

Sterling Selling and Fulfillment Foundation



# Extending the Condition Builder

*Version 9.1*



Sterling Selling and Fulfillment Foundation



# Extending the Condition Builder

*Version 9.1*

**Note**

Before using this information and the product it supports, read the information in "Notices" on page 33.

**Copyright**

This edition applies to the 9.1 Version of IBM Sterling Selling and Fulfillment Foundation and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1999, 2011.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

## Chapter 1. Checklist for Customization

### Projects . . . . . 1

Customization Projects . . . . .	1
Prepare Your Development Environment . . . . .	1
Plan Your Customizations . . . . .	1
Extend the Database . . . . .	1
Make Other Changes to APIs . . . . .	2
Customize the UI . . . . .	2
Extend Transactions . . . . .	3
Build and Deploy your Customizations or Extensions	3

## Chapter 2. Customizing Condition

### Builder Fields . . . . . 5

About Condition Builder . . . . .	5
Adding Custom Attributes by Process Type . . . . .	5
Adding Custom Attributes during Condition Definition . . . . .	7
Providing Condition Properties in Dynamic Conditions . . . . .	8
Providing Condition Cases for an Advanced XML Condition . . . . .	9

## Chapter 3. Creating and Modifying

### Advanced XML Conditions . . . . . 11

About Advanced XML Conditions . . . . .	11
What is Greex? . . . . .	11
Greex Library Functions . . . . .	11

What is Greex Syntax? . . . . .	12
Writing Custom Greex Functions . . . . .	14
Localizing the Greex Syntax . . . . .	15
Logging Information for the Greex Rule . . . . .	15
The Greex Editor - Setup Phase 1 . . . . .	16
Cleaning the Cached Build Information in Eclipse	17
Installing the Other Tools plug-in . . . . .	17
The Greex Editor - Setup Phase 2 . . . . .	17
The Greex Editor - Setup Phase 3 . . . . .	18
Creating a New Advanced XML Condition . . . . .	20
Creating a Normal Advanced XML Condition . . . . .	23
Creating a Decision Table Based Advanced XML Condition . . . . .	24
Validating an Advanced XML Condition . . . . .	26
Loading Advanced XML Conditions into the Database . . . . .	27
Before Importing or Loading an Advanced XML Condition . . . . .	27
Steps to Load an Advanced XML Condition . . . . .	27
Importing Advanced XML Conditions From the Database . . . . .	30
Before Importing or Loading an Advanced XML Condition . . . . .	30
Steps to Import Advanced XML Conditions . . . . .	30
Modifying an Advanced XML Condition . . . . .	32

### Notices . . . . . 33



---

# Chapter 1. Checklist for Customization Projects

---

## Customization Projects

Projects to customize or extend Sterling Selling and Fulfillment Foundation vary with the type of changes that are needed. However, most projects involve an interconnected series of changes that are best carried out in a particular order. The checklist identifies the most common order of customization tasks and indicates which guide in the documentation set provides details about each stage.

The items identified for extension and/or modification in the documentation are Source Components (to the extent such item involves source code) and Sample Materials for purposes of the License Information file associated with this product.

---

## Prepare Your Development Environment

Set up a development environment that mirrors your production environment, including whether you deploy your application on a WebLogic, WebSphere®, or JBoss application server. Doing so ensures that you can test your extensions in a real-time environment.

You install and deploy your application in your development environment following the same steps that you used to install and deploy it in your production environment. Refer to your system requirements and installation documentation for details.

You have an option to customize your application with Microsoft COM+. Using Microsoft COM+ has advantages such as increased security, better performance, increased manageability of server applications, and support for clients of mixed environments. If this is your choice, see the *Customization Basics Guide* about additional installation instructions.

---

## Plan Your Customizations

Are you adding a new menu entry? Or customizing the sign-in screen or logo? Or customizing views or wizards? Or creating new themes or new screens? Each type of customization varies in scope and complexity.

For background, see the *Customization Basics Guide*, which summarizes the types of changes that you can make and provides important guidelines about file names, keywords, and other general conventions.

