods in the Rule subsystem API



Concepts

Provides an overview of rules and the fundamentals of how to develop rule-based applications

- [Chapter 1, “About Rules in exteNd Director”](#)
- [Chapter 2, “How You Use Rules”](#)
- [Chapter 3, “Developing Custom Conditions and Actions”](#)

1

About Rules in exteNd Director

This chapter provides an overview of exteNd Director's Rule subsystem and describes some of the benefits of developing rules-based applications. It has these sections:

- ◆ [Rules and the Rule subsystem](#)
- ◆ [Why use rules?](#)
- ◆ [Planning a rules-based application](#)

Rules and the Rule subsystem

Typically, exteNd Director applications fire rules at crucial points in their logic flows to make application decisions. A rule can perform virtually any programming task. You can handle the result in your portlet code, or you can let the rule handle processing outside the scope of your application.

The Rule subsystem

The Rule subsystem provides a Java API and tool support for creating flexible, reusable logic for your exteNd Director applications.

What a rule is

A rule is a combination of *conditions* and *actions* that return a value. The basic formula for a rule is: if a condition or group of conditions are true, the associated action or actions are executed. Rule structure is based on the *case statement*, a standard construct in many programming languages. You create rules using exteNd Director's *Rule Editor*, which provides a user interface for building the rule logic.

Basic rule structure Every rule consists of at least one *decision node* (case) that has at least these two sections:

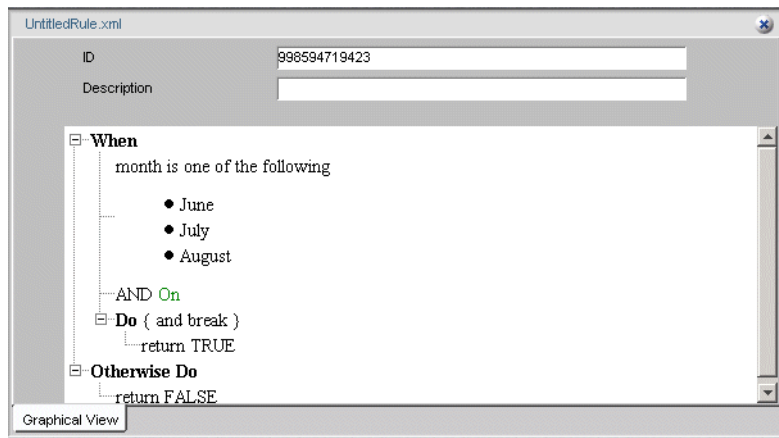
- ◆ The **When** section contains one or more conditions. You select the conditions and specify how the results of individual conditions are combined using AND and OR operators.
- ◆ The **Do** section contains one or more actions that you select. It is executed when the When section evaluates to true. You can also insert decision nodes within an action, which are evaluated as nested rules.

A final **Otherwise Do** section is optional. It specifies default actions for the rule that are executed when none of the decision nodes are true.

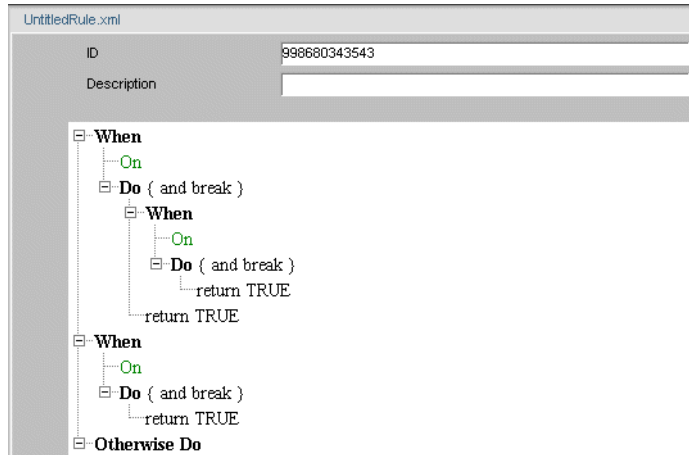
For example, a rule with one case and a default section has this format:

```
When {condition group is true}
  Do {actions}
Otherwise Do {actions}
```

Here's what it looks like in the Rule Editor:




Nested logic Logic can be nested within rules by adding child nodes to a parent node, and as with standard case statements, you can control the flow of processing using break and continue statements:



exteNd Director features for rules

Here are the rule-based components you can create in the exteNd Director development environment:

Feature	Description
Conditions and actions	Conditions and actions are prebuilt, reusable JavaBeans with properties you can set in the Rule Editor. exteNd Director supplies an installed set of general-purpose conditions and actions. You can also create your own custom versions.
Macros	A macro is a series of conditions or actions that can be combined for reuse. For example, a condition macro could be: if today is a weekday, and the time is between 9 a.m. and 5 p.m., and this is not November.
Pipelines	A pipeline is a series of rules that you can set up in the Rule Editor and execute in your applications. Pipelines can be bound to users and groups, or to any named pipeline you define.

 For information about the Rule and Macro Editors, see [Chapter 4, “Rule and Macro Editors”](#).

The Rule API

After you define rules in the Rule Editor, you can access them in your applications using the Rule API. The API includes a *rule manager* and a *context* object for firing rules and accessing session data. There are also implementation classes for defining custom conditions and actions.



For more information, see the `com.sssw.re` packages in the API online help.

Why use rules?

As an application developer, you might ask: why use rules? Why not just code the logic directly in my application? From the application development and deployment standpoint, rules provide several advantages:

Benefit of rules	Details
Easy to create and use	<p>The Rule Editor provides a set of general-purpose conditions and actions that can address many application needs. Using a point-and-click interface, you can combine conditions and actions in flexible ways to create different types of rules. You also have access to runtime properties (such as <code>userID</code>), and you can easily store, retrieve, and evaluate your own session values using the whiteboard feature.</p> <p>exteNd Director API methods make it easy to fire rules and build logic by exchanging information between rules and application code.</p>
Have encapsulated logic	<p>Rules contribute toward the goal of making applications modular. You can enhance code maintenance, for example, by separating business logic in rules and handling presentation in your application code.</p> <p>Also, depending on your application design, this separation of functions allows nonprogrammers such as business analysts to reconfigure logic without adversely affecting the underlying application code.</p>

Benefit of rules	Details
Reusable and extendable	<p>Conditions and actions can be reused to create different rules for different purposes, or the rules themselves can be reused in the same application or across applications. For example, you might use the same set of rules with different properties to be executed when users in different groups log in to an application, or you might have some unique rules for each group, with some overlapping rules.</p> <p>Because conditions and actions are implemented as JavaBeans, they are easily extensible. The Rule subsystem provides complete support for JavaBeans, including built-in constructs that facilitate creating your own custom conditions and actions.</p>
Interoperable with other exteNd Director subsystems	<p>Rules can easily be integrated with other exteNd Director subsystems. For example, you can use rules:</p> <ul style="list-style-type: none"> ◆ To control access to documents in the Content Management subsystem ◆ To define routing logic in a Workflow application ◆ For user profiling in exteNd Director applications

Planning a rules-based application

Because rules are flexible and provide many implementation options, you need to carefully plan how and when to use them in your applications. Your most fundamental decision is whether to implement your business logic in rules rather than directly in your application code.

When to use rules

Here are criteria to consider:

If logic	Then use
<ul style="list-style-type: none"> ◆ May be shared in different parts of an application ◆ Can be used in multiple applications ◆ Data or criteria change frequently ◆ Can be updated by nonprogrammers 	Rules

If logic	Then use
<ul style="list-style-type: none"> ◆ Has a narrow application ◆ Data or criteria change rarely ◆ Must be isolated for a specific scenario ◆ Must be controlled by programmers 	Direct Java code

Design guidelines

Here are some guidelines for designing a rules-based application:

Guideline	Details
Determine what business logic can be encapsulated in rules and what needs to be handled by the application itself	This will vary with requirements. As a general principle, the more logic you can encapsulate in rules, the more reusable your code will be.
Create a detailed design specification that defines what each rule does	<p>The design document should include the following:</p> <ul style="list-style-type: none"> ◆ The condition or set of conditions (condition node) that determine an action. For example, WHEN the current user belongs to the administrator group AND today is a weekday. ◆ The action or actions taken when the condition is true or false. For example: if the condition node is true, return true AND return an HTTP response phrase to the caller. OTHERWISE return false AND return a different response phrase. ◆ How the application that fires the rule should respond to the result of the action. For example: if the action is true, display the response AND increment a log; if the action is FALSE, display the response and send an e-mail. In some cases, the rule may not respond at all.

2

How You Use Rules

This chapter describes how you use rules in your applications. It has these sections:

- ◆ [How rules work](#)
- ◆ [Working with conditions and actions](#)
- ◆ [Accessing and firing rules](#)
- ◆ [Handling the result of a rule](#)
- ◆ [Using pipelines](#)

How rules work

As stated in the first chapter, a rule is a combination of conditions and actions that return a value. You define rules in the exteNd Director Rule Editor, and then access rules and handle the results in your exteNd Director application code.

Basic process

Using rules is a three-step process:

- 1** In the Rule Editor, you create a rule by selecting conditions and actions and setting their properties. For example, you might select the **CheckDay** condition and set the property to **When today is Thursday**, then select the **ReturnResponsePhrase** action and enter an appropriate phrase.
- 2** From the appropriate point in your application code, you fire the rule using one of the available methods. For example:

```
fireRule(Thursdays)
```

- 3 You handle the result of the rule in your application. For example, if the rule returns a certain response phrase when today is Thursday and another phrase otherwise, you set the result in your application.

Rule application components

These are the components for building rules:

Component	Description
EbiContext	EbiContext provides access to the whiteboard and has methods for firing rules, validating pipelines, and handling rule results.
EbiRuleManager	EbiRuleManager provides alternative methods for firing rules and accessing rules associated with rule owners. Owners, defined in the Rule Editor, allow you to organize rules by application or other criteria.
Rule	A rule defines a set of conditions and actions that you configure in the Rule Editor. The rule definition is saved as an XML descriptor, which is used by the Rule subsystem to execute rules and return a value to the caller.
Conditions and actions	Conditions and actions are Java classes (typically JavaBeans) with properties you can set using the Rule Editor. These properties can include session and object values accessed from the whiteboard.
Whiteboard	The whiteboard is an area where session values and specified objects can be stored. The whiteboard is accessed from EbiContext (the context object).

Working with conditions and actions

You use the exteNd Director’s Rule Editor to create rules by selecting installed (prebuilt) conditions and actions.

NOTE: Before creating rules, you should become familiar with the Rule Editor. For more information, see [Chapter 4, “Rule and Macro Editors”](#).

There are several ways you can set up your business logic. Some setup can be done in the rule itself, and other setup needs to be done in application code before the application fires the rule. Here are some techniques for using the installed conditions and actions to set up values and build results:

Technique	How to implement it
Getting, setting, and removing whiteboard values	<ul style="list-style-type: none">◆ Use one of the several Save To Whiteboard conditions or actions. The data on the whiteboard is then available to other conditions and actions via !valueOf templates.◆ Check whether a whiteboard key exists (Check Whiteboard condition) and if not, use Save To Whiteboard to create it.
Saving cookies on the client	<ul style="list-style-type: none">◆ Check for the existence of a cookie with the Check For Cookie condition.◆ Use the Save Cookies To Whiteboard action to make cookies available as whiteboard keys.◆ Use the Set Cookie Value action to create cookies that the application or rule needs.
Building a response phrase	<ul style="list-style-type: none">◆ Use the several Return Response and Return HTML actions to put data in the response phrase.◆ Select the Append check box (when available in an action property sheet) to incrementally build a response from several actions.
Setting response status	<ul style="list-style-type: none">◆ Use the Return True, Return False, or Set Response Status action.

 For details about these and other installed conditions and actions, see [Chapter 8, “Installed Conditions”](#) and [Chapter 7, “Installed Actions”](#).

Using whiteboard values

An exteNd Director session includes a *whiteboard* where you can store values needed for your business logic. You give each whiteboard value a key that you use to retrieve the value. You can access the whiteboard from installed conditions and actions in the Rule Editor as well as from your application code. Whiteboard values are accessed in a condition or action property sheet in the Rule Editor.

Using the !valueOf template

For some properties, you can either enter the actual value or specify a whiteboard key that holds the value. You can also specify a key that holds the name of another key.

In the Rule Editor, if a condition or action property has a caret (^) as part of its name, you can use the special code `!valueOf.whiteboardkey`, where *whiteboardkey* is the name of the key whose value you want. The rule substitutes the key's value for the special code.

To specify a whiteboard key, use this format:

```
!valueOf.keyname
```

where *keyname* is a key that exists on the whiteboard when the application runs the rule.

NOTE: *keyname* is case sensitive.

Using built-in whiteboard values

There are several built-in constants you can use like whiteboard keys in !valueOf expressions. They provide information about the logged-in user, the rule that is being run, and the current date and time:

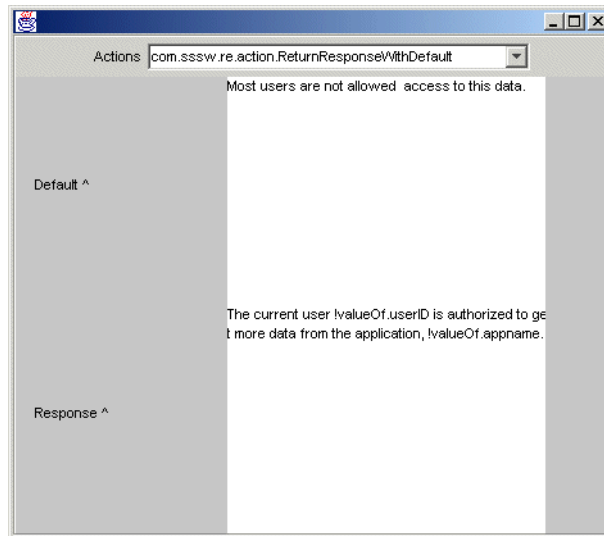
Constant	Description
userID	The ID for the user logged in to the application. If the user is not logged in, the userID is anonymous . If the user's session times out, the userID reverts to anonymous .
today	The current date. The value is returned as a string using the date format appropriate for the user's locale.
now	The current time as a number, returned as a string. The time value is the difference (measured in milliseconds) between the current time and midnight on January 1, 1970 UTC (Coordinated Universal Time).
response	The text stored in the response phrase of the context object. Many rule actions assign a value to the response phrase.

Constant	Description
ruleID	The ID of the rule currently being run.
ruleDesc	The description of the rule currently being run.
uri	A reference to the content currently set in the browser.

Example of !valueOf

In the following panel, the ReturnResponseWithDefault action has a Default property for a message to the user. In the message you can refer to values on the whiteboard. This sample text uses the system whiteboard value **userID** and a key called **appname**, created by the application:

```
The current user !valueOf.userID is authorized to get more data
from the application !valueOf.appname.
```



Accessing object attributes from the whiteboard

If an item on the whiteboard is a programming object rather than plain text, you can access any attributes (instance variables) associated with the object. You can refer to object attributes on the whiteboard with this format:


```
!valueOf.whiteboardkey^attributename
```

For example, if your application puts an EbiUser object on the whiteboard, you can get the user's last name this way:

```
!valueOf.myUser^lastName
```

Accessing whiteboard values

The context object provides several methods for accessing values stored on the whiteboard.

 For more information, see “[Context methods for accessing the whiteboard](#)” on [page 33](#).

Removing values from the whiteboard


You can use an *eraser* to remove a whiteboard key at a designated time or after a specified number of accesses. An eraser has the same name as the whiteboard key it will erase. You can use the installed actions **AddEraser** and **RemoveEraser** to manage erasers. You can also access eraser methods from the `com.sssw.fw.api.EbiWhiteboard` class.

Accessing scoped paths

The `^` template field in conditions and actions also supports substitution syntax for scoped paths. Scoped paths allow you to access different types of data in your exteNd Director applications, such as documents in the resource set and the Content Management subsystem.

You can specify a scoped path using this format:

```
#{spath}
```

 For more information, see the chapter on [working with scoped paths and XPath](#)s in *Developing exteNd Director Applications*.


Accessing and firing rules

When you fire a rule, the Rule subsystem evaluates the conditions for a decision node (the When section in the Rule Editor) and if true, executes the node’s actions (Do section). If no decision node evaluates to true, the Rule subsystem executes the default actions (Otherwise Do section— if any). The result of the action is then returned to the application.

NOTE: You can fire rules from an `EbiContext` object or from an `EbiRuleManager`. The rule manager provides additional methods for firing and accessing rules and is the recommended implementation for applications that use rules extensively.

Methods for firing rules

Here are the context and rule manager methods for firing rules:

Method	In class	When to use
fireRule()	EbiContext EbiRuleManager	When you have defined and saved a rule in the Rule Editor.
fireTemporaryRule()	EbiContext EbiRuleManager	When you have a rule definition as an XML string. This allows you to fire a rule by providing the entire XML rule definition, rather than a rule ID defined in the Rule Editor in exteNd Director.  See “Firing temporary rules” on page 31 .
isTrue()	EbiRuleManager	When a rule defined in exteNd Director sets the response status to a true or false value. The isFalse() method fires the specified rule and reports whether the rule returned a true value.
isFalse()	EbiRuleManager	When a rule defined in exteNd Director sets the response status to a true or false value. The isFalse() method fires the specified rule and reports whether the rule returned a false value.

Firing rules from the context object

Typically, you fire rules from an exteNd Director portlet or a JSP page. First you need to instantiate a rule context object using this method:

```
public static EbiContext createEbiContext(HttpServletRequest request, HttpServletResponse response, ServletContext servletContext)
    throws EboFactoryException
```

You can get the request and response objects from an EbiContext object, which you can instantiate using another factory method.

Here is sample code showing how to fire a rule from a portlet’s doView() method; where **myapp** is the (optional) rule owner and **access rule** is the rule ID:

```
public void doView( RenderRequest req, RenderResponse res ) {
    try{
        // get an EbiContext from the RenderRequest
        com.sssw.fw.api.EbiContext context =
            com.sssw.fw.factory.EboFactory.createEbiContext( req, res,
```

```

        GenericPortlet.getPortletContext() );
// Get the rule context
    reContext = com.sssw.re.factory.EboFactory.createEbiContext(
        context );
// Fire the rule
    reContext.fireRule("myapp.accessRule");
// handle the result...
// Catch exceptions ...

```

Firing rules from a rule manager

A rule manager allows you to fire rules without specifying the owner each time. You can invoke a rule manager for a particular owner and fire rules by specifying the rule ID.