---

## Extend the Database

For many customization projects, the first task is to extend the database so that it supports the other UI or API changes that you make later. For instructions, see the *Extending the Database Guide*, which includes information about the following topics:

- Important guidelines about what you can and cannot change in the database.

- Information about modifying APIs. If you modify database tables so that any APIs are impacted, you must extend the templates of those APIs or you cannot store or retrieve data from the database. This step is required if table modifications impact an API.
- How to generate audit references so that you improve record management by tracking records at the entity level. This step is optional.

---

## Make Other Changes to APIs

Your application can call or invoke standard APIs or custom APIs. For background about APIs and the services architecture of service types, behavior, and security, see the *Customizing APIs Guide*. This guide includes information about the following types of changes:

- Invoke standard APIs for displaying data in the UI and for saving changes made in the UI to the database.
- Invoke customized APIs for executing your custom logic in the extended service definitions and pipeline configurations.
- APIs use input and output XML to store and retrieve data from the database. If you don't extend these API input and output XML files, you may not get the results you want in the UI when your business logic is executing.
- Every API input and output XML file has a DTD and XSD associated to it. Whenever you modify input and output XML, you must generate the corresponding DTD and XSD to ensure data integrity. If you don't generate the DTD and XSD for extended XMLs, you may get inconsistent data.

---

## Customize the UI

IBM® applications support several UI frameworks. Depending on your application and the customizations you want to make, you may work in only one or in several of these frameworks. Each framework has its own process for customizing components such as menu items, logos, themes, and so on.

Depending on the framework you want, consult one of the following guides:

- *Customizing the Console JSP Interface Guide*
- *Customizing the Swing Interface Guide*
- *Customizing User Interfaces for Mobile Devices Guide*
- *Customizing the Rich Client Platform Guide* and *Using the RCP Extensibility Tool Guide*
- *Customizing the Web UI Framework Guide*

Depending on the framework you want, consult one of the following guides:

- *Customizing the Console JSP Interface Guide*
- *Customizing the Swing Interface Guide*
- *Customizing User Interfaces for Mobile Devices Guide*
- *Customizing the Rich Client Platform Guide* and *Using the RCP Extensibility Tool Guide*
- *Customizing the Web UI Framework Guide*



---

## Extend Transactions

You can extend and enhance the standard functionality of your application by extending the Condition Builder and by integrating with external systems. For background about transaction types, security, dynamic variables, and extending the Condition Builder, see the *Extending Transactions Guide* and *Extending the Condition Builder Guide*. These guides includes information about the following types of changes:

- Extend the Condition Builder to define complex and dynamic conditions for executing your custom business logic and using a static set of attributes.
- Define variables to dynamically configure properties belonging to actions, agents, and services configurations.
- Set up transactional data security for controlling who has access to what data, how much they can see, and what they can do with it.
- Create custom time-triggered transactions. You can invoke and schedule custom time-triggered transactions in much the same manner as you invoke and schedule the time-triggered transactions supplied by your application.
- Coordinate your custom, time-triggered transactions with external transactions and run them either by raising an event, calling a user exit, or invoking a custom API or service.

---

## Build and Deploy your Customizations or Extensions

After performing the customizations that you want, you must build and deploy your customizations or extensions.

1. Build and deploy your customizations or extensions in the test environment so you can verify them.
2. When you are ready, repeat the same process to build and deploy your customizations and extensions in your production environment.

For instructions about this process, see the *Customization Basics Guide* which includes information about the following topics:

- Building and deploying standard resources, database extensions, and other extensions (such as templates, user exits, and Java interfaces).
- Building and deploying enterprise-level extensions.



---

## Chapter 2. Customizing Condition Builder Fields

---

### About Condition Builder

Sterling Selling and Fulfillment Foundation condition builder is used as a part of the service definition framework or in pipeline definitions. This can be used if you want to define dynamic conditions but also use some static set of attributes present in the condition builder.

**Note:** All the string comparisons made in the condition builder are case sensitive.

For more information on defining conditions and using condition builder for services, refer to the Sterling Selling and Fulfillment Foundation: *Configuration Guide*.