This code instantiates a rule manager in a portlet:

```

public void doView( RenderRequest req, RenderResponse res ) {
    try{
        {
            // If a rule owner exists
            EbiRuleManager rm = com.sssw.re.factory.EboFactory.
                createRuleManager( "myapp" );
            /** For rules with no owner
            EbiRuleManager rm =
                com.sssw.re.factory.EboFactory.createRuleManager();
            */
            // get an EbiContext from the RenderRequest
            com.sssw.fw.api.EbiContext context =
                com.sssw.fw.factory.EboFactory.createEbiContext( req, res,
                    GenericPortlet.getPortletContext() );
            // Get the rule context
            reContext = com.sssw.re.factory.EboFactory.createEbiContext(
                context );
            rm.fireRule("myRule", reContext);
            // handle the result...
            // Catch exceptions ...

```

NOTE: Regardless of the owner you used to instantiate a rule manager, you can fire any rule as you would from the context object by specifying an owner:

```

    rm.fireRule("secondApp.contentRule", reContext);

```

Firing rules from a JSP page

You can also get a rule manager directly from a JSP page or a servlet. In this case you need to get a class that extends HttpServlet and call getServletContext(). Then you can get the rule manager and fire the rule the same way you would from a portlet:

```

public class reTester extends HttpServlet
{

```

```

ServletContext m_servletContext = null;
public void init( ServletConfig config )
    throws ServletException
{
    super.init( config );
    // Initialize any instance variables...
    m_servletContext = config.getServletContext();
}
// Create a rule manager and fire the rule
EbiRuleManager rm = com.sssw.re.factory.EboFactory.
    createRuleManager();
com.sssw.re.api.EbiContext ctx =
    com.sssw.re.factory.EboFactory.createEbiContext(
        m_servletContext.getEbiRequest().getHttpServletRequest(),
        m_servletContext.getEbiResponse().getHttpServletResponse(),
        m_servletContext.getServletContext());
rm.fireRule(myRule, ctx);

```

Firing temporary rules

Typically you define rules in the exteNd Director Rule Editor, where they are saved to a known location in your exteNd Director project resource set. This is what allows you to reference rules by rule name. Temporary rules are XML strings that can be referenced directly. You can get the XML for a rule by defining it in the Rule Editor and exporting the XML using `EbiRuleManager.toXMLString()`.

Here is how you fire a temporary rule from the rule context object:

```

string xmlrule = ...;
reContext.fireTemporaryRule(xmlrule);

```

Here is how you fire a temporary rule from the rule manager:

```

rm.fireTemporaryRule(xmlrule, reContext);

```

Handling the result of a rule

This section describes methods you can use to handle HTTP response values and whiteboard values returned from a rule, and includes some code examples.

Context methods for accessing HTTP response values

This table lists the methods available on the rule EbiContext to handle HTTP response values:

Accessor method	Usage
get/setResponsePhrase()	<p>Accesses an HTTP response phrase. A response phrase can be used to pass data between an action and the application code.</p> <p>Typically, actions set a response phrase by calling <code>setResponsePhrase()</code>; the application code can then retrieve the data by calling <code>getResponsePhrase()</code>. When your action sets a response phrase, it should also set a response type by calling the <code>setResponseType()</code> method.</p>
get/setResponseStatus()	<p>Accesses an HTTP response status code. A response status code tells the calling object the status of the return value. For example, the action <code>Return True</code> returns the status code 200. <code>Return False</code> returns 412. HTTP status codes are defined in <code>EbiResponse</code>.</p> <p>Typically, actions set the status code by calling <code>setResponseStatus()</code>; the application code calls <code>getResponseStatus()</code> to retrieve the code.</p> <p>NOTE: If a rule action returns true or false, use the <code>isTrue()</code> or <code>isFalse()</code> method to fire the rule. For more information, see “Methods for firing rules” on page 29.</p>
get/setResponseType()	<p>Accesses the type of value associated with the response phrase. Response types are defined in <code>EbiResponse</code> as <code>CONTENT</code>, <code>HTML</code>, <code>TEXT</code>, and <code>URL</code>.</p>

Context methods for accessing the whiteboard

This table lists some EbiContext methods to store and retrieve session and object values from the whiteboard.

Method	Description
get/setValue()	Gets the object associated with a specified whiteboard key. Use these methods for values that need to persist during the session.
get/setTemporaryValue()	Gets the object associated with a specified whiteboard key. Use these methods for values that do not need to persist beyond the initiating request.
get/setValueNames()	Gets an array of the whiteboard keys that are defined in the session.
hasValue()	Checks whether the specified key has a value on the whiteboard.
merge()	Processes text that contains !valueOf expressions. The merge() method finds !valueOf expressions in the passed template, gets the keyname associated with it, and gets the value of that key from the whiteboard.
removeValue()	Removes a value associated with the specified key from the whiteboard.
removeValues()	Removes all values and their keys from the whiteboard.

Examples of handling return values

This section provides some code examples for handling values returned by rules in your application code.

Example of a rule that returns a boolean

This code shows a portlet that fires a rule called **bonus** whose owner is **sample**.

The bonus is based on this rule: “If today is a weekday, display the value **the bonus is \$100,000**; otherwise, display **\$1**.”

The rule's actions return true or false by setting the response status to 200 for true and 412 for false. The code determines what to do with those results:

```
public void doView(RenderRequest req, RenderResponse res) {
    try {
        //get the context
        com.sssw.fw.api.EbiContext context =
            com.sssw.fw.factory.EboFactory.createEbiContext(req,
                res, GenericPortlet.getPortletContext());
        // Get the rule manager
        EbiRuleManager rm =
            com.sssw.re.factory.EboFactory.createRuleManager("sample" );
        // Get the rule context
        com.sssw.re.api.EbiContext reContext =
            com.sssw.re.factory.EboFactory.createEbiContext(context);
        // nitialize bonus
        Double bonusAmount = null;

        // Fire rule to determine bonus.
        if (rm.ruleExists( "bonus" ) )
        {
            if (rm.isTrue( "bonus", reContext ))
                bonusAmount = new Double(100000.00);
            else
                bonusAmount = new Double(1.00);
        }
        else
            bonusAmount = new Double(0.00);
    }
    catch (Exception e)
    {
        // ..exception handling ...
    }
}
```

Example of a rule that returns data

Building on the preceding example, suppose the rule returns a bonus amount instead of true or false. Both the Do and Otherwise Do actions are ReturnContentFromData, rather than ReturnTrue and ReturnFalse. Do gets the value **bonus1** and Otherwise Do gets **bonus2** from the whiteboard. The selected action returns the value as a response phrase, which is available from the context object.

Before firing the rule, you must get the values (from a company database, for example) and call `EbiContext.setValue()` to set the values for the whiteboard keys **bonus1** and **bonus2** in this user's session:

```
String bonusString = "0";
Double bonusAmount = null;

// Get a rule manager and context objects.
EbiRuleManager rm = com.sssw.re.factory.EboFactory.
```

```

createRuleManager();
com.sssw.fw.api.EbiContext context =
    com.sssw.fw.factory.EboFactory.createEbiContext(req,
        res, GenericPortlet.getPortletContext());
com.sssw.re.api.EbiContext reContext =
    com.sssw.re.factory.EboFactory.createEbiContext(context);

// Fire rule to determine bonus.
if ( rm.ruleExists( "bonus" ) )
{
    context.setValue("bonus1", ...); // retrieve value from db
    context.setValue("bonus2", ...); // retrieve value from db
    rm.fireRule( "bonus", reContext );
    bonusString = context.getResponsePhrase();
    bonusAmount = Double.valueOf(bonusString);
}

```

Example of a rule that uses a whiteboard value

Again using the bonus example, suppose instead that the bonus value is set in the Rule Editor. The Do and Otherwise Do sections both use the installed action **Return Response**, where the bonus amounts are specified:

```

String bonusString = "0";
Double bonusAmount = null;

// Get a rule manager and context objects as shown
// in previous example.


// Fire rule to determine bonus.
if ( rm.ruleExists( "bonus" ) )
{
    rm.fireRule( "bonus", reContext );
    bonusString = context.getResponsePhrase();
    bonusAmount = Double.valueOf(bonusString);
}

```

Using pipelines

A *pipeline* is a mechanism for binding a rule or set of rules to a known user or group, or for firing a set of rules in a specified order as a unit.

You set up a pipeline using exteNd Director’s Pipeline and Binding Editors.

 For information about setting up pipelines in exteNd Director, see [Chapter 6, “Pipeline and Binding Editors”](#).

Benefits of pipelines

Pipelines enhance the power of rules by providing a higher level of logical encapsulation and reusability, while maintaining the benefits of tool-based maintenance and the hot deployment capability. Here is a summary of some of the benefits:

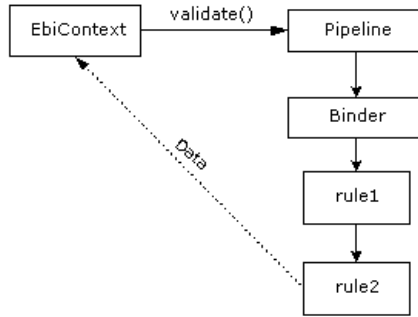
Benefit of pipelines	Details
Easy to organize and maintain complex logic	<p>Pipelines are especially useful for applications with complex logic that depends on one or more top-level conditions. This is especially relevant with user and group bindings.</p> <p>For example, a corporate Web site might be driven primarily by user or organizational group. You might have rules that stop processing under certain conditions, or whose actions fire other rules under other conditions. It would be much easier to maintain this kind of logic in a single pipeline, especially if the logic needs to be updated frequently.</p>
Enhanced separation of logic and presentation	<p>When you validate a pipeline, the application does not need to know what rules will be fired—only what value will be returned.</p> <p>For example, you can avoid firing rules in case statements and let the pipeline handle this higher-level logic. This means you can push more of your business logic out of your portlet or JSP page, which can be used to handle presentation.</p>
Enhanced rule reusability	<p>Pipelines are reusable, just like rules. You might have two or more applications that share logic that can be built into pipelines.</p>

How pipelines work

When you create a pipeline, you can associate it with one or more binders, of which there are three types:

Binder type	When to use
User	To fire rules associated with a specified user. The rules will fire only when this user is in the application session.
Group	To fire rules associated with a specified group. The rules will fire only when a member of this group is in the application session.
Pipeline	To fire rules associated with the specified pipeline.

Here is how a pipeline is processed:



After you create a pipeline, you use the appropriate Binding Editor to select a rule or rules and specify the order in which you want them to fire. Then you use the `validate()` method to execute the pipeline in your application code. You handle the result of the pipeline the same way you handle the result of a rule.

 For more information, see [“Handling the result of a rule” on page 32](#).

Validating a pipeline

Firing the rules in a pipeline is called *validating* the pipeline. You use the `validate()` method on the `EbiContext` object.

First you need to get the `EbiContext`, as shown in [“Firing rules from the context object” on page 29](#).

This code validates the pipeline you specify:

```
ctx.validate("CheckAllAccessRestrictions");
```

This code sets a pipeline ID in the `EbiContext` object and validates it:

```
ctx.setPipelineID("CheckAllAccessRestrictions");  
ctx.validate();}
```


3

Developing Custom Conditions and Actions

This chapter describes how to write your own classes for special-purpose conditions and actions. It includes the following topics:

- ◆ [About custom conditions and actions](#)
- ◆ [Designing a condition or action](#)
- ◆ [Defining logic](#)
- ◆ [Defining properties](#)

This chapter assumes you are familiar with rules-based applications and using the Rule Editor in exteNd Director. For background information, see [Chapter 2, “How You Use Rules”](#).

About custom conditions and actions

Conditions and actions are Java classes used to build rules fired by portlets and other application components. The Rule Editor provides a selection of prebuilt conditions and actions. You can also write custom conditions and actions to meet your application’s specific requirements.

Conditions and actions are typically implemented as JavaBeans, with properties you can set in the Rule Editor.

Designing a condition or action

Designing a condition or action involves several decision points.



What logic should be implemented? When designing rule-based logic, consider how conditions and actions will interact with each other and with the application source code. Here are key considerations:

- ◆ What condition should be tested?
- ◆ What action should be executed if the condition is true?
- ◆ What action should be executed if the condition is false?
- ◆ What portlets will fire the rule that contains these conditions and actions?
- ◆ What data must be passed by the condition to its actions and by actions back to the firing portlet?

Will any of the installed conditions or installed actions meet my requirements? Before you begin your development, you should become familiar with the installed conditions and actions. There may be one that will meet your requirements, or you may want to modify one of the installed versions. For more information, see [Chapter 8, “Installed Conditions”](#) and [Chapter 7, “Installed Actions”](#).

What kind of user interface should be rendered by the Rule Editor? By default, the Rule Editor generates a generic property panel as the interface for interacting with conditions and actions you use to build rules. If a condition or action has properties, the property panel provides controls—such as check boxes and text boxes—for choosing or entering required values. You can also specify a custom user interface in your condition and action class, which the Rule Editor then uses to render the property panel. For more information, see [“Defining properties” on page 45](#).

What supporting classes will be needed? Conditions and actions often rely on supporting classes such as:

Type of class	Used for
Custom classes	Implementing logic not provided in the exteNd Director API
BeanInfo classes	Providing additional information for the condition or action JavaBean  For more information, see “Writing a BeanInfo class” on page 51
Resource bundles	Localizing static strings that appear in the property panel for your conditions and actions  For more information, see “Using resource bundles” on page 53


Does the rule need to access runtime values? You can define properties to access session whiteboard values that can be set dynamically at runtime. For more information, see [“Defining runtime properties” on page 46](#).

Defining logic

Conditions and actions each have three methods that define their implementations:

Method	Used in	Description
doCondition()	Conditions	Fulfills the requirements of a condition In this method, you write code that makes comparisons or evaluates property values and returns a boolean value
doAction()	Actions	Fulfills the requirements of an action In this method, you write code that performs actions based on whether the associated condition evaluates to true or false This method typically returns the result of a business rule
toString()	Both conditions and actions	Returns a string that describes the condition and its current settings This string appears as a description of a condition or action in the Rule Editor

These methods are included in the templates generated by exteNd Director’s Condition Wizard and Action Wizard.

 For an overview of the templates and core methods, see [“About the template methods” on page 77](#).

Defining logic for a condition

Typically a condition compares property values entered in the Rule Editor and returns a boolean. This logic is coded within the doCondition() method.

The doCondition() method includes an EbiContext object, allowing you to access runtime values.

Example of doCondition()

This example shows code from the installed condition CheckTime, which checks whether the current time of day is within the time range specified for the condition in the Rule Editor:

```
// ...
// JavaBean get/set accessor methods
public int getFrom() {
    return from;
}

public void setFrom( int from ) {
    this.from = from;
}

public int getTo() {
    return to;
}

public void setTo( int to ) {
    this.to = to;
}
// Check to see if the current time of day is within the specified
// range.
public boolean doCondition( com.sssw.re.api.EbiContext context )
throws com.sssw.re.exception.EboConditionException {
    int hour = new GregorianCalendar().get( Calendar.HOUR_OF_DAY
);

    if ( from < to ) {
        // Daytime check
        if ( hour >= from ) {
            if ( hour <= to ) {
                return true;
            }
        }
        return false;
    }
    else {
        // Crossover check
        if ( hour >= to ) {
            if ( hour <= from ) {
                return false;
            }
        }
        return true;
    }
}
// ...
```

Notes about the condition example

get/set methods These are JavaBean accessor methods that define two properties for this condition: **from** a specified time and **to** a specified time (inclusive). These values are accessed from the property panel in the Rule Editor.

 For information, see [“Defining JavaBeans” on page 45](#).

doCondition() This method is the workhorse of condition classes. In this example the property values are compared to the current time, which is obtained from a Java `GregorianCalendar` object. The code does two checks—for within current day and for crossover times—in each case returning a boolean.

Defining logic for an action

An action returns the result of a rule, based on the return value of one or more conditions. The logic for an action is coded in the `doAction()` method.

Actions can perform virtually any activity, including accessing a database via SQL, manipulating whiteboard values, returning HTTP and HTML values, performing operations, and controlling user access to objects. Like conditions, actions typically have properties settable in the Rule Editor.

Returning values to the caller

Because actions return the result of a rule, you need to understand how the result is handled by the caller, typically a portlet or JSP page. Here is a summary of the common HTTP response methods you might use to return a value from an action to the portlet that fired the rule:

Return value method	Description	Usage
<code>EbiContext.setResponseStatus()</code>	Sets an HTTP status code that indicates whether the action succeeded or failed	Called from <code>doAction()</code> method
<code>EbiContext.setResponsePhrase()</code>	Sets a response phrase used to pass data from the action back to the firing portlet	Called from <code>doAction()</code> method
<code>EbiContext.setResponseType()</code>	Sets the type of the response phrase to indicate how the firing portlet should process the data	Should be called from the <code>doAction()</code> method whenever the method <code>EbiContext.setResponsePhrase()</code> is called

Example of doAction()

The following shows code from the installed action ReturnAsHTMLBold. It returns the text entered by the user in bold format:

```
public void doAction( com.sssw.re.api.EbiContext context ) throws
    com.sssw.re.exception.EboActionException {
    context.setResponsePhrase( "<b>" + getValue().getValue( context ) +
        "</b>" );
```

Defining a condition or action rule descriptor

The toString() method provided by the template returns a description of the condition or action that appears in the Rule Editor when a condition or action is selected. Typically, toString() calls the properties' get methods to include property values.

Example of toString()

Here is how toString() is implemented in the installed condition CheckTime:

```
// Use resource bundle, "caResource" instantiated with Class
// definition, for string processing.
public String toString() {
    return caResource.getString("the hour is between <b>") +
        getHours().elementAt( from ) + caResource.getString("</b>")
        and <b>") + getHours().elementAt( to );
}
// Implement a JPanel using getParameterPanel() using getHours()
// to get selected property values.

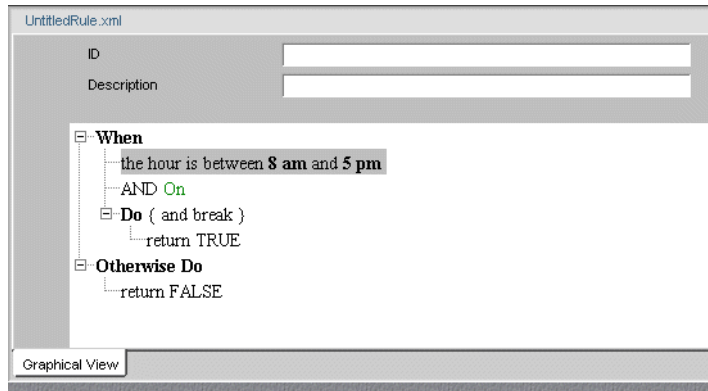
private Vector getHours() {
    if ( hours == null ) {
        hours = new Vector();
        hours.addElement( caResource.getString("12 am"));
        hours.addElement( caResource.getString("1 am"));
        hours.addElement( caResource.getString("2 am"));
        // ...
```

Notes about toString() example

A local method, getHours(), uses a Vector to store property values. The toString() method calls getHours() to display the hours selected in the Rule Editor.

NOTE: In this example, toString() directs processing to a resource bundle that is instantiated with the condition class. This provides support for localization. For more information, see ["Using resource bundles" on page 53](#).

Here is what the result looks like in the Rule Editor:



Defining properties

There are several ways to define properties and property panels for conditions and actions. The Rule subsystem supports Java constructs like JavaBeans and JPanels, and also provides default controls for certain data types and objects and support for using runtime values in properties. You can also follow the implementation of resource bundles and BeanInfo classes used in the installed conditions and actions.

This section provides descriptions and examples of each approach.

Defining JavaBeans

Like other JavaBeans, conditions and actions can have properties. By defining properties, you allow conditions and actions to be customized in the Rule Editor for use in a variety of rules.

When you add properties in a custom condition or action, you must follow JavaBeans standards. For each property, define:

- ◆ A member variable whose name has all lowercase letters
 - ◆ A pair of get and set accessor methods
- By convention, each method name consists of **get** or **set** prepended to the property name starting with an initial capital letter (such as `getTime`).

The Rule Editor can use these constructs to automatically generate a property panel for the condition or action, based on the data type of the property. For a model implementation, see [“Example of doCondition\(\)” on page 42](#).


You can also use a JDK BeanInfo class to enhance the display properties. See [“Writing a BeanInfo class” on page 51](#).

Defining runtime properties

The Rule subsystem provides two string template data types that allow properties in conditions and actions to be set dynamically from runtime values:

- ◆ **EboStringTemplateSingle**: single-line text box, rendered as a TextField
- ◆ **EboStringTemplateMulti**: multiline text box, rendered as a TextArea

When you define properties using these data types, the Rule Editor lets you specify values for these properties by referencing whiteboard keys. When the rule is fired, the property will be set to the real-time value associated with the whiteboard key.

 For information about using the whiteboard, see [Chapter 2, “How You Use Rules”](#).

Working with the !valueOf construct

For each property you want to set dynamically, you need to define a member variable of type EboStringTemplateSingle or EboStringTemplateMulti. The Rule Editor will generate the associated TextField or TextArea, labeled with a string that ends with the caret (^) symbol. This symbol indicates that the property has been defined as a string template that can be set either statically by entering a literal string or dynamically by using the construct **!valueOf**. With this construct, you specify the value associated with a whiteboard key. Here is the syntax:

```
!valueOf.whiteboard key
```

Certain session values, such as `userID`, are automatically provided. For example: to set the property to the value associated with the whiteboard key `UserID`, enter **!valueOf.UserID**.

Resolving the valueOf! expression When you write a condition or action that needs to evaluate a runtime value, use one of these methods to get the value:

Method	In class	Usage
merge()	EbiContext	Gets a string of one or more whiteboard values by providing the whiteboard key names
getValue()	EboStringTemplateSingle EboStringTemplateMulti	Gets a string value for a whiteboard key associated with a specified template

Example of using string template values

This example shows code from the installed action SaveToWhiteBoard. The action saves data to the whiteboard in the specified whiteboard key:

```
public boolean resolve;
    public com.sssw.re.core.EboStringTemplateSingle data = new
com.sssw.re.core.EboStringTemplateSingle();
    public com.sssw.re.core.EboStringTemplateSingle template = new
com.sssw.re.core.EboStringTemplateSingle();

// Set the template value for the whiteboard key that will be used
// to store the data value.
    public void setKeyTemplate( EboStringTemplateSingle detailKey )
{
    this.template = detailKey;
}
// Return the template for the whiteboard key.
    public EboStringTemplateSingle getKeyTemplate() {
    return template;
}

// Return the template for the data.
    public EboStringTemplateSingle getData() {
    return data;
}

// Set the template value for the data to be saved to the
// whiteboard.
    public void setData( EboStringTemplateSingle data ) {
    this.data = data;
}

// Return boolean true if the data value is a templated string
// that needs to be resolved, and false if the data value is the
// value to be saved.
    public boolean getResolve() {
    return resolve;
}
```

```

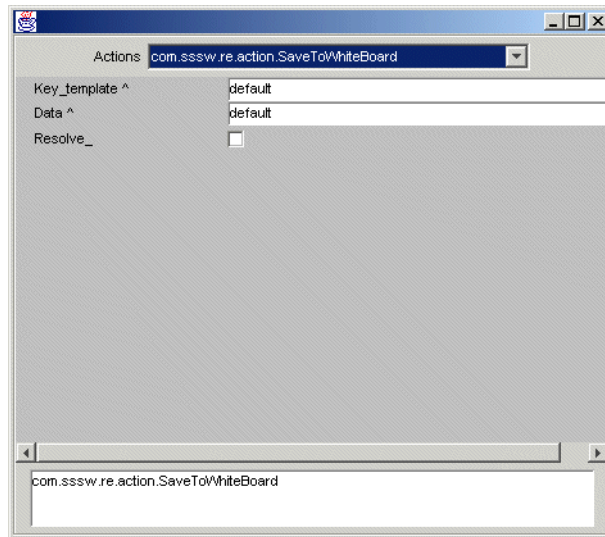
// Set the value that determines if the data value needs to be
// resolved.
public void setResolve( boolean resolve ) {
    this.resolve = resolve;
}

// Save the data value, or resolved data value if resolved is
// true, to the whiteboard in the specified whiteboard key.
public void doAction( com.sssw.re.api.EbiContext context ) throws
com.sssw.re.exception.EboActionException {
    String merged = getKeyTemplate().getValue( context );
    context.setValue( merged, ( getResolve() ?
getData().getValue( context
) : getData().getTemplate() ) );
}

```

Notes on string template example

This action defines three properties that appear in the Rule Editor:



EboStringTemplateSingle The code instantiates two template objects for each template property. The first property (Key-template) will be resolved to get its data representation. The second property (Data) is resolved only if the Resolve control is selected.

doAction() Gets the properties and implements these methods:

Method	Description
getValue()	The template method that resolves the key value for Key_template
setValue()	The context method that sets the Key-template value on the whiteboard—and: <ul style="list-style-type: none">◆ If Resolve is false, sets the Data value string on the whiteboard◆ If Resolve is true, uses template.getValue() to resolve the key value

Using generic property panels

The Rule subsystem supports a set of data types for which it can automatically generate generic property panels. For each supported data type, the Rule Editor renders a preselected GUI control in the property panel. All you need to do is implement JavaBean accessor methods for each property.

Here is a list of the supported data types and the associated controls:

Supported data type	GUI control
java.lang.String	TextField
char	
int	
long	
double	
float	
boolean	Checkbox
com.sssw.re.core.EboStringTemplateSingle	TextField
com.sssw.re.core.EboStringTemplateMulti	TextArea
com.sssw.re.condtion.Compare	DropDownListBox
com.sssw.re.core.EboRule	

Creating a custom property panel

If you prefer to create your own property panel rather than use the generic controls, you can implement the `getParameterPanel()` method to specify a custom user interface. The Rule Editor requires the custom interface to be defined as a `JPanel`; it will not create a custom property panel based on any other type of portlet returned by `getParameterPanel()`.

When including the `getParameterPanel()` method in a condition or action, you must implement a listener interface to allow the controls in the `JPanel` to respond to user actions.

Example of using a custom JPanel

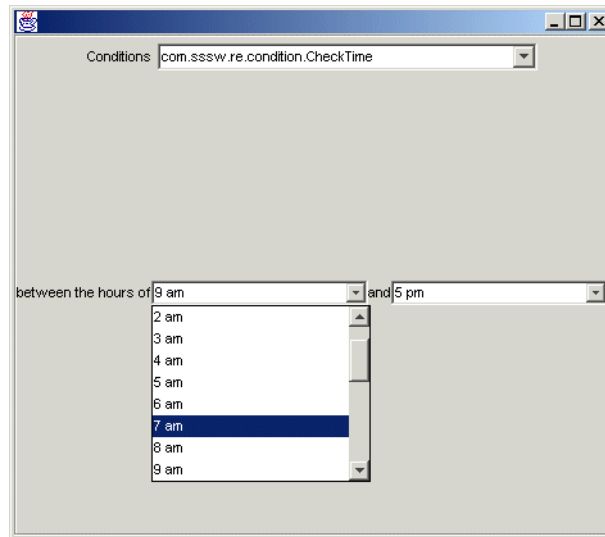
This is the code that defines the property panel for the installed condition `CheckTime`:

```
// ...
// Return a custom ui component to be used for editing this
// condition.
public java.awt.Component getParameterPanel() {
    JPanel p = new JPanel();
    p.setLayout( new BorderLayout( p, BorderLayout.X_AXIS ) );
    p.add( new JLabel( caResource.getString("between the hours
of") ) );
    p.add( FROM = new JComboBox( getHours() ) );
    p.add( new JLabel( caResource.getString("and") ) );
    p.add( TO = new JComboBox( getHours() ) );
    FROM.setAlignmentY( JComponent.CENTER_ALIGNMENT );
    TO.setAlignmentY( JComponent.CENTER_ALIGNMENT );
    FROM.setSelectedIndex( from );
    TO.setSelectedIndex( to );
    FROM.addItemListener( this );
    TO.addItemListener( this );
    return p;
}

// Return a vector of hours for selection in the ui.
private Vector getHours() {
    if ( hours == null ) {
        hours = new Vector();
        hours.addElement( caResource.getString("12 am"));
        hours.addElement( caResource.getString("1 am"));
        hours.addElement( caResource.getString("2 am"));
    }
    // ...
}
```

Notes on custom JPanel example

Here is the custom property panel generated by the code:



JComboBox implementation The variables TO and FROM are defined as ints. Using a generic panel would result in text fields for each property. Instead, `getParameterPanel()` instantiates a JPanel and implements a pair of JComboBox controls, providing a dropdown list of time selections in the Rule Editor. The code defines a local method, `getHours()`, that uses a Vector to fill in the values of the combo boxes.

Resource bundle The code uses a resource bundle `caResource` to localize the labels for the ComboBox. For information, see [“Using resource bundles” on page 53](#).

Writing a BeanInfo class

You can write a BeanInfo class for any condition and action to specify the appearance of attributes in the property panel. For example, you can use a BeanInfo class for:

- ◆ Specifying a display name for each property
- ◆ Implementing localization for property names and static text strings that appear on the panel for each property

NOTE: Using a BeanInfo class is the recommended way to implement resource bundles for localization. For information about resource bundles, see [“Using resource bundles” on page 53](#).

Like JavaBeans, BeanInfo classes require their associated condition or action classes to specify properties as member variables with associated get and set methods.


➤ **To create a BeanInfo class:**

- 1 Create a Java class in your preferred Java editor. The class must extend EboBeanInfo and implement these code elements:

Code element	Description
imports	For conditions: <code>package com.sssw.re.condition.*</code> For actions: <code>package com.sssw.re.action.*</code> For conditions and actions: <code>com.sssw.re.core.*</code> <code>java.beans.*</code>
getClazz() method	Returns the condition or action class associated with the BeanInfo class
PropertyDescriptor[] pds	Provides information to the Rule Editor for generating the property panel for the condition or action The array should be the same size as the number of properties you want to appear in the property panel
PropertyDescriptor pd	Defines a descriptor for (each) property Each individual property descriptor is added to the property descriptor array

- 2 Name the BeanInfo class by appending **BeanInfo** to the name of the associated condition or action class.

For example: if your class name is **CheckCondition**, the BeanInfo class must be named **CheckConditionBeanInfo**.

 For an example of a BeanInfo class, see [“Example of a BeanInfo class” on page 53](#).

Using resource bundles

Resource bundles are a standard feature of the Java JDK used for localization. You can store any object, string, number, or other data in a resource bundle. To create a resource bundle, you need to create either a Java class that extends `ListResourceBundle` or a properties file that contains static strings. For each locale you want to support, you provide a separate version of the resource bundle.

You can use resource bundles to implement localization in conditions, actions, and associated `BeanInfo` classes. Here is a recommended approach:

- 1 Create resource bundles for all locales supported by your application.
- 2 In your condition, action, or `BeanInfo` class, declare `ResourceBundle` objects that reference your resource bundles by calling the `EboResourceBundle.getBundle()` method. This simplest version of this method takes one argument, the fully qualified name of your resource bundle class.

For example: if your resource bundle is **MyBundle.class** located in package **locales**, your declaration should look like this:

```
static com.sssw.re.util.EboResourceBundle eoCABundle =
    com.sssw.re.util.EboResourceBundle.getBundle("locales.MyBundle"
    )
```

- 3 Access the data in the resource bundle by calling `get` methods on the `ResourceBundle` object.

For example: if your resource bundle contains localized strings, call the `getString()` method on the `ResourceBundle` object, passing the key associated with the desired string value.

NOTE: For complete information on creating resource bundles, see the reference documentation for the `ResourceBundle`, `ListResourceBundle`, and `PropertyResourceBundle` classes in the Java standard API documentation. All these classes are in the `java.util` package.

Example of a `BeanInfo` class

This is the code that defines the `BeanInfo` class for the installed condition `CheckTime`:

```
package com.sssw.re.condition;
import com.sssw.re.core.*;
import java.beans.*;

public class CheckTimeBeanInfo extends EboBeanInfo {

    // Resource Bundle definition
    static com.sssw.re.util.EboResourceBundle eoCABundle =
        com.sssw.re.util.EboResourceBundle.getBundle(
EboConstant.CA_BUNDLE,
        getClass().getClassLoader() );

    public Class getClazz() {
```

```

        return com.sssw.re.condition.CheckTime.class;
    }

    public PropertyDescriptor[] getPropertyDescriptors() {
        PropertyDescriptor[] pds = new PropertyDescriptor[ 2 ];
        PropertyDescriptor pd;
        pd = super.getPropertyDescriptor( "From" );
        pd.setDisplayName( eoCABundle.getString( "From" ) );
        pds[ 0 ] = pd;
        pd = super.getPropertyDescriptor( "To" );
        pd.setDisplayName( eoCABundle.getString( "To" ) );
        pds[ 1 ] = pd;
        return pds;
    }
}

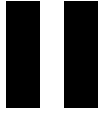
```

Notes on the BeanInfoClass example

Getting the resource bundle Uses `EboResourceBundle.getBundle()` to get the name of the resource bundle for the installed conditions and actions stored in `CA_BUNDLE` and get the class loader.

Defining property descriptors Since the `CheckTime` condition defines two properties, it requires two property descriptors—as indicated in the array declaration.

Specifying a display name for a property You can use the `setDisplayName()` method to specify a property name. When the Rule Editor generates the property panel, it uses this display name as the label for the UI control associated with the property. The `setDisplayName()` method is called on the property descriptor and takes a string argument. This example shows the recommended approach for implementing resource bundles. It uses key values **To** and **From**—passing them to the resource bundle so that the Rule Editor displays a localized string.



Tools

Describes how to use the Director Workbench tools to design rules and pipelines and write custom conditions and actions


- [Chapter 4, “Rule and Macro Editors”](#)
- [Chapter 5, “Condition and Action Wizards”](#)
- [Chapter 6, “Pipeline and Binding Editors”](#)

4

Rule and Macro Editors

This chapter describes how to create and edit rules in exteNd Director. It has the following sections:

- ◆ [Accessing the Rule Editor](#)
- ◆ [Using conditions](#)
- ◆ [Using actions](#)
- ◆ [Using cases](#)
- ◆ [Testing, editing, and saving rules](#)
- ◆ [Working with condition and action macros](#)

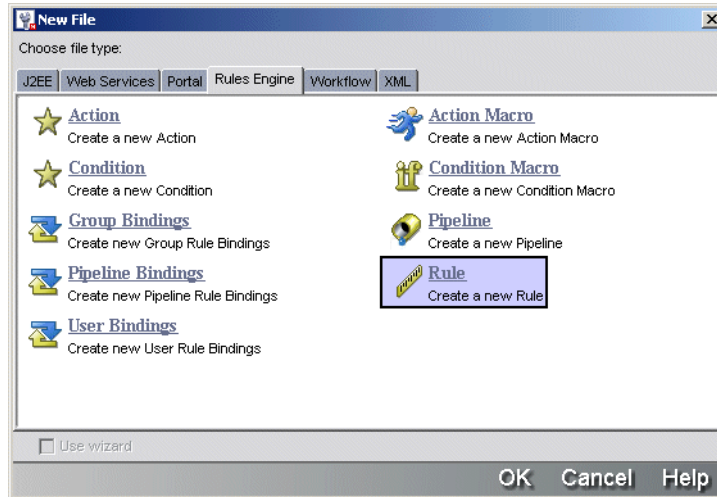
 For background information about using rules in exteNd Director applications, see [Chapter 2, “How You Use Rules”](#).

Accessing the Rule Editor

The Rule Editor is where you create rules to use in a portlet or JSP page in your application. You build a rule by selecting from predefined conditions and actions. Each rule definition is saved as an XML file in a specified directory in your exteNd Director project.

➤ **To access the Rule Editor:**

- 1 From the exteNd Director menu, select **File>New**.
- 2 In the New File dialog, select the **Rules Engine** tab:



- 3 Select the **Rule** icon.

The Rule Editor displays in the upper-right panel:



About the rule tree view

Graphically, a rule is a combination of one or more conditions (When) and one or more actions (Do) that produces some result. It has the form:

```
When the combination of some conditions is true,  
    Do some actions;  
Otherwise Do some other actions.
```

Each When-Do statement is called a *node* and maps to a decision node in XML. You can also embed case statements within a parent node, as described in [“Using cases” on page 65](#).

The Otherwise Do section is executed only when all nodes in the rule evaluate to false.

Naming a rule

You invoke the rule name when firing a rule.

You can specify a rule owner as part of the identifier; doing so allows you to organize your rules by application and avoids possible naming conflicts with other applications. You can use the Rule API to access rules associated with an owner.

➤ To name a rule:

- 1 In exteNd Director, open the Rule Editor.
- 2 Enter the rule name in the ID text box. For example:
`myrule`
- 3 (Optional) To specify the owner, use the format *ownername.rulename*. For example:
`myapp.myrule`
- 4 (Optional) Enter a description in the Description text box.