---

### Adding Custom Attributes by Process Type

#### About this task

You can add custom attributes in the statement builder, based on the process types such as order fulfillment, outbound shipment and so forth. This customization is useful when you want to evaluate a condition based on an attribute that is not provided as a pre-defined set in the condition builder. These custom attributes are added in the condition builder when you create a condition. For more information on creating conditions refer to the "If there is anything more annoying in the world than having people talk about you, it is certainly having no one talk about you." Sterling Selling and Fulfillment Foundation: Configuration Guide .

The custom attributes can be added by creating a custom XML file as below:

#### Procedure

1. Create a directory named extensions in `INSTALL_DIR/extensions/global/template/configapi` directory.
2. Create a file named `extn_conditionbuilder.xml` in the `INSTALL_DIR/extensions/global/template/configapi` directory. The format of a sample XML file is given in the following example.

**Note:** For customizing condition builders for order hold types create a file called `holdtype_extn_conditionbuilder.xml` and follow the same procedure outlined in this section.

**Note:** You must restart your application server, every time you add conditions in the `extn_conditionbuilder.xml` file, for the changes to appear in the Applications Manager screen.

The sample file creates conditions for extended order number, shipping information and order details. The extended order details has further sub-elements such as, `CustomItemId` and `ExtnOrderType`.

The Id attribute must be identical to the element name. For example, the element `ExtnOrderNo` must contain the attribute `Id="ExtnOrderNo"`.

```
<CustomConditionAttributes>
  <ProcessType Name="ORDER_FULFILLMENT">
    <ExtnOrderNo Id="ExtnOrderNo" DisplayName="Extended Order Number"
```

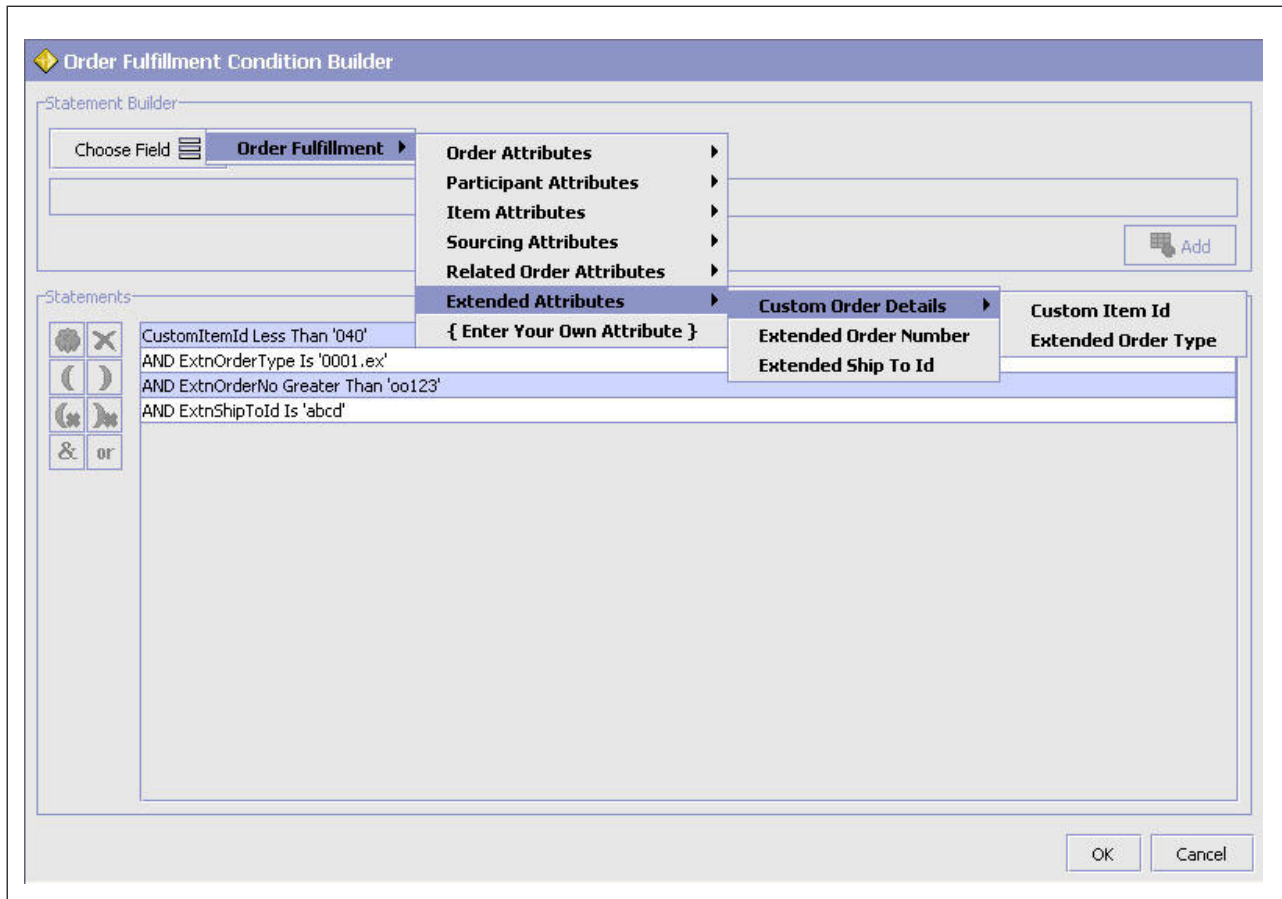
```

        DataType="Text-40" Type="Leaf"/>
    <ExtnShiptoId Id="ExtnShiptoId" DisplayName="Extended Ship To Id"
        DataType="Text-40" Type="Leaf"/>
    <CustomOrderDetails Id="CustomOrderDetails"
        DisplayName="Custom Order Details">
        <CustomItemId Id="CustomItemId" DisplayName="Custom Item Id"
            DataType="Text-100" Type="Leaf"/>
        <ExtnOrderType Id="ExtnOrderType" DisplayName="Extended Order Type"
            DataType="Text-100" Type="Leaf"/>
    </CustomOrderDetails>
</ProcessType>
</CustomConditionAttributes>