Using conditions

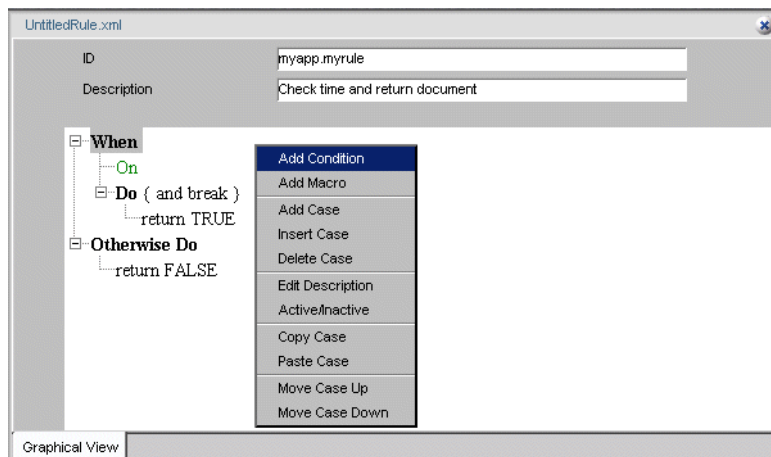
A condition is a Java class containing logic that returns true or false. The installed conditions are implemented as JavaBeans that in many cases allow you to set relevant properties. You can add multiple conditions for a rule and manipulate those conditions in various ways.

Default condition The default value for a condition is **When:On**. In cases where you always want the associated actions to execute (for certain types of pipelines, for example), you can leave the default and just select actions for the Do section.

Reusing a series of conditions If you have a series of conditions that you might want to reuse in another rule, you can create a macro for it. For more information, see [“Working with condition and action macros” on page 68](#).

➤ **To add a condition:**

- 1 In the Rule Editor, select the **When** section where you want to enter the condition and right-click to display the condition popup menu:



- 2 Select **Add Condition**.

The condition property panel displays. The dropdown lists all installed conditions plus any custom conditions you have written and deployed.

- 3 Select the condition you want.

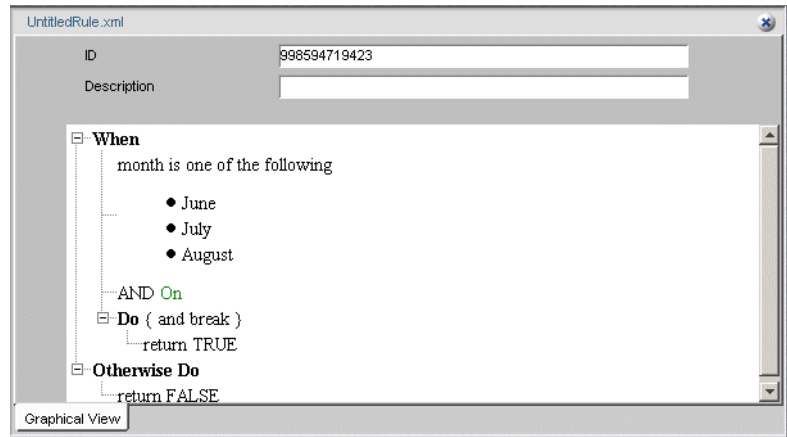
The settable properties for the condition display on the panel.



For information about using the installed conditions, see [Chapter 8, “Installed Conditions”](#).

- 4 Set the properties as required and exit the panel.

The condition description appears in the Rule Editor. For example, choosing the Check Month condition displays a result like this:



➤ **To add more conditions:**

- 1 In the Rule Editor, select the description of the condition below which you want to add a condition. To add the condition at the top, select the appropriate **When** element.
- 2 Right-click and select **Add condition** from the popup menu.
- 3 Add the condition as described in **“To add a condition:”** on page 60.
By default, the condition is added with the AND operator, meaning that this condition and the previous condition must be true for the Do section to execute.

Selecting a logical operator for processing When you have two or more conditions in a series, you can specify the OR operator if you want either condition to be true and specify the NOT operator if you want a condition to be false.

➤ **To toggle the operator between AND and OR:**

- 1 Select the condition containing the operator.
- 2 Right-click and select **AND/OR** from the popup menu.

➤ **To toggle the NOT operator to and from NOT:**

- 1 Select the condition where you want to add or remove the operator.
- 2 Right-click and select **NOT/NOT**.

Editing and deleting conditions

This table lists other ways to edit conditions in the Rule Editor:

To	Do this
Move a condition up or down	<ol style="list-style-type: none">1 Select the condition you want to move.2 Right-click and select Move Up or Move Down from the popup menu.
Edit condition properties	<ol style="list-style-type: none">1 Select the condition you want to edit.2 Right-click and select Edit from the popup menu. The property panel for the condition displays.3 Edit the properties and close the panel.
Delete a condition	<ol style="list-style-type: none">1 Select the condition you want to delete.2 Right-click and select Delete from the popup menu.

Deactivating a condition

If you want a condition not to be evaluated but do not want to delete it from the rule, you can deactivate it.

- **To toggle a condition between activate and deactivate:**
- 1 Select the condition.
 - 2 Right-click and select **Active/Inactive** from the popup menu.

Using actions


An action is a Java class that does something based on the condition it is associated with in a rule. Actions are added to the Do and Otherwise Do sections of a decision node. The Do section is activated if the When section of the node evaluates to true. The Otherwise Do section is executed when the When section is false.

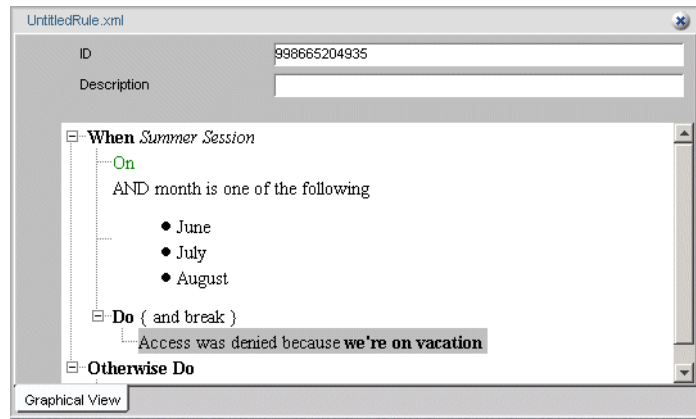
For many of the installed actions, you can set relevant properties in the Rule Editor. You can also embed cases in an action and control the processing between decision nodes.

Default action The default value for an action is return true.

To specify any other action (including return false), you need to add an action.

➤ **To add actions:**

- 1 With the Rule Editor open, select the **Do** or **Otherwise Do** section where you want to enter the action, and right-click to display the conditions.
- 2 Select **Add Action**.
The action property panel displays. The dropdown lists all the installed actions plus any custom actions you have written and deployed.
- 3 Select the action you want.
The settable properties for the condition display on the panel.
 For information about properties, see [Chapter 7, “Installed Actions”](#).
- 4 Set the properties as required and exit the panel.
The action description appears in the Rule Editor. For example, choosing the DenyAccess action displays a result like this:



To add additional actions for this node, repeat this procedure.

➤ **To add a case for an action:**

- 1 In the Rule Editor, select the action where you want to insert a case.
- 2 Right-click and select **Add Case** from the popup menu.

Editing and deleting actions

This section describes other ways to edit actions in the Rule Editor:

To	Do this
Move an action up or down	<ol style="list-style-type: none">1 Select the action you want to move.2 Right-click and select Move Up or Move Down from the popup menu.
Edit action properties	<ol style="list-style-type: none">1 Select the action you want to edit.2 Right-click and select Edit from the popup menu. The property panel for the action displays.3 Edit the properties and close the panel.
Delete an action	<ol style="list-style-type: none">1 Select the action you want to delete.2 Right-click and select Delete from the popup menu.

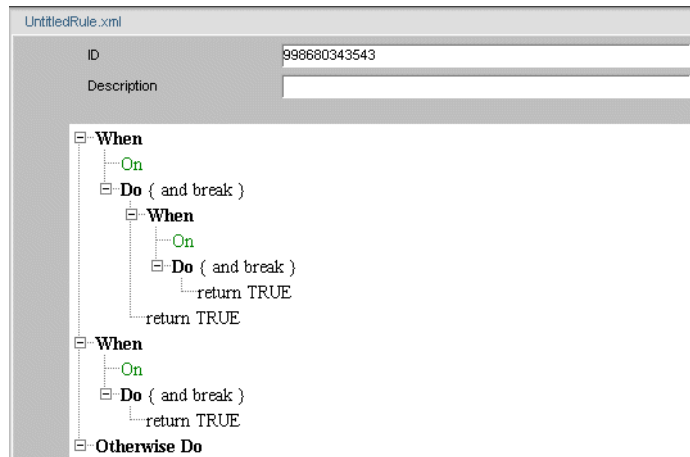
Deactivating an action

If you want a condition not to be evaluated but do not want to delete it from the rule, you can deactivate it.

- **To toggle an action between activate and deactivate:**
- 1 Select the action.
 - 2 Right-click and select **Active/Inactive** from the popup menu.

Using cases

You can add When-Do statements to a parent node to make case statements (or child nodes) in a rule. Adding cases expands the tree control for the rule. You can add a case at the top (When) or in an embedded (Do) level of your logic:



You can handle the flow of processing for cases in two ways:

- ◆ **Break** specifies that if the When section is true, the rule ends with that case.
- ◆ **Continue** specifies that after the rule does the actions for the case, it goes on to evaluate the next node.

➤ **To add a case to a rule:**

- 1 Select the **When**, **Do**, or **Otherwise Do** section where you want to add a case.
- 2 Right-click and select **Add Case** from the popup menu.
The Rule Editor adds a decision node (When-Do statement).
- 3 Repeat **Step 2** for each additional case.
- 4 Use the Rule Editor to add conditions and actions for each node.

➤ **To toggle the case processing value:**

- 1 Select the **Do** section you want to change.
- 2 Right-click and select **Break/Continue** from the popup menu.

Adding case descriptions

If you have a complex rule with many cases, you can add a description for each to make them easier to identify. The description appears next to the When section that begins the case.

➤ **To add a case description:**

- 1 Select the **When** section for the case you want to add the description to.
- 2 Right-click and choose **Edit Description** from the popup menu.
- 3 Add the description in the dialog and click **OK**.

The description appears in the Rule Editor.

Using other case commands

The Rule Editor provides various other ways to manipulate cases.

➤ **To use other case commands:**

- 1 Select the **When** section of the case.
- 2 Right-click and select the appropriate command from the popup menu:

To	Do this
Delete the case	Click Delete Case
Deactivate/activate a case	Click Active/Inactive
Copy a case	Click Copy Case
Paste a case	Select the point where you want to paste and click Paste Case
Move a case up one level in the logic	Click Move Case Up
Move a case down one level in the logic	Click Move Case Down

Testing, editing, and saving rules

Testing rules

There are a couple of ways you can test rules before deploying them:

- ◆ Using the Run Rule command
- ◆ Using the RuleWrapper portlet

Using the Run Rule command

The Run Rule command displays rule return values in the development environment Output Pane. Use this command to verify that the rule is returning the correct values at each logic point. The Run Rule command does not display actual return values, like HTML data.

➤ To run a rule:

- ◆ With the Rule Editor open, select the **Run Rule** icon from the exteNd Director menu:



Output for return values displays on the development environment Output Pane.

To clear the Output Pane:

- ◆ Place the cursor anywhere in the pane.
- ◆ Right-click and select **Clear**.

TIP: You can use the RuleWrapper portlet in the installed MyPortal application to display the result of any rule that returns HTML.

Saving and editing rules

➤ To save a rule:

- 1 With the rule open in the Rule Editor, select **File>Save** from the exteNd Director menu or press **Ctrl-S**.

The contents of the rule definitions folder display in your [project resource set](#).

- 2 Save the rule and exit the dialog.

➤ To edit a rule:

- 1 Navigate to the location of the [rule XML descriptor](#) in your project.
- 2 Double-click the rule XML file.

Working with condition and action macros

Macros allow you to save conditions and actions in reusable groups. For example, you might have standard logic that uses multiple conditions. Similarly, you might have a group of discrete actions that you want to fire under different conditions. Macros provide a higher level of organization while preserving the flexibility of using single conditions and actions.

Each macro is saved as an XML file. After you create a macro, you can add it to a condition and action in the Rule Editor.

Using condition macros

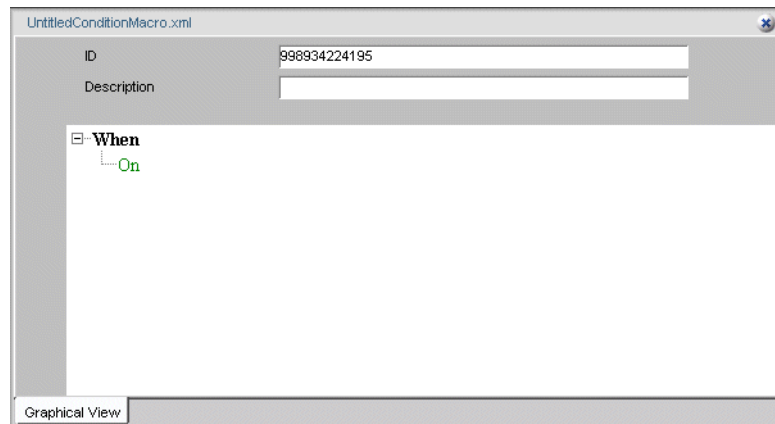
This section describes how to create a condition macro and use it in a rule.

➤ To create a condition macro:

- 1 From the exteNd Director menu, select **File>New**.
- 2 In the New File dialog, select the **Rules Engine** tab.
- 3 Select the **Condition Macro** icon:



The Condition Macro Editor displays:



- 4 Specify a name and description at the top of the Macro Editor.
The name you specify is the default file name for the macro XML file.
- 5 Click the **When** section in the tree view.
- 6 Select **Add Condition**.
The condition property panel displays.

- 7 Select a condition from the dropdown, as described in “To add a condition:” on page 60.
- 8 To add another condition, select the existing condition and right-click.
- 9 Add more conditions, as described in “To add more conditions:” on page 61.
- 10 Select an operator, as described in “Selecting a logical operator for processing” on page 61.
- 11 Edit, move, or delete conditions, as described in “Editing and deleting conditions” on page 62.

Saving and editing a condition macro

When you save a condition macro, exteNd Director opens the appropriate subdirectory of your project’s [resource set](#).

➤ To save a condition macro:

- 1 With your macro open in the Condition Macro Editor, select **File>Save** from the exteNd Director menu or press **Ctrl-S**.
The contents of the rule-conditions-macro folder display in your [project resource set](#).
- 2 Save the macro and exit the dialog.

➤ To edit a condition macro:

- 1 Navigate to the location of the [condition macro descriptor](#).
- 2 Double-click the file.

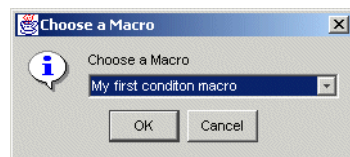
Using a condition macro in a rule

After you save the macro in an appropriate directory, you can use it in a rule.

➤ To use a condition macro in a rule:

- 1 Open the rule in the Rule Editor.
- 2 Select the point in the **When** section where you want to add the macro.
- 3 Right-click and choose **Insert Macro** or **Add Macro**.

A dropdown list of condition macro descriptions displays:



- 4 Select a macro and click **OK**.

- **To delete, inactivate, or move a macro within the rule:**
 - 1 Select the macro in the Rule Editor and right-click.
 - 2 Select the appropriate command from the popup menu.

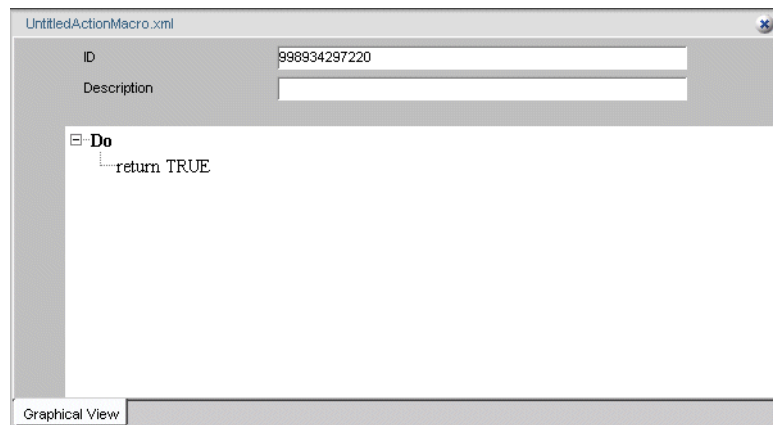
Using action macros

This section describes how to create an action macro and use it in the Rule Editor.

- **To create an action macro:**
 - 1 From the exteNd Director menu, select **File>New**.
 - 2 In the New File dialog, click the **Rules Engine** tab.
 - 3 Select the **Action Macro** icon:



The Action Macro Editor displays:



- 4 Specify a name and description at the top of the Action Macro Editor. The name you specify is the default file name for the macro XML file.
- 5 Click the **Do** section in the tree view.
- 6 Right-click and choose **Add Action**. The action property panel displays.
- 7 Select an action from the dropdown menu, as described in [“To add actions:” on page 63](#).
- 8 To add another action, select the existing action and right-click.
- 9 Add more actions, as described in [“To add a case for an action:” on page 63](#).

- 10 Edit, move, or delete actions, as described in “Editing and deleting actions” on page 64.

Saving and editing an action macro

When you save an action macro, exteNd Director opens the appropriate subdirectory of your project’s [resource set](#).

➤ To save an action macro:

- 1 With your macro open in the Action Macro Editor, select **File>Save** from the exteNd Director menu or press **Ctrl-S**.

The contents of the rule-action-macro folder display in your [project resource set](#).

- 2 Save the macro and exit the dialog.

➤ To edit an action macro:

- 1 Navigate to the location of the [action macro descriptor](#).
- 2 Double-click the file.

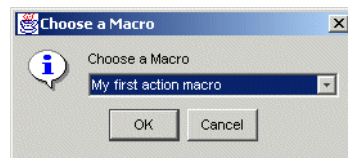
Using an action macro in a rule

After you save the action macro in an appropriate directory, you can use it in a rule.

➤ To use an action macro in a rule:

- 1 Open the rule in the Rule Editor.
- 2 Select the point in the **Do** section where you want to add the macro.
- 3 Right-click and choose **Insert Macro** or **Add Macro**.

A dropdown list of action macro descriptions displays:



- 4 Select a macro and click **OK**.

➤ To delete, inactivate, or move a macro within a rule:

- 1 Select the macro in the Rule Editor and right-click.
- 2 Select the appropriate command from the popup menu.

5

Condition and Action Wizards

This chapter describes how to use the Condition Wizard and Action Wizard in exteNd Director to create custom conditions and actions. It includes these topics:

- ◆ [Using the Condition Wizard and Action Wizard](#)
- ◆ [Using Java templates to define custom conditions and actions](#)
- ◆ [Using condition and action properties](#)
- ◆ [Deploying custom conditions and actions](#)

 For background information, see [Chapter 3, “Developing Custom Conditions and Actions”](#).

Using the Condition Wizard and Action Wizard

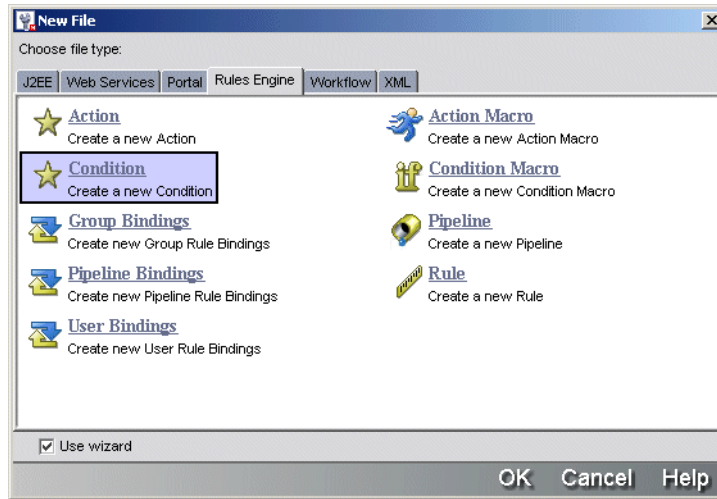
The exteNd Director installation includes a set of conditions and actions that are accessible from the Rule Editor. You can also create your own conditions and actions and access them in the same way.

exteNd Director provides a Condition Wizard and an Action Wizard to create a Java source file template.

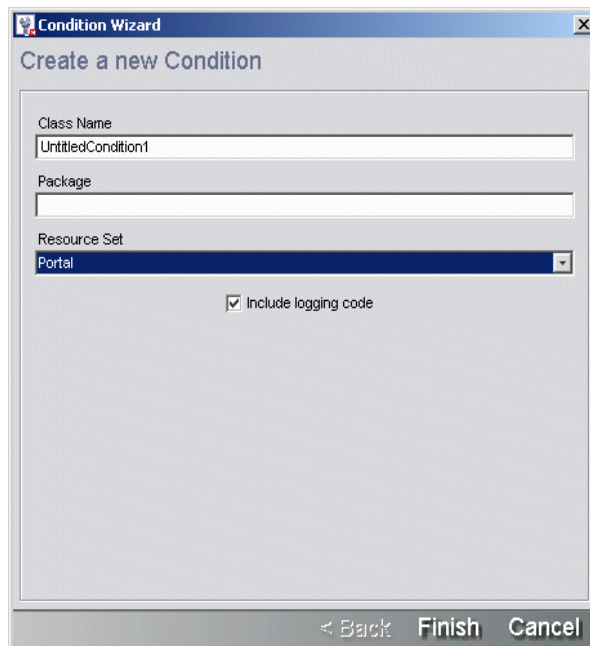
➤ **To use the Condition Wizard or Action Wizard:**

- 1** From the exteNd Director menu, select **File>New**.
- 2** In the New File dialog, select the **Rules Engine** tab.

3 Select the **Condition** or **Action** icon:



The appropriate wizard panel displays:



4 Complete the information in the wizard panel:

Option	What to do
Class Name	Enter a name for the class.
Package	(Optional) Enter a package name. By default, exteNd Director adds the class to your project source layout in the Resource Set/resource.spf/src directory. If you specify a package, it will be appended to /src.
Resource Set	Enter the resource set to use for this condition or action. The resource set determines the source and archive locations for the elements in your project. For more information, see the chapter on using the resource set in <i>Developing exteNd Director Applications</i> .
Include logging code	Check if you want the code template to include logging code for error tracing and debugging messages.

5 Click **Finish**.

Based on your information, exteNd Director creates a Java template for creating either a custom condition or a custom action.

Using Java templates to define custom conditions and actions

This section describes the methods defined in the condition and action Java templates.

 For a summary of methods, see [“About the template methods” on page 77](#).

Condition template

The defining method for a condition is `EbiCondition.doCondition()`, where you place the logic for the condition.

Here is the template generated for a condition named `myCondition` in package `myconditions`:

```
package myconditions;  
  
/**  
    myCondition
```

```

*/
public class myCondition implements com.sssw.re.api.EbiCondition
{
    // An Instance of a log for error/trace reporting
    private com.sssw.fw.api.EbiLog log =
        com.sssw.fw.log.EboLogFactory.getLog(
com.sssw.fw.log.EboLogFactory.RE );
    /**
     * return a component (JPanel) that represents the editor of
the
        condition,
     * OR return null if you would like an editor generated
automatically.
     */
    public java.awt.Component getParameterPanel () {
        return null;
    }

    public boolean doCondition( com.sssw.re.api.EbiContext context )
throws
        com.sssw.re.exception.EboConditionException {
        // Only log items if the log level indicates we should
        if ( log.isTrace() ) {
            log.trace( "myCondition in doCondition method" );
        }
        // Place your condition code here...
        return false;
    }

    public String toString() {
        return "<i>myCondition</i>";
    }
}

```

Action template

The defining method for an action is `EbiAction.doAction()`, where you place the logic for the action.

Here is the template generated for a condition named **myAction** in package **myactions**:

```

package myactions;

/**
    myAction
*/
public class myAction implements com.sssw.re.api.EbiAction
{
    // An Instance of a log for error/trace reporting
    private com.sssw.fw.api.EbiLog log =

```

```

    com.sssw.fw.log.EboLogFactory.getLog(
com.sssw.fw.log.EboLogFactory.RE );
/**
 * return a component (JPanel) that represents the editor of the
action,
 * OR return null if you would like an editor generated
automatically.
 */
public java.awt.Component getParameterPanel () {
    return null;
}


public void doAction( com.sssw.re.api.EbiContext context ) throws
com.sssw.re.exception.EboActionException {
    // Only log items if the log level indicates we should
    if ( log.isTrace() ) {
        log.trace( "myAction in doAction method" );
    }
    // Place your action code here...
    context.setResponsePhrase( "myAction" );
}




public String toString() {
    return "<i>myAction</i>";
}
}

```

About the template methods

A condition requires implementing the doCondition() method, and an action requires implementing the doAction() method. The other methods are the same for both Java condition and action templates:

Template method	Usage	What it does
getLog()	Optional	Lets you generate trace and error messages for debugging. This method appears if you checked Include logging code in the wizard.
getParameterPanel()	Optional	Lets you create a custom property panel for the condition or action. The Rule subsystem will provide default controls for some data types.  For more information, see “Using condition and action properties” on page 79.

Template method	Usage	What it does
doCondition()	Condition Wizard only Required	<p>Fulfills the requirements of the EbiCondition interface. In this method, you write code that makes comparisons or evaluates property values and returns a boolean value.</p> <p>The EbiContext object is passed to doCondition() so you can access the current session, user, and portlet.</p> <p> For more information, see “Defining logic for a condition” on page 41.</p>
doAction()	Action Wizard only Required	<p>Fulfills the requirements of the interface. In this method, you write code that performs actions based on whether the associated condition evaluates to true or false. This method typically returns the result of a business rule.</p> <p>The result returned might be a simple boolean value or information accessed from a database. You can design rules that return formatted content that the portlet can include in its generated content, or you can change the portlet’s processing based on the rule’s action value.</p> <p>The EbiContext object is passed to doAction() so that you can access the current session, user, and portlet.</p> <p> For more information, see “Defining logic for an action” on page 43.</p>
toString() method	Required	<p>Returns a String that describes the condition and its current settings as a phrase. Typically, the phrase calls the properties’ get methods to include property values. The Rule Editor displays this phrase in the rule definition.</p> <p> For more information, see “Defining a condition or action rule descriptor” on page 44.</p>

Using condition and action properties

Typically, conditions and actions are implemented as JavaBeans that include properties you can set in the Rule Editor. Here are the code elements you can use for rendering properties:

Property element type	Usage	For information see
JavaBeans	By defining properties, you allow conditions and actions to be customized in the Rule Editor for use in a variety of rules.	“Defining JavaBeans” on page 45
Generic properties	The Rule subsystem provides default controls for most data types that you can use in your JavaBean implementation.	“Using generic property panels” on page 49
Custom property panel	If you prefer to create your own property panel rather than use generic properties, you can implement the <code>getParameterPanel()</code> method.	“Creating a custom property panel” on page 50
Runtime properties	The Rule subsystem provides string template data types that allow properties in conditions and actions to be set dynamically from runtime whiteboard values.	“Defining runtime properties” on page 46
BeanInfo class	You can include a <code>BeanInfo</code> class to provide formatted property descriptions and to implement Java resource bundles for localization.	“Writing a BeanInfo class” on page 51
Resource bundles	Java resource bundles provide localized descriptions for property panels.	“Using resource bundles” on page 53

Deploying custom conditions and actions

After you have completed your condition or action source file, you need to compile it. exteNd Director then places the result in your resource set. This allows the Rule Editor to find and display the runtime version when you select it from the condition or action dropdown menu.

Compiling the condition or action source code

➤ **To compile the condition or action source:**

- 1** Highlight the file in exteNd Director Source view.
- 2** Right-click and choose **Compile** from the popup menu.
exteNd Director compiles the source. If there are no compile errors, exteNd Director stores the result in the resource set in your project archive.
Now you can select the condition or action in the Rule Editor.

Deploying support files

If you have any supporting files such as a BeanInfo class, they must be archived in an appropriate location before you can access the condition or action in the Rule Editor. The location must be specified in the libPath of your ResourceSet.war project web.xml file.

6

Pipeline and Binding Editors

This chapter describes how to use the Pipeline and Binding Editors in exteNd Director to set up a rules pipeline. It has these sections:

- ◆ [Basic steps of setting up a pipeline](#)
- ◆ [Creating and editing a pipeline](#)
- ◆ [Binding rules to a user, group, or pipeline](#)

Basic steps of setting up a pipeline

A *pipeline* is a mechanism for running a set of rules that are associated with specific users or groups, or for running any set of rules in a prescribed sequence.


Here is the basic procedure for setting up a pipeline:

- 1** Use the Pipeline Editor to create a pipeline and associate it with one or more of the three binding *steps* (types): User, Group, and Pipeline.
- 2** Use the Binding Editor to bind rules to a user, group, or pipeline.

After you set up a pipeline, you fire the rules defined in the pipeline by calling the `validate()` method in your application:

- ◆ If the pipeline includes the User or Group step, the Rule subsystem fires the rules defined in the user or group binding definition when the user or group member is in the current application context.
- ◆ If the pipeline includes the Pipeline step, the Rule subsystem fires the rules defined in the pipeline step defined for the pipeline being validated.

The rest of this chapter describes how to create a pipeline in the Pipeline Editor and define a rule binding in the Binding Editor.

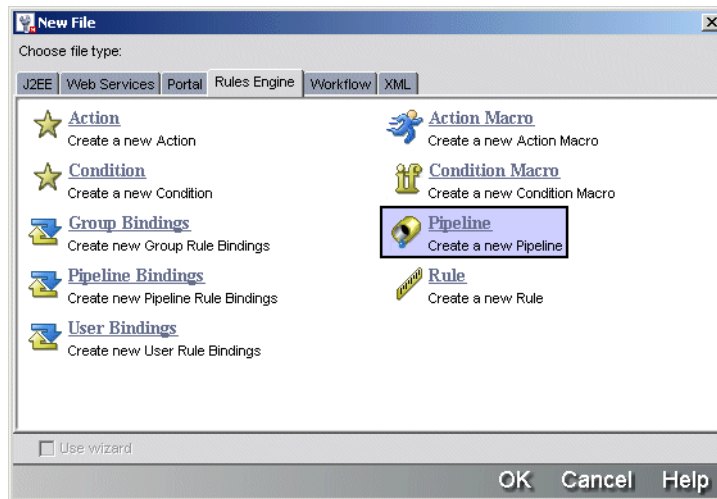
 For background information about using pipelines in exteNd Director applications, see [Chapter 2, “How You Use Rules”](#).

Creating and editing a pipeline

You use exteNd Director’s Pipeline Editor to create a pipeline and associate it with a step or steps.

Creating a pipeline

- **To create a pipeline:**
- 1 From the exteNd Director menu, select **File>New**.
 - 2 In the New File dialog, select the **Rules Engine** tab.
 - 3 Select the **Pipeline** icon:



The General tab of the Pipeline Editor displays:

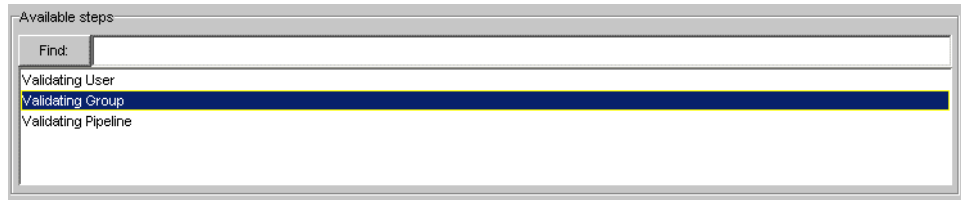


- 4** Enter information on the General tab:

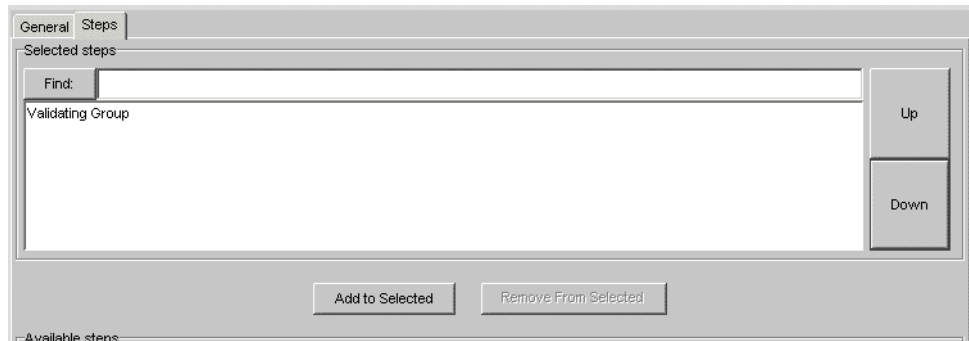
Option	What to do
ID	Enter a pipeline ID. This is the string you supply for exteNd Director API method calls, including the validate() method.
Description	(Optional) Add a description.
Active?	Select to make the pipeline active. Only active pipelines can be validated.
Generate log info?	(Optional) Select to generate pipeline processing and debugging messages to your server console.

- 5** Select the **Steps** tab.

- 6 In the **Available steps** section, select a step and click **Add to Selected**:



The step is added to the **Selected steps** section:



- 7 Add other steps as needed.
If you add more than one step, the rules defined in the corresponding binding will fire in the order of the steps in the **Selected steps** section.
 - ◆ To reorder a selected step, select it and click **Up** or **Down**.
 - ◆ To remove a selected step, select it and click **Remove from selected**.
- 8 Select **File>Save** or press **Ctrl-S**.
The contents of the pipeline definitions folder display in your [project resource set](#).
- 9 Specify a name and click **Save**.
- 10 Exit the editor by choosing the exit icon in the upper-right corner.

Editing a pipeline

➤ To edit a pipeline:

- 1 In your project source or archive layout in exteNd Director, navigate to the [pipeline XML descriptor](#).
- 2 Double-click the file to open it in the Pipeline Editor.

Binding rules to a user, group, or pipeline

You use the Binding Editor to bind a rule or set of rules to a selected user, group, or pipeline.

Creating a rule binding

➤ **To define a rule binding:**

- 1 From the exteNd Director main menu select **File>New**.
- 2 In the New File dialog, select the **Rules Engine** tab.
- 3 Select the **Group Bindings**, **Pipeline Bindings**, or **User Bindings** icon.
The appropriate binding editor displays. The name of the text box at the top of the editor depends on which icon you selected.
- 4 Make the appropriate entry in the text box:

Binding text box	What you do
User	Enter a valid user ID in the User text box. NOTE: If you are accessing an LDAP realm specify the distinguished name—for example: cn=sample,ou=users
Group	Enter a valid group ID in the Group text box. NOTE: If you are accessing an LDAP realm specify the distinguished name—for example: cn=managers,ou=management
Pipeline	Select a pipeline ID from the dropdown list.

- 5 In the **Available rules** section, select a rule and click **Add to Selected**.
The rule is added to the **Selected rules** section.
- 6 Add other rules as needed. Rules will fire in the order in which they appear in the list.
 - ◆ To reorder a rule, select it and click the **Up** or **Down** button.
 - ◆ To remove a selected rule, select it and click **Remove from selected**.
- 7 Choose **File>Save** or press **Ctrl-S**.
A dialog displays with the contents of the appropriate binding definitions in your [project resource set](#).
- 8 Specify a name and click **Save**.
- 9 Exit the editor by choosing the exit icon in the upper-right corner.

Editing a rule binding

➤ **To edit a binding:**

- 1** In your project source or archive layout in exteNd Director, navigate to the location of the binding XML definition in your project resource set:
 - ◆ [User binding descriptor](#)
 - ◆ [Group binding descriptor](#)
 - ◆ [Pipeline binding descriptor](#)
- 2** Double-click the file to open it in the Binding Editor.



Reference

Describes the Director installed conditions and actions and the JSP rules tag library

- [Chapter 7, “Installed Actions”](#)
- [Chapter 8, “Installed Conditions”](#)
- [Chapter 9, “Rule JSP Tag Library”](#)

7

Installed Actions

This chapter describes the actions installed with your exteNd Director project. It has these sections:

- ◆ [Accessing condition and action sources](#)
- ◆ [Properties that support string templates](#)
- ◆ [Properties that support database drivers and URLs](#)
- ◆ [Alphabetical list of actions](#)

 For background information, see [Chapter 4, “Rule and Macro Editors”](#).