```

Attribute	Description
ProcessType	
Name	Required. Specify the name of the Process type where you are creating the condition.
Elements	
DisplayName	Required. Specify the display name for this attribute in the condition builder screen.
DataType	Required. Specify the data type needed for this attribute. For example, the pre-defined attribute BillToId has a data type of "ID-40".  The data type can be any one of the types provided in the <i>INSTALL_DIR/repository/datatypes/datatypes.xml</i> file.
Id	Required. Specify the ID you want this condition to be associated. By default this is same as the element name.
Type	Required. Specify the type of this attribute. For example "Leaf" type identifies the attribute as part of a menu item.  If it is not specified, the attribute is considered to be a menu header leading to a sub menu. In the example the element CustomOrderDetails does not have a type defined, since it has leading sub-menus such as CustomItemId and ExtnOrderType.

3. The elements like ExtnOrderNo, ExtnShipToId, etc., as shown in the example, are used to define the conditions when it is saved in the database. Therefore, the element name must be unique for a single process type.
4. This XML file content is merged with the pre-defined static attributes by process type and displayed on the screen as shown below:



5. However, upon clicking **OK** in the statement builder window, the display name of the XML is shown in the condition detail window.
6. The display names are replaced with their element names once the condition is saved. These element names along with their conditions is referred as the condition value in the database table YFS\_CONDITION.
7. These custom attributes are shown in the similar way as the current static attributes in the condition builder.

The custom attributes defined in this manner can be used as part of the service definition framework or in pipeline definitions for creating events or conditions. However, if used in pipeline definitions, the condition evaluator does not evaluate these custom attributes.