Accessing condition and action sources

Condition and action class files, sources, and supporting files are contained in JAR files that are added to your project when you create it. The default location for the JARs in your project is ResourceSet/Web-INF/lib. These are the JARs that are added, depending on the subsystem(s) you select in the wizard:

Subsystem(s) selected	JAR file added
Rule	RuleCA.jar
Rule and Portal	PortalCA.jar
Rule and Workflow	WorkflowRE.jar
Rule, Portal, and Content Management	CQA.jar

Properties that support string templates


!valueOf template

Some of the condition and action properties support the !valueOf template. Properties that support this feature have the ^ character.

You can either enter the actual value you want to use or specify a whiteboard key that holds the value you want. Use this format:

```
!valueOf .keyname
```


You can also specify a key that holds the name of another key. To get a value from another key, specify !valueOf .anotherkey.

 For more information about the !valueOf construct, see [“Using whiteboard values” on page 26](#).

Scoped path support

The ^ template fields also support the substitution syntax for scoped paths. You can specify a scoped path using this format:

```
#{spath}
```

 For more information, see the section on [working with scoped paths](#) in *Developing exteNd Director Applications*.

Properties that support database drivers and URLs

Some of the conditions and actions have properties for accessing a database. These properties include **JDBC Driver** and **Database Name**. These values vary with the database and server vendors. For details, see your database and application server documentation.

Here is a list of database drivers and URL patterns for some of the databases supported in exteNd Director:

Database/server	Driver class name	Database name
Sybase ASA (exteNd™ application server)	com.sssw.jdbc.mss.odbc. AgOdbcDriver Default	jdbc:sssw:odbc:Database Name Default

Database/server	Driver class name	Database name
Microsoft SQL Server (extended application server)	com.sssw.jdbc.mss.odbc.AgOdbcDriver	jdbc:sssw:odbc:Database Name
Microsoft SQL Server (WebLogic)	weblogic.jdbc.mssqlserver4.Driver	jdbc:weblogic:mssqlserver4:DatabaseName@server:port NOTE: You can specify either the IP address or the server host name
DB2	COM.ibm.db2.jdbc.app.DB2 Driver	jdbc:db2:DatabaseName
Oracle Thin	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@server:port:SID NOTE: You can specify either the IP address or the server host name
Oracle (extended application server)	com.sssw.jdbc.oracle8.Driver	jdbc:sssw:oracle:DatabaseName
Oracle (WebLogic)	weblogic.jdbc.oci.Driver	jdbc:weblogic:oracle:DatabaseName
Sybase ASE	com.sybase.jdbc2.jdbc.Syb Driver	jdbc:sybase:Tds:server:port/DatabaseName NOTE: You can specify either the IP address or the server host name

Alphabetical list of actions

- ◆ Add
- ◆ Add Eraser
- ◆ Calculate Age
- ◆ Clear Request Data From Whiteboard
- ◆ Create Collection Of Objects From SQL
- ◆ Default
- ◆ Delete Cookie
- ◆ Deny Access
- ◆ Display Component
- ◆ Display Cookies
- ◆ Display Request Headers
- ◆ Display Whiteboard
- ◆ Divide
- ◆ Drop Cookie User ID
- ◆ Fire Rule
- ◆ Flush
- ◆ Format Date
- ◆ Get Cookie Value
- ◆ Get User Property
- ◆ Log User Off
- ◆ Multiply
- ◆ Query
- ◆ Remove From Whiteboard
- ◆ Return As Decimal Format
- ◆ Return As Html Body
- ◆ Return As Html Bold
- ◆ Return As Html Break
- ◆ Return As Html Checkbox
- ◆ Return As Html File Upload
- ◆ Return As Html Hidden Field
- ◆ Return As Html JavaScript
- ◆ Return As Html Option List
- ◆ Return As Html Password
- ◆ Return As Html Radio Button
- ◆ Return As Html Reset Button
- ◆ Return As Html Scripted Button
- ◆ Return As Html Submit Button

- ◆ Return As Html Table
- ◆ Return As Html Text Area
- ◆ Return As Html Text Field
- ◆ Return As XML
- ◆ Return Authentication Required
- ◆ Return False
- ◆ Return Response
- ◆ Return Response With Default
- ◆ Return True
- ◆ Save Cookies To Whiteboard
- ◆ Save Form Get Data To Whiteboard
- ◆ Save Request Data To Whiteboard
- ◆ Save To Whiteboard
- ◆ Send Mailer SMTP
- ◆ Set Component Parameter
- ◆ Set Cookie Value
- ◆ Set Date On Whiteboard
- ◆ Set Expired
- ◆ Set Next Activity
- ◆ Set Pipeline Status
- ◆ Set Response Header
- ◆ Set Response Status
- ◆ Set User Property
- ◆ Set Workitem Priority
- ◆ Set Workitem Value
- ◆ SQL Hierarchy
- ◆ SQL String
- ◆ Stop Rule Processing
- ◆ Subtract

Add

Description

Performs addition. The value you specify is added to a value stored in a whiteboard key. The whiteboard key then contains the new, incremented value. No response status or response phrase is set.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Whiteboard Key ^	The keyname of a numeric value stored on the whiteboard. If the key doesn't exist, this action adds it. If its value is not numeric, it is treated as zero for the arithmetic operation.
Value	The numeric value you want to add to the value of the whiteboard key.

Add Eraser

Description

Specifies that a whiteboard key will be removed at some future time: either after a number of seconds or after a number of times accessed.

Properties

Property	Description
Activation Counter	The number of times the whiteboard key's value can be accessed before it is deleted. Setting the key and getting the key's value both increment the activation counter.
Eraser Name	The name of the whiteboard key you want to have erased.
Seconds	The number of seconds the key will stay on the whiteboard.

Usage

To use the activation counter, set seconds to zero. To use seconds, set the activation counter to zero.

Calculate Age

Description

Calculates the difference in years between a date on the whiteboard and the current date and stores the difference in another whiteboard key. The value is saved as an integer.

Properties

Property	Description
Detail Key	The key in which you want to store the difference between the two dates.

Property	Description
Date Key	The key containing the date you want to compare to the current date. (The current date is the date the rule is run.)

Usage Code in the application must add a key to the whiteboard using the name you specified in Date Key, or you could use the **Save To Whiteboard** action. The data type of its value must be Date.

Clear Request Data From Whiteboard

Description Removes a key and its value from the whiteboard. If the key doesn't exist, nothing happens.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Detail Key ^	The name of the whiteboard key to be removed.



Usage If you use a template for Detail Key, you need to add the key to the whiteboard either in your application or through another action (see the **Save To Whiteboard** action).

Create Collection Of Objects From SQL

Description Creates a Vector of objects from a specified database column and makes the Vector available from the whiteboard through a specified detail key.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Key Column ^	The name of the column for the SQL WHERE clause.
User ID ^	The user ID expected by the database.
Detail Key ^	The name of the key that holds the object Vector name.
Password ^	The password expected by the database.
Object Name ^	The name of the object or objects to retrieve for the SQL WHERE clause.

Property	Description
SQL String ^	The SQL statement you want to execute. NOTE: If you are using a template key for this value, place the expression in quotes—for example: <code>"!valueOf.mySQL"</code>
JDBC Driver ^	The Java class name for the JDBC driver.  For more information, see “Properties that support database drivers and URLs” on page 90.
Database Name ^	The URL for the database. The format for the URL depends on the DBMS.  For more information, see “Properties that support database drivers and URLs” on page 90.

Default

Description Does nothing. Use this action when you want nothing to happen if the conditions are met.

Usage You can use the Default action as a **placeholder** as you work on a rule.
To set a **success status code**, you can use the Set Response Status action instead of the Default action. Portal code that uses the rule can check the status code to confirm that the rule ran correctly.

See also [Set Response Status](#) action

Delete Cookie

Description Deletes a cookie from the user’s Web browser or other client. If the cookie exists, this action sets the cookie’s age to zero, and that signals the browser to delete it. No status code is set. If the cookie does not exist, nothing happens.

Properties

Property	Description
Cookie Name	The name of the cookie you want to delete.

Deny Access

Description Sets the response status and response phrase. Application code can check the response status and act accordingly. The response status is set to 403 and the response phrase is set to the specified message. (Neither the console nor the log displays the response status.)

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Message ^	The text to store in the response phrase.

Usage Application code can display this text to the user to explain what this action is doing.

Display Component

Description Displays the contents of a component.

Properties

Property	Description
CID	The component ID

Display Cookies

Description Displays the cookies and their values in the Rule subsystem log. By default, the log output displays on the server console.

Usage Use this action when debugging a rule.

See also For information on configuring logs, see the chapter on [logging information](#) in *Developing exteNd Director Applications*.

Display Request Headers

Description Displays the parameters in the request header and their values in the Rule subsystem log. By default, the log output displays on the server console.

Usage Use this action when debugging a rule.

See also For information on configuring logs, see the chapter on [error handling](#) in *Developing exteNd Director Applications*.

Display Whiteboard

Description Displays the whiteboard keys and their values in the Rule subsystem log. By default, the log output displays on the server console.

Usage Use this action when debugging a rule.

See also For information on configuring logs, see the chapter on [logging information](#) in *Developing exteNd Director Applications*.

Divide

Description Performs division. The value you specify is divided into a value stored in a whiteboard key. The whiteboard key then contains the quotient of the division. If the divisor value is zero, the whiteboard key's value is set to the text **Infinity**. No response status or response phrase is set.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Whiteboard Key ^	The keyname of a numeric value stored on the whiteboard. If the key doesn't exist, this action creates it. If its value is not numeric, it is treated as zero for the division operation.
Value	The numeric value you want to divide into the value of the whiteboard key.

Drop Cookie User ID

Description Drops a cookie named userID into the browser with a value of the portal user's ID. No response status or response phrase is set. If the user has disabled cookies, nothing happens. Success or failure is reported in the log; the default log output is displayed on the server console.

Properties

Property	Description
Maximum Number of Days	The number of days before the cookie expires.

Fire Rule

Description Executes a rule.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Rule ID ^	The ID of the rule you want to fire. Specify the ID (it is displayed in the Rule Editor).

Flush

Description Tells the system to empty the caches that hold rules.

Format Date

Description Gets a whiteboard Timestamp value stored in a specified Date key, formats it for the current locale, and saves the result to the whiteboard in the specified Detail key. If the value for Date Key is not a valid Timestamp object, the Detail Key is set to an empty string.

Properties

Property	Description
Detail Key	The name of the key used to save the formatted date text.
Date Key	The name of the key containing a Timestamp object. The Timestamp specifies the date you want to format.

See also [Set Date On Whiteboard](#) action

Get Cookie Value

Description Gets the value of the specified cookie. Sets the response phrase to the cookie value.

Properties

Property	Description
Stop Processing	When checked, this action terminates the rule after getting the cookie value, regardless of the continue setting for the case.
Cookie Name	The name of the cookie whose value you want.

Get User Property

Description Gets the value of a user attribute for the current portal context. The attribute must be set on the whiteboard in the specified key.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Key ^	The name of the attribute.

See also [Set User Property](#) action
[Check User Property](#) condition

Log User Off

Description Resets the user to Anonymous and removes all key/value pairs from the whiteboard. No response status or response phrase is set.

Multiply

Description Performs multiplication. The value you specify is multiplied by a value stored in a whiteboard key. The whiteboard key then contains the result of the multiplication. No response status or response phrase is set.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Whiteboard Key ^	The keyname of a numeric value stored on the whiteboard. If the key doesn't exist, this action creates it. If the key's value is not numeric, it is treated as zero for the multiplication operation.
Value	The numeric value you want to multiply by the value of Whiteboard Key.

Query

Description Returns the result of a query to the Content Management subsystem as an XML string.

Usage This action is used by the Content Query sample application.

See also Section on [using the Content Query action](#) in the *Content Management Guide*.

Remove From Whiteboard

Description Removes a value and its key from the whiteboard. No response status or response phrase is set.

Properties

Property	Description
Whiteboard Key	The name of the key you want to remove.

Return As Decimal Format

Description Formats a decimal value as text using the specified formatting. The formatted number is stored as text in the key specified by Detail Key. The response type is set to TEXT and the response status is set to 302.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see “[Properties that support string templates](#)” on page 90.

Property	Description
Max. Number of Integers	The maximum number of digits in the formatted number. If there are more digits in the actual number, digits on the left are dropped.
Min. Number of Integers	The minimum number of digits in the formatted number. If the number has fewer digits, extra zeros are added on the left.
Detail Key ^	The name of a key that holds the number you want to format. If the key doesn't exist, this action creates it. If the key's value is not numeric, the number is set to zero.
Min. Decimal Places	The minimum number of decimal places. Zeros are added on the right if necessary.
Max. Decimal Places	The maximum number of decimal places. Digits are truncated on the right if necessary.
Grouping	Select this to include a separator character every so many digits, specified by the grouping size and counting left from the decimal point. In an American locale, the separator is a comma.
Grouping Size	The number of digits between separators.
Mask	Specifies special characters to add to the formatted number, such as currency or percent. Other formatting you might put in a mask (such as commas, periods, or dashes) is ignored.

Usage

You can add special characters such as currency and percent signs. This action does not support formatting as phone numbers and other nonarithmetic formats.

Return As Html Body

Description

Returns the opening of an HTML BODY tag with a color attribute. This action:

- ◆ Sets the response phrase to `<body bgColor="colorvalue">`
- ◆ Sets the response type to TEXT
- ◆ Sets the response status to 302
- ◆ Ends rule processing

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Background Color ^	A color value specified as text that would be recognized by a browser. You can specify a hexadecimal value (such as #FFFFFF) or a browser-supported color name.

Return As Html Bold

Description

Returns the specified text enclosed in HTML bold tags (sets the response phrase to value). No response status or response type is set.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Value ^	The text you want to format with bold tags.

Return As Html Break

Description

Returns the specified text preceded or followed by HTML break tags, setting the response phrase to one of the following:

- ◆ If you select Before:
value
- ◆ If you select After: value

- ◆ If you select both:
value

This action does not set the response status or response type.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Value ^	The text to which you want to add break tags.
After	When selected, a break tag is added after the text.
Before	When selected, a break tag is added before the text.

Return As Html Checkbox

Description Returns HTML for a check box. Sets the response phrase to HTML that looks something like this:

```
<INPUT TYPE="CHECKBOX" VALUE="important" NAME="cb1" CHECKED>
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Name ^	A name for the check box, which becomes the value of the NAME attribute.
Stop Processing	When selected, the rule ends after this action.
Value ^	The value assigned to the VALUE attribute of the check box. This value is returned to the server when the check box is selected.
Checked	When selected, adds the CHECKED attribute to the HTML.
Append Response?	When selected, adds the HTML to the response phrase after any other text that is already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.

Return As Html File Upload

Description Returns HTML for a file upload field. Sets the response phrase to HTML that looks something like this:

```
<INPUT TYPE="FILE" VALUE="Select a file" NAME="fileupload"
ACCEPT="image/*, text/html">
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Name ^	A name for the field, which becomes the value of the NAME attribute.

Property	Description
Stop Processing	When selected, the rule ends after this action.
Accept ^	A comma-separated list of MIME types for the ACCEPT attribute. These are the file types the user can select in the field. The attribute is not supported by all browsers.
Value ^	The value assigned to the VALUE attribute of the field. In an HTML page, the value is displayed in the field and replaced by the value the user enters.
Append Response?	When selected, adds the HTML to the response phrase after any other text that is already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.

Return As Html Hidden Field

Description

Returns HTML for a hidden field. Sets the response phrase to HTML that looks something like this:

```
<INPUT TYPE="HIDDEN" VALUE="important" NAME="hfld1">
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Name ^	A name for the hidden field, which becomes the value of the NAME attribute.
Stop Processing	When selected, the rule ends after this action.
Value ^	The value assigned to the VALUE attribute of the field. In an HTML page, the value is returned to the server when the user submits the enclosing form.
Append Response?	When selected, adds the HTML to the response phrase after any other text that is already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.

Return As Html JavaScript

Description Returns the JavaScript code you specify enclosed in HTML SCRIPT tags. Sets the response phrase to HTML that looks something like this:

```
<SCRIPT LANGUAGE=JavaScript> [your code here] </SCRIPT>
```

This action sets the response type to TEXT and the response status to 302.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
JavaScript ^	The JavaScript code to be returned.

Return As Html Option List



Description Performs a SQL query and creates an HTML SELECT element from the result set. Sets the response phrase to HTML that looks something like this:

```
<SELECT id=opt1 name=opt1 size=1 width= 100">  
<OPTION value="portalcorpid">Official  
<OPTION value="anonymous">Anonymous  
<OPTION value="administrator">User0  
<OPTION value="contentadmin">User1  
<OPTION value="default">null  
<OPTION value="sample">Sample  
<OPTION value="testID">Smith  
</SELECT>
```

This action sets the response type to TEXT and the response status to 302. The result set and SQL statement are stored on the whiteboard (see the Detail Key property in the table below). If you specify a Detail Key, the rule ends after this action.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
HTML Tag	The name of the SELECT element. The value is used for the NAME attribute.
User ID ^	The user ID expected by the database. This is not usually the same as the portal user ID.

Property	Description
SQL String ^	A SQL statement whose result set contains the values for the list. NOTE: If you are using a template key for this value, place the expression in quotes—for example: <code>"!valueOf.mysql"</code>
Key Column	The column whose values will be used for the VALUE attribute of each OPTION element.
Size ^	The value for the SIZE attribute, which specifies the number of items to display in the list. Specify 1 for a dropdown list.
Database Name ^	The URL for the database. The format for the URL depends on the DBMS.  For more information, see “Properties that support database drivers and URLs” on page 90 .
JDBC Driver ^	The Java class name for the JDBC driver.  For more information, see “Properties that support database drivers and URLs” on page 90 .
Password ^	The password expected by the database. This is not usually the same as the portal user’s password.
Detail Key	A name used to create whiteboard keys for the result set and SQL statement. The name is used for the result set, and <code>name.sql</code> is used for the SQL statement.
Children Column	The column containing related information. Generally this is left blank; it requires the database to be organized with parent/child relationships.
Width (%) ^	The value for the WIDTH attribute, which specifies the width of the list. This attribute is ignored by some browsers.
Description Column	The column whose values will be used as the text of each OPTION element.

Return As Html Password

Description

Returns HTML for a password input field. Sets the response phrase to HTML that looks something like this:

```
<INPUT TYPE="PASSWORD" VALUE="PASSWORD" NAME="default" SIZE="25"
MAXLENGTH="50" READONLY>
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Read-only	Whether the user is allowed to enter a value.
Name ^	A name for the password field, which becomes the value of the NAME attribute.
Stop Processing	When selected, the rule ends after this action.
Maximum Length ^	The maximum number of characters allowed for the password.
Size ^	The value for the SIZE attribute, which is the width of the field.
Value ^	The value assigned to the VALUE attribute of the field. In an HTML page, this is the default value for the password field. Leave it blank if you don't want a default.
Append Response?	When selected, adds the HTML to the response phrase after any other text that is already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.