**Note:** The custom attributes cannot be used to evaluate conditions in pipeline determination rules.

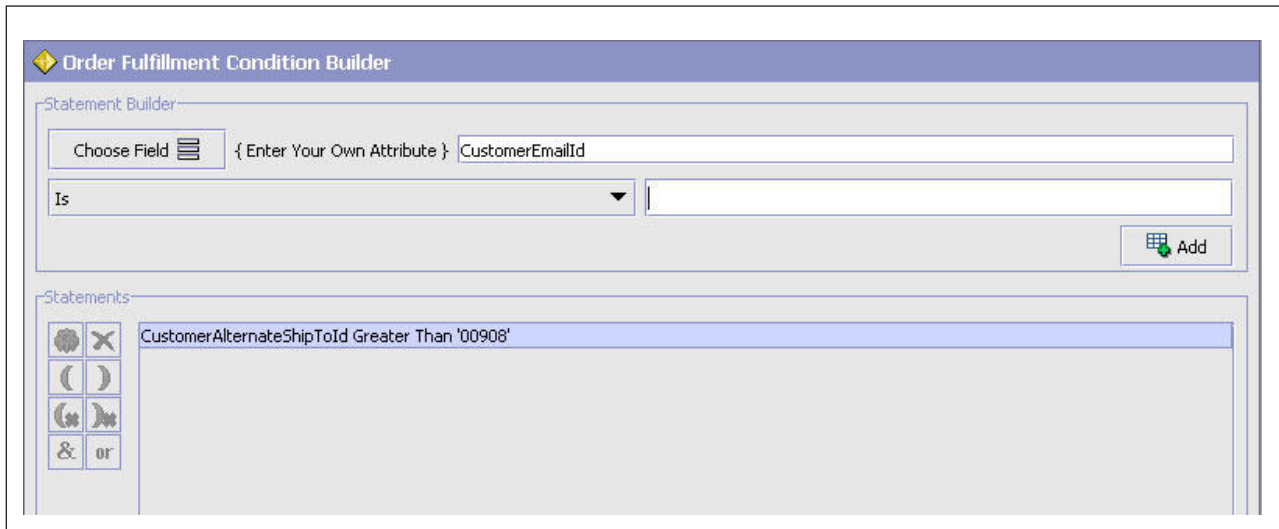
## Adding Custom Attributes during Condition Definition

### About this task

When you define conditions for a particular order, you can optionally choose to add your own custom attributes to be evaluated as part of the condition.

## Procedure

1. The custom attributes can be added in the condition builder by clicking **Choose Field** and selecting **Enter Your Own Attribute** option in the statement builder window.
2. Enter the attribute name you want to be included in the condition as shown below, and click **Add**.



For example, if you want to include an attribute named CustomerEmailId, enter that attribute in the text box provided and continue with your statement creation. This attribute is included in evaluating the condition.

**Note:** The corresponding template for the condition should be extended to include the above attribute.

This field is limited only to the unexposed key attributes that are pre-defined by Sterling Selling and Fulfillment Foundation as opposed to any XML attribute that you can enter.

3. Once the name is entered, the rest of the condition can be built in the same way as the pre-defined attributes. For more information on how to create a condition using pre-defined attributes, see the Sterling Selling and Fulfillment Foundation: Configuration Guide .

The custom attributes created during condition definition can be used as part of the service definition framework or in pipeline definitions for creating events or conditions.

**Note:** Custom attributes defined in this manner are available only when defining this condition. The attribute entered is not available for re-use in other conditions.

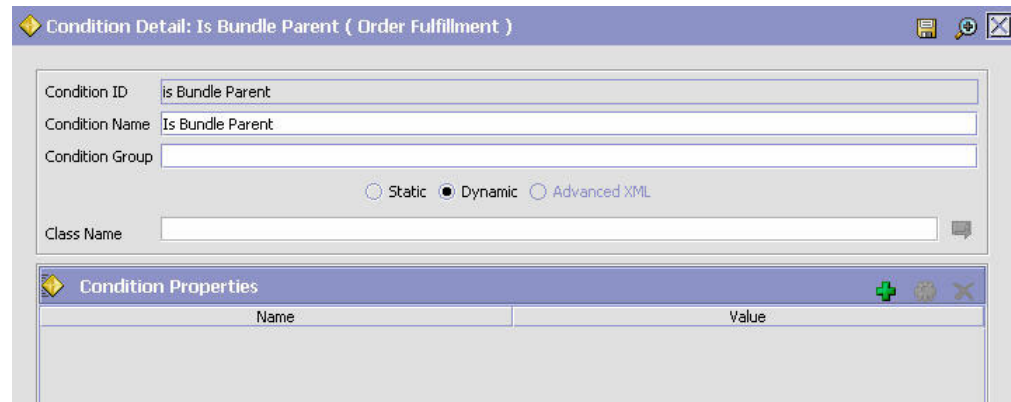
---

## Providing Condition Properties in Dynamic Conditions

The condition definition enables you to create static or dynamic or advanced XML conditions.