Return As Html Radio Button

Description

Returns HTML for a radio button. Sets the response phrase to HTML that looks something like this:

```
<INPUT TYPE="RADIO" VALUE="One" NAME="radio1" CLASS="INPUT">
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Name ^	A name for the radio button, which becomes the value of the NAME attribute. In HTML, to create a group of radio buttons that work together, give them all the same name.
Stop Processing	When selected, the rule ends after this action.

Property	Description
Value ^	The value for the VALUE attribute. In an HTML page, the value is returned to the server when the radio button is selected and its enclosing form is submitted.
Checked	When selected, adds the CHECKED attribute to the HTML.
Append Response?	When selected, adds the HTML to the response phrase after any other text that is already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.

Return As Html Reset Button

Description Returns HTML for a reset button. Sets the response phrase to HTML that looks something like this:

```
<INPUT TYPE="RESET" VALUE="Reset" >
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Stop Processing	When selected, the rule ends after this action.
Value ^	The value for the VALUE attribute, which is the text displayed on the button.
Append Response?	When selected, adds the HTML to the response phrase after any other text already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.

Return As Html Scripted Button

Description Returns HTML for a reset button. Sets the response phrase to HTML that looks something like this:

```
<INPUT TYPE="BUTTON" VALUE="Cancel" onclick="JavaScript code here" >
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Stop Processing	When selected, the rule ends after this action.
Value ^	The value for the VALUE attribute, which is the text displayed on the button.
Append Response?	When selected, adds the HTML to the response phrase after any other text already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.
OnClick ^	JavaScript code to be run when the button is clicked.

Return As Html Submit Button

Description

Returns HTML for a submit button. Sets the response phrase to HTML that looks something like this:

```
<INPUT TYPE="SUBMIT" VALUE="Submit">
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties



Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Stop Processing	When selected, the rule ends after this action.
Value ^	The value for the VALUE attribute, which is the text displayed on the button.
Append Response?	When selected, adds the HTML to the response phrase after any other text already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.

Return As Html Table

Description Constructs an HTML table tag, using the result set from the SQL query specified to create associated HTML table row and table data tags, and sets the result in the specified detail key.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
HTML Tag	Appends a NAME or ID attribute to the TABLE tag element.
User ID ^	The user ID expected by the database.
SQL String ^	The SQL statement you want to execute.
Border ^	The value for the HTML table BORDER attribute.
Database Name ^	The URL for the database. The format for the URL depends on the DBMS.  For more information, see “Properties that support database drivers and URLs” on page 90 .
JDBC Driver ^	The Java class name for the JDBC driver.  For more information, see “Properties that support database drivers and URLs” on page 90 .
Query String ^	The value for the HTML hyperlink query string.
Cell Spacing ^	The value for the HTML table CELLSPACING attribute.
Display Headings	When selected, appends the HTML table row for the table headings to the output buffer.
Password ^	The password expected by the database. This is not necessarily the same as the application user’s password.
Cell Padding ^	The value for the HTML table CELLPADDING attribute.
Detail Key	A name used to create a whiteboard key for the result set and SQL statement. The name is used for the result set, and <code>name.sql</code> is used for the SQL statement.
Width ^	The value for the HTML table WIDTH attribute.
Description Column	The value for the column name in the result set whose data is used for the HTML OPTION tag DESCRIPTION.

Return As Html Text Area

Description Returns HTML for a multiline text field. Sets the response phrase to HTML that looks something like this:

```
<TEXTAREA NAME="txt1" ROWS="25" COLS="50">This is where to enter  
text</TEXTAREA>
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Name ^	The name for the field, assigned to the NAME attribute.
Stop Processing	When selected, the rule ends after this action.
Columns ^	The width of the text field (specified as a number of characters) assigned to the COLS attribute.
Value ^	The default text to be displayed in the text field.
Rows ^	The length of the text field (specified as a number of characters) assigned to the ROWS attribute.
Append Response?	When selected, adds the HTML to the response phrase after any other text that is already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.

Return As Html Text Field

Description Returns HTML for a single-line text field. Sets the response phrase to HTML that looks something like this:

```
<INPUT TYPE="TEXT" VALUE="TEXT" NAME="default" SIZE="25"  
MAXLENGTH="50" READONLY>
```

This action sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Read Only	When selected, the text cannot be edited.

Property	Description
Name ^	The name for the input field, assigned to the NAME attribute.
Stop Processing	When selected, the rule ends after this action.
Maximum Length ^	The maximum number of characters allowed, assigned to the MAXLENGTH attribute.
Size ^	The width of the text field (specified as a number of characters) assigned to the COLS attribute.
Value ^	The default text displayed in the text field.
Append Response?	When selected, adds the HTML to the response phrase after any other text already there.
Disabled	When selected, adds the DISABLED attribute to the HTML.

Return As XML

Description

Submits a SQL query and formats the result set as XML. The root element is **table**, and its subelements are **columns** and **tuples**:

Subelement	Description
columns	Contains column elements that identify the column names.
tuples	Are the rows of data. Each tuples element contains a tuple element for each row. Each tuple element contains a subelement for each column in the result set. The order of these match the column elements. The order of the elements must be preserved.

Sets the response phrase to XML that looks something like this:

```
<table description="test user list" key="test" sqlString="select *
from FWUSERS">
  <columns count="3">
    <column value="USERID"/>
    <column value="FIRSTNAME"/>
    <column value="LASTNAME"/>
  </columns>
  <tuples count="2">
    <tuple number="1">
      <tuple value="portalcorpid"/>
      <tuple value="Portal Corporate Administrator"/>
      <tuple value="unknown"/>
    </tuple>
    <tuple number="2">
```

```



        <tuple value="anonymous"/>
        <tuple value="Portal Anonymous User"/>
        <tuple value="unknown"/>
    </tuple>
</tuples>
</table>

```

This action sets the response type to TEXT and the response status to 302. The result set and SQL statement are stored on the whiteboard (see the Detail Key property in the table below). If you specify a Detail Key, the rule ends after this action.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
User ID ^	The user ID expected by the database. This is not usually the same as the portal user ID.
Detail Key	A name used to create whiteboard keys for the result set and the SQL statement. The name is used for the result set, and name.sql is used for the SQL statement.
Password ^	The password expected by the database. This is not usually the same as the portal user's password.
Description	Text that will be the value of a description attribute assigned to the root element, which is called table .
SQL String ^	A SQL statement whose result set contains the data to be formatted as XML elements. NOTE: If you are using a template key for this value, place the expression in quotes—for example: "!valueOf.mySQL"
JDBC Driver ^	The Java class name for the JDBC driver.  For more information, see “Properties that support database drivers and URLs” on page 90 .
Database Name ^	The URL for the database. The format for the URL depends on the DBMS.  For more information, see “Properties that support database drivers and URLs” on page 90 .

Return Authentication Required

Description Sets the context response status to 401, forcing the browser to ask for authentication.

Return False

Description Sets the response status to 412, which signifies the boolean value false.

Return Response

Description Sets the response phrase to the specified text. You can use !valueOf templates to build a text value from phrases stored on the whiteboard. Sets the response type to TEXT and the response status to 302. If Stop Processing is checked, rule processing ends.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Stop Processing	When selected, the rule ends after this action.
Response ^	The response phrase.
Append Response?	When selected, adds the HTML to the response phrase after any other text already there.

Return Response With Default

Description Returns text associated with a whiteboard key or uses a default message. Sets the response phrase to the message, stores the message on the whiteboard, sets the response type to TEXT, sets the response status to 302, and stops rule processing.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Default ^	A message to use if the Response key doesn't exist. If the Response key doesn't exist or has a null value, a new key is created whose name is the Response keyname and whose value is the Default message.
Response ^	A whiteboard key containing the message you want.

Return True

Description Sets the response status to 200, which signifies the boolean value true.

Save Cookies To Whiteboard


Description Puts the cookies the browser has sent onto the whiteboard. The whiteboard keys are created using the cookie name as the keyname and the cookie's value as the value for the key.

See also [Save Cookies To Whiteboard](#) condition

Save Form Get Data To Whiteboard

Description Puts the query string from the browser on the whiteboard using the specified key.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Whiteboard Key ^	The name of the whiteboard key.  See “Properties that support string templates” on page 90 .

See also [Save Form Get Data To Whiteboard](#) condition

Save Request Data To Whiteboard

Description Puts the parameters in the request header on the whiteboard. The names of the request parameters are used as the whiteboard keys, and the values of the request parameters are their values.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
AutoClear	If this action has run before and the Detail Key includes a set of previously created keys, those keys are removed from the whiteboard before new keys are created from the current request data.
Detail Key ^	A whiteboard key in which to store the names of the keys being created. NOTE: The keys are stored in a Vector.

See also

[Save Request Data To Whiteboard](#) condition

Save To Whiteboard

Description

Saves a value on the whiteboard using the specified keyname. The value is set on the whiteboard. No response phrase or status is set.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Whiteboard Key ^	The name to use for the whiteboard key where you want to save the value.
Data ^	The value you want to save on the whiteboard.
Resolve	When selected, !valueOf templates in the Data property are processed and the stored value contains the result. When not selected, the text is not evaluated and the whiteboard contains the text as entered.

Send Mailer SMTP

Description

Sends an e-mail message to a specified e-mail address. No response phrase or status is set.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Subject	The subject line for the e-mail.
To	A valid e-mail address, in the form recipient@yourcompany.com .
SMTP Host	An SMTP mail host. These usually have the form smtp.yourcompany.com or mail.yourcompany.com .
Message ^	The text of the e-mail.
From	The sender's e-mail address, which is an e-mail address associated with your portal application.

Usage

Use this action to send alerts or logging information to an administrator.

Set Component Parameter

Description

Sets the value of a component parameter for the current portal context. The component parameter must already exist.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Key ^	The name of the parameter.
Value ^	The value for the parameter.

Set Cookie Value

Description

Creates a cookie or changes its value and expiration date. No response phrase or status is set.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Maximum Number of Days	The number of days the cookie will exist before the user's browser deletes it.


Property	Description
Cookie Value ^	The text value assigned to the cookie.
Cookie Name	The cookie's identifier.

Set Date On Whiteboard

Description

Sets the current date on the whiteboard with a specified key and format. The default format is a timestamp, and the default time is the runtime value. You can specify other formats and time values.

Properties

Property	Description
Key	The key for the date value.
Format	<p>The date format. For example:</p> <pre>yyyy-MM-dd k:mm:ss.S</pre> <p>Default time is current (runtime). You can also specify a time value. For example, the following indicates today at 12 p.m.:</p> <pre>yyyy-MM-dd 00:00:00.000</pre> <p> For information:</p> <ul style="list-style-type: none"> ◆ About date formats, see the API documentation for <code>java.text.SimpleDateFormat</code> ◆ About date values, see the API documentation for <code>java.util.Date</code>

Set Expired

Description

Sets the context response headers in the browser to be expired. No response phrase or status is set.

Set Next Activity

Description Sets the destination link in a workflow process.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Addressee ^	User activities only: the user, group, or role defined for the activity. For nonuser activities, use the default. NOTE: if you are accessing an LDAP realm, specify the distinguished name—for example: <code>cn=sample,ou=users</code>
isRole	Check this if the Addressee field is a role.
Activity Name ^	The name of the activity, as defined in the workflow process.

Usage This action is for use with a Rule link in the Workflow Editor.

Set Pipeline Status

Description Sets the active pipeline’s canProcessRequests flag to On/Off for this action.

NOTE: The active status is changed for this action only.

Properties

Property	Description
On	When selected, the pipeline is active for this action.

Set Response Header

Description Sets a named parameter value in the HTTP response header (EbiResponse) using the keyname and value you specify. No response phrase or status is set.


Properties

Property	Description
Key	The name of the parameter being added to the response header.
Value	The text value associated with the parameter name.

Set Response Status

Description Sets the response status to the value you specify. If your rule performs several actions, the response status will be the last status set.

Properties

Property	Description
Status	A status value. You must use one of the numeric values identified in EbiResponse.  For more information, see the <i>API Reference</i> .

Usage To control the status value, make the Set Response Status action the last one in the rule.

Set User Property

Description Sets the value of a user attribute for the current portal context. The user attribute must exist.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Key ^	The name of the user attribute.
Value ^	The value for the user attribute.

Set Workitem Priority

Description Sets the specified value as an Integer associated with the current workitem.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Priority ^	A string value.

Usage This action is for use with a Rule activity or a Rule link in a workflow process. If you use this action with a Rule link, be sure the action section also sets the next destination. See [Set Next Activity](#) action.

Set Workitem Value

Description Sets the property value for the document associated with a workitem in a workflow process. The document and property must exist and already have been added to the workitem. This action handles locking and unlocking of the document to set the property.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Document Property ID ^	The ID for the property
Document ID ^	The document ID
Property Value ^	The property value to set

Usage This action is for use with a Rule activity or a Rule link in a workflow process. If you use this action with a Rule link, be sure the action section also sets the next destination. See [Set Next Activity](#) action.

SQL Hierarchy



Description Does the same as the [SQL String](#) action (next). There is no corresponding condition.

SQL String

Description Executes a SQL statement and stores the result on the whiteboard. If more than one row is returned, only the last row in the result set is saved to the whiteboard.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
User ID ^	The user ID expected by the database.

Property	Description
Password ^	The password expected by the database. This is not necessarily the same as the portal user's password.
SQL String ^	The SQL statement you want to execute. NOTE: If you are using a template key for this value, place the expression in quotes—for example: <code>"!valueOf.mysql"</code>
JDBC Driver ^	The Java class name for the JDBC driver.  For more information, see “Properties that support database drivers and URLs” on page 90.
Database Name ^	The URL for the database. The format for the URL depends on the DBMS.  For more information, see “Properties that support database drivers and URLs” on page 90.

See also [SQL String](#) condition

Stop Rule Processing

Description Ends processing of the current rule by throwing `EboActionException`. If the rule has been invoked as part of a rules pipeline, it ends the pipeline too.

Subtract

Description Performs subtraction. The value you specify is subtracted from a value stored in a whiteboard key. The whiteboard key then contains the result of the subtraction. No response status or response phrase is set.

Properties Any property with the ^ character supports the `!valueOf` template construct. For more information, see [“Properties that support string templates” on page 90.](#)

Property	Description
Whiteboard Key ^	The keyname of a numeric value stored on the whiteboard. If the key doesn't exist, this action creates it. If the key's value is not numeric, it is treated as zero for the arithmetic operation.
Value	The numeric value you want to subtract from the value of the whiteboard key.

8

Installed Conditions

This chapter describes the conditions installed with your exteNd Director project.



For information:

- ◆ About how to use conditions and actions in the Rule Editor, see [Chapter 4, “Rule and Macro Editors”](#).
- ◆ About accessing sources and supporting files, see [“Accessing condition and action sources” on page 89](#).

Alphabetical list of conditions

- ◆ [Check Component Parameter](#)
- ◆ [Check Date](#)
- ◆ [Check Date Within Range](#)
- ◆ [Check Day](#)
- ◆ [Check For Cookie](#)
- ◆ [Check Month](#)
- ◆ [Check Request Data](#)
- ◆ [Check Time](#)
- ◆ [Check User](#)
- ◆ [Check User Group](#)
- ◆ [Check User Property](#)
- ◆ [Check Whiteboard](#)
- ◆ [Check Whiteboard Value](#)
- ◆ [Check Whiteboard Value Is Empty](#)
- ◆ [Check Workitem Value](#)

- ◆ Default
- ◆ Is Form Get Data Available
- ◆ Is New Session
- ◆ Save Cookies To Whiteboard
- ◆ Save Form Get Data To Whiteboard
- ◆ Save Request Data To Whiteboard
- ◆ Set Action Off
- ◆ Set Action On
- ◆ Set Action On Or Off
- ◆ PSQL Check For Column
- ◆ SQL String

Check Component Parameter

Description Compares the portal component parameter value with a value you specify. If the comparison is true, the condition returns true.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Key ^	The name of the component parameter.
Compare	The type of comparison you want to make. Select from the dropdown list: <ul style="list-style-type: none"> ◆ For numeric values: use the various equal to, greater than, and less than conditions ◆ For text: use equal, begins, ends, and contains
Value ^	The value to which you want to compare the parameter.

Check Date

Description Compares the current date with the date you specify and returns true if they are the same.

Properties

Property	Description
Date	The date you want to check against the current date. Click Pick Date to use a calendar tool to select a date: <ul style="list-style-type: none">◆ The << and >> buttons change the year◆ The < and > buttons cycle through the months of the year

Check Date Within Range

Description

Returns true when the current date is within the specified range. If the date is out of range or if you don't specify a date for both start and end, the condition returns false.

Properties

Property	Description
Start Date	The start date of the range. Click the button to use a calendar tool to select a date.
End Date	The end date of the range. Click the button to use a calendar tool to select a date: <ul style="list-style-type: none">◆ The << and >> buttons change the year◆ The < and > buttons cycle through the months of the year
Inclusive	Whether the start and end dates are considered part of the range: <ul style="list-style-type: none">◆ If selected, the dates are included—the current date must be greater than or equal to the start and less than or equal to the end◆ If not selected, the current date must be greater than the start and less than the end

Check Day

Description

Returns true when the current weekday is in the set of selected weekdays.

Properties

Property	Description
List of weekdays	Check one or more weekdays to select them.

Check For Cookie

Description Returns true if the specified cookie is defined in the user's browser.

Properties

Property	Description
Cookie Name	The name of the cookie you want to find.

Check Month

Description Returns true when the current month is in the set of selected months.

Properties

Property	Description
List of months	Check one or more months to select them.

Check Request Data

Description Checks whether a parameter value in the request header matches a value you specify. If the comparison is true, the condition returns true.

Properties

Property	Description
Key	The name of a request parameter.
Value	The value to which you want to compare the parameter.
Condition	The type of comparison you want to make: <ul style="list-style-type: none">◆ For numeric values: use the various equal to, greater than, and less than conditions◆ For text: use equal, begins, ends, and contains

Check Time

Description Returns true if the current time is between the hours you specify.

Properties

Property	Description
hours of ... and ...	Select an hour from each of the two dropdown lists to specify the start and end of the time span.

Check User

Description

Compares the current user with the specified user based on the text of the user ID. If the comparison is true, the condition returns true.

Properties

Property	Description
User	<p>A user ID defined in your server directory realm.</p> <p>If you are accessing an LDAP realm, use the distinguished name—for example:</p> <pre>cn=user,ou=users</pre>
Condition	<p>The relational operator for the comparison:</p> <ul style="list-style-type: none">◆ The most useful comparisons are equal and not equal◆ Greater than and less than comparisons are based on the ASCII values of the characters in the ID◆ Starts with, ends with, and contains are less useful, since the condition compares the whole user ID (you can't specify part of an ID for the comparison)

Check User Group

Description

Returns true if the user is a member of the specified group in the authentication directory.

NOTE: Users can be assigned to groups in the DAC.

Properties

Property	Description
Group	A group name defined in your server directory realm. NOTE: If you are accessing an LDAP realm, use the distinguished name—for example: <code>cn=managers,ou=management</code>

Check User Property

Description

Compares the value of a user attribute for the current context with a value you specify in the condition property. Returns true if they are the same.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Key ^	The name of the user attribute.
Compare	The type of comparison you want to make. Select from the dropdown list: <ul style="list-style-type: none">◆ For numeric values: use the various equal to, greater than, and less than conditions◆ For text: use equal, begins, ends, and contains
Value ^	The value to which you want to compare the attribute value.

Check Whiteboard

Description

Returns true if the specified whiteboard key exists.

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Whiteboard Key ^	The name of the whiteboard key.

Check Whiteboard Value

Description Compares the value of the specified whiteboard key to a value based on the data type of the value in the whiteboard key. If the comparison is true, the condition returns true.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Value	The value to which you want to compare the whiteboard value. The condition will convert the value you specify to the data type of the whiteboard value. If the value can't be converted, the condition returns false.
Whiteboard Key ^	The name of the whiteboard key.
Condition	Select the relational operator for the comparison.

Check Whiteboard Value Is Empty

Description Returns true if the value of a whiteboard key is null or an empty string (“”).

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Whiteboard Key ^	The name of the whiteboard key.

Check Workitem Value

Description Compares a value you specify to a workitem document value and returns true if the values are the same.

Properties Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
Value	The value to use for the comparison.
Document ID ^	The document ID containing the property to compare with.

Property	Description
Document Property ID ^	The property ID for the compare value.
Condition	Select the relational operator for the comparison.

Usage For use with a Rule activity or Rule link in the Workflow Designer.

See also [Workflow Designer](#) chapter in the *Workflow Guide*

Default

Description This condition does nothing—by returning false. If Default is the only condition in a When section, the actions for that case won't be executed.

Is Form Get Data Available

Description Returns true if the request header contains parameters from an HTML form.

Is New Session

Description Returns true if this is the first request for this user's session.

Save Cookies To Whiteboard

Description Puts the cookies the browser has sent onto the whiteboard. Returns true if successful and false if an error occurs.

See also [Save Cookies To Whiteboard](#) action

Save Form Get Data To Whiteboard

Description Puts the query string from the browser on the whiteboard using the specified key. Returns true if successful and false if an error occurs.

See also [Save Form Get Data To Whiteboard](#) action

Save Request Data To Whiteboard

Description Puts the parameters in the request header on the whiteboard. The names of the request parameters are used as the whiteboard keys, and the values of the request parameters are their values. Returns true if successful and false if an error occurs.

See also [Save Request Data To Whiteboard](#) action

Set Action Off

Description Returns false. If it is the only condition in a When section, the actions for that case are **not** executed. If there are other conditions, the actions may or may not be executed depending on the values of the other conditions and the logical operators used.

Set Action On

Description Returns true. If it is the only condition in a When section, the actions for that case are executed. If there are other conditions, the actions may or may not be executed depending on the values of the other conditions and the logical operators used.

Set Action On Or Off

Description Returns true if the On? property is selected and false if it is not.

Properties

Property	Description
On?	Whether the condition returns true or false.

See also [Set Action Off](#) and [Set Action On](#) conditions (just above)



PSQL Check For Column

Description Executes a SQL query with a WHERE clause and returns true if the result set contains any rows. The SQL query is built from the condition's properties and looks like this:

```
SELECT fields FROM table WHERE column condition value
```

Properties

Any property with the ^ character supports the !valueOf template construct. For more information, see [“Properties that support string templates” on page 90](#).

Property	Description
User ID ^	The user ID expected by the database.
Column	The name of a column for the WHERE clause.
Database Name ^	The URL for the database. The format for the URL depends on the DBMS.  For more information, see “Properties that support database drivers and URLs” on page 90 .
Condition	A SQL comparison operator for the WHERE clause.
JDBC Driver ^	The Java class name for the JDBC driver.  For more information, see “Properties that support database drivers and URLs” on page 90 .
Fields	The columns to include in the result set. Use an asterisk (*) to specify all columns.
Value	The value you want to match in the WHERE clause.
Password ^	The password expected by the database. This is not necessarily the same as the portal user’s password.
Table	The table for the SQL query.

SQL String

Description

Executes a SQL query and returns true if the result set contains any rows.

See also


[SQL String](#) action

9

Rule JSP Tag Library

This chapter describes the JSP tags contained in **RuleTag.jar**:

- ◆ **doAction**
- ◆ **doCondition**
- ◆ **conditionalRule**
- ◆ **fireRule**

 For background information, see the chapter on [using the exteNd Director tag libraries](#) in *Developing exteNd Director Applications*.

doAction

Description

Fires an exteNd Director action and returns the response phrase.

This tag wraps the doAction() method on the EbiAction interface and the getResponsePhrase() method on the EbiContext interface.

Syntax

```
<prefix:doAction action="action" id="ID" />
```

Attribute	Required?	Request-time expression values supported?	Description
action	Yes	No	Specifies the class name for the action that will be fired.
id	No	No	Specifies the name of the variable that will be used to store the response phrase. If no value is specified, the response phrase is inserted in the page at the location where the tag appears.

Examples

This example shows how to use the doAction tag to insert an action's response phrase at the tag location:

```
<% taglib uri="/re" prefix="re" %>
...
Result of Action is...
<re:doAction action="com.sssw.re.action.ReturnAsHtmlBold" />
```

This example shows how to use the doAction tag to store an action's response phrase in a variable for further processing:

```
<% taglib uri="/re" prefix="re" %>
...
<re:doAction action="com.sssw.re.action.ReturnAsHtmlBold"
id="results"/>
...
The response phrase is: <%=pageContext.getAttribute("results")%>
```


doCondition

Description Fires an exteNd Director condition and returns a string that represents the boolean value returned by the condition.

This tag wraps the doCondition() method on the EbiCondition interface.

Syntax

```
<prefix:doCondition condition="condition" id="ID" />
```

Attribute	Required?	Request-time expression values supported?	Description
condition	Yes	No	Specifies the class name for the condition that will be fired.
id	No	No	Specifies the name of the variable that will be used to store the result of the condition. If no value is specified, the result is inserted in the page at the location where the tag appears.

Examples

This example shows how to use the doCondition tag to insert a condition’s result at the tag location:

```
<% taglib uri="/re" prefix="re" %>
...
Result of condition is ...
<re:doCondition condition="com.sssw.re.condition.CheckDate" />
```

This example shows how to use the doCondition tag to store an action’s response phrase in a variable for further processing:

```
<% taglib uri="/re" prefix="re" %>
...
<re:doCondition condition="com.sssw.re.condition.CheckDate"
id="result"/>
...
The result is: <%=pageContext.getAttribute("result")%>
```

conditionalRule

Description Fires an exteNd Director rule that returns a true or false result.
This tag wraps the `isTrue()` method on the `EbiRuleManager` interface.

Syntax `<prefix:conditionalRule rule="rule" owner="owner" id="ID" />`

Attribute	Required?	Request-time expression values supported?	Description
rule	Yes	No	Specifies the ID for the rule that will be fired.
owner	No	No	Specifies the owner for the rule.
id	No	No	Specifies the name of the variable that will be used to store the result returned by the rule. If no value is specified, a default id of ruleResult is used.

Example

```
<% taglib uri="/re" prefix="re" %>
...
<re:conditionalRule rule="sample.bonus" id="rule1"/></p>
<%=pageContext.getAttribute("rule1")%>
```

fireRule

Description

Fires an exteNd Director rule and returns the response phrase.

This tag wraps the fireRule() method on the EbiRuleManager interface and the getResponsePhrase() method on the EbiContext interface.

Syntax

```
<prefix:fireRule rule="rule" owner="owner" id="ID" />
```

Attribute	Required?	Request-time expression values supported?	Description
rule	Yes	Yes	Specifies the ID for the rule that will be fired.
owner	No	No	Specifies the owner for the rule.
id	No	No	Specifies the name of the variable that will be used to store the response phrase. If no value is specified, the response phrase is inserted in the page at the location where the tag appears.

Examples

This example shows how to use the fireRule tag to insert a rule's response phrase at the tag location:

```
<% taglib uri="/re" prefix="re" %>
...
<re:fireRule rule="myrule" />
```

This example shows how to use the fireRule tag to store a rule's response phrase in a variable for further processing:

```
<% taglib uri="/re" prefix="re" %>
...
<re:fireRule rule="myrule" id="results"/>
...
The response phrase is: <%=pageContext.getAttribute("results")%>
```


Index

A

- action macros
 - using in the Rule Editor 70
- actions
 - accessing sources 89
 - adding cases in the Rule Editor 63
 - deactivating in the Rule Editor 64
 - defining logic for 43
 - designing custom 40, 41
 - installed, reference documentation for 89
 - installed, working with 25
 - using in the Rule Editor 62
 - writing custom 39
 - see also custom conditions and actions
- Action Wizard 73
- Add
 - installed action 93
- Add Eraser
 - installed action 94

B

- BeanInfo class
 - writing for custom conditions and actions 51
- bindings
 - setting up for pipelines 85

C

- Calculate Age
 - installed action 94
- cases
 - in the Rule Editor 65
- Check Component Parameter
 - installed condition 126
- Check Date
 - installed condition 126
- Check Date Within Range
 - installed condition 127
- Check Day
 - installed condition 127
- Check For Cookie
 - installed condition 128
- Check Month
 - installed condition 128
- Check Request Data
 - installed condition 128
- Check Time
 - installed condition 128
- Check User
 - installed condition 129
- Check User Group
 - installed condition 129
- Check User Property
 - installed condition 130
- Check Whiteboard
 - installed condition 130
- Check Whiteboard Value
 - installed condition 131
- Check Whiteboard Value Is Empty
 - installed condition 131
- Check Workitem Value
 - installed condition 131
- Clear Request Data From Whiteboard
 - installed action 95
- conditionalRule tag 138
- condition macros
 - using in the Rule Editor 68
- conditions
 - accessing sources 89
 - adding in the Rule Editor 60
 - deactivating in the Rule Editor 62
 - designing custom 40, 41
 - installed, reference documentation for 125
 - installed, working with 25
 - using in the Rule Editor 59
 - writing custom 39
 - see also custom conditions and actions
- Condition Wizard 73
- context methods
 - for accessing the whiteboard 33
- context object
 - firing rules from 29
- cookies
 - using in rules 25

- Create Collection Of Objects From SQL
 - installed action 95
- custom conditions and actions
 - BeanInfo classes 51
 - choosing a user interface 40
 - compiling 80
 - designing 41
 - properties 41, 45
 - resource bundles 53
 - rule descriptors 44
 - templates 46
- custom tags
 - conditionalRule 138
 - doAction 136
 - doCondition 137
 - fireRule 139

D

- database driver
 - property in conditions and actions 90
- database name
 - property in conditions 90
- database URL
 - property in conditions and actions 90
- data types
 - in custom conditions and actions 49
- decision mode
 - in rules 18
- Default
 - installed action 96
 - installed condition 132
- Delete Cookie
 - installed action 96
- Deny Access
 - installed action 97
- Display Component
 - installed action 97
- Display Cookies
 - installed action 97
- Display Request Headers
 - installed action 97
- Display Whiteboard
 - installed action 98
- Divide
 - installed action 98
- doAction() 78
- doAction() method 41, 49, 78
- doAction() method (code example) 44

- doAction tag 136
- doCondition() 78
- doCondition() method 41
- doCondition() method (code example) 42
- doCondition tag 137
- Drop Cookie User ID
 - installed action 98

E

- EbiContext
 - rules and 24
- EbiRuleManager
 - defined 24

F

- Fire Rule
 - installed action 99
- fireRule() method 29
- fireRule tag 139
- fireTemporaryRule() method 29
- firing rules
 - methods for 29
- Flush
 - installed action 99
- Format Date
 - installed action 99

G

- getClazz() method
 - custom conditions and actions and 52
- Get Cookie Value
 - installed action 100
- getLog() method
 - and custom conditions and actions 77
- getParameterPanel() method
 - and custom conditions and actions 77
- getResponsePhrase() method 32
- getResponseStatus() method 32
- getResponsePhrase() method 32
- getTemporaryValue() method 33
- Get User Property
 - installed action 100
- getValue() method 33, 47, 49
- getValueNames() method 33
- group binding, in pipelines 85

H

hasValue() method 33

I

isFalse() method 29
Is Form Get Data Available
 installed condition 132
isTrue() method 29

J

JavaBeans
 defining for custom conditions and actions 45
JComboBox
 in custom conditions and actions 51
JDBC Driver
 property in conditions and actions 90
JPanel
 in custom conditions and actions (code example) 50
JSP pages
 firing rules from 30

L

Log User Off
 installed action 100

M

merge() method 33, 47
Multiply
 installed action 100

N

New Session
 installed condition 132
now
 built-in whiteboard key 26

O

object attributes
 accessing from whiteboard 27
owners for rules 30

P

pipelines
 benefits of 36
 developing 35
 editing 84
 how they work 36
 pipeline binding, setting up 85
 setting up, in the Pipeline Editor 81
 validating 37
properties
 in custom conditions and actions 41, 45
PropertyDescriptor code element 52
property panels
 creating in custom conditions and actions 50
 designing for conditions and actions 40
 generic panels in custom conditions and actions 49

Q

Query
 installed action 101

R

Remove From Whiteboard
 installed action 101
removeValue() 33
resource bundles
 BeanInfo class (code example) 53
 in custom conditions and actions 51, 53
response
 built-in whiteboard key 26
response phrase
 in rules 25
response status
 in rules 25
Return As Decimal Format
 installed action 101
Return As Html Body
 installed action 102

- Return As Html Bold
 - installed action 103
- Return As Html Break
 - installed action 103
- Return As Html Checkbox
 - installed action 104
- Return As Html File Upload
 - installed action 104
- Return As Html Hidden Field
 - installed action 105
- Return As Html JavaScript
 - installed action 106
- Return As Html Option List
 - installed action 106
- Return As Html Password
 - installed action 107
- Return As Html Radio Button
 - installed action 108
- Return As Html Reset Button
 - installed action 109
- Return As Html Scripted Button
 - installed action 109
- Return As Html Submit Button
 - installed action 110
- Return As Html Table
 - installed action 111
- Return As Html Text Area
 - installed action 112
- Return As Html Text Field
 - installed action 112
- Return As XML
 - installed action 113
- Return Authentication Required
 - installed action 115
- Return False
 - installed action 115
- Return Response
 - installed action 115
- Return Response With Default
 - installed action 115
- Return True
 - installed action 116
- return values
 - handling in rules (code example) 33
- rule binding
 - creating 85
 - editing 86
- ruleDesc
 - built-in whiteboard key 27
- rule descriptors
 - defining for a custom condition or action 44
- Rule Editor
 - about 19
 - accessing 57
 - caret, meaning of 26
 - macros defined 19
 - pipelines defined 19
- ruleID
 - built-in whiteboard key 27
- rule manager
 - about 30
 - firing rules from 30
- rules
 - accessing and firing 28
 - and pipelines, developing 23
 - binding to a user, group, or pipeline 85
 - boolean results 33
 - cookies 25
 - design guidelines 21
 - editing 67
 - firing from a JSP page 30
 - firing from a rule manager 30
 - firing from the context object 29
 - getting manager (code example) 30
 - handling return values (code example) 33
 - handling the results of 32
 - how they work 23
 - in JSP pages 29
 - methods for firing 29
 - naming in Rule Editor 59
 - response phrase 25
 - response status, setting 25
 - returning data (code example) 34
 - running in the Rule Editor 67
 - saving 67
 - structure 17
 - temporary 31
 - testing 67
 - understanding 17
 - using a whiteboard value (code example) 35
 - !valueOf syntax 26
- Rule subsystem
 - about 17
- Run Rule command
 - in the Rule Editor 67
- runtime properties
 - in conditions and actions 46

S

Save Cookies To Whiteboard
 installed action 116
 installed condition 132

Save Form Get Data To Whiteboard
 installed action 116
 installed condition 132

Save Request Data To Whiteboard
 installed action 116
 installed condition 133

Save To Whiteboard
 installed action 117

scoped paths
 rules, using in 28, 90

Send Mailer SMTP
 installed action 117

Set Action Off
 installed condition 133

Set Action On
 installed condition 133

Set Action On Or Off
 installed condition 133

Set Component Parameter
 installed action 118

Set Cookie Value
 installed action 118

Set Date On Whiteboard
 installed action 119

Set Expired
 installed action 119

Set Next Activity 120

Set Pipeline Status
 installed action 120

Set Response Header
 installed action 120

setResponsePhrase() method
 about 32
 in custom action 43

Set Response Status
 installed action 121

setResponseStatus() method
 about 32
 in custom action 43

setResponseType() method
 about 32
 in custom action 43

setTemporaryValue() method 33

Set User Property
 installed action 121

setValue() method 33, 49

setValueNames() method 33

Set Workitem Priority
 installed action 121

Set Workitem Value
 installed action 122

SQL Check For Column
 installed condition 133

SQL Hierarchy
 installed action 122

SQL String
 installed action 122
 installed condition 134

Stop Rule Processing
 installed action 123

string templates
 using in custom conditions and actions (code example) 47

Subtract
 installed action 123

T

tag libraries
 Rules tag library 135

templates
 using in custom conditions and actions 46
 using in custom conditions and actions (code example) 47

today
 built-in whiteboard key 26

toString() method
 about 78
 in conditions and actions 41
 in custom conditions and actions (code example) 44

U

uri
 built-in whiteboard key 27

user binding
 in pipelines 85

userID
 built-in whiteboard key 26

V

validate() method 37
!valueOf construct 26, 46

W

whiteboard
 and object attributes 27
 built-in values 26
 context methods for accessing 33
 defined 24
 removing values from 28
 using in rules 25
 !valueOf syntax 26
 values, accessing 28
 values, in rules 26

X

XML
 and temporary rules 31