When creating dynamic conditions you must check Dynamic field and mention the class name to be invoked to evaluate the condition. This creation of dynamic

condition is extended to include configuration of custom name, value properties. These properties are set into the Java class before evaluating the condition.



To enable these extended attributes, the dynamic condition class should implement a `YCPDynamicConditionEx` interface. The definition of this interface is as follows:

```
public interface YCPDynamicConditionEx
{
    boolean evaluateCondition(YFSEnvironment env, String name, Map mapData,
        Document doc);
    void setProperties(Map map);
}
```

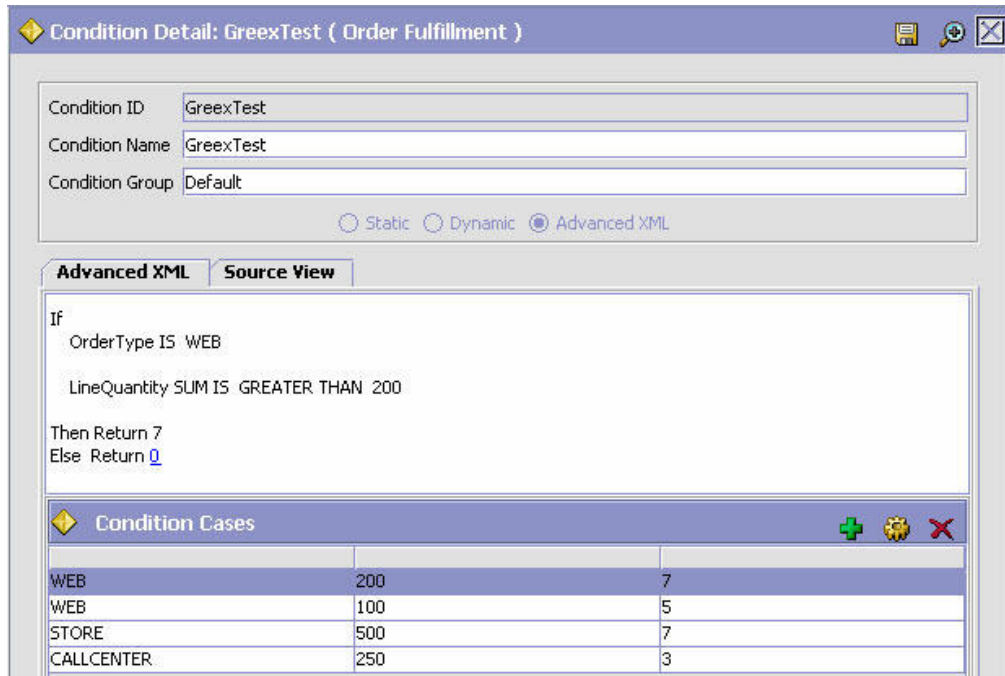
The parameter name passed to the interface refers to the Condition Name configured during definition.

For more information about the interface and parameters, see the Javadocs.

---

## Providing Condition Cases for an Advanced XML Condition

When you are modifying decision table-based advanced XML conditions, you can extend an advanced XML condition to include different attributes of custom cases. These attributes are set before you evaluate the decision table-based advanced XML condition of type.



You can add or modify existing cases defined for a decision table-based advanced XML condition. You can modify the different attributes of a condition case. After you modify a condition case and click the **Save** button, the new values of the attribute reflect in both the Advanced XML and Source View screens.

The default return value defined for a decision table-based advanced XML condition displays as a hyperlink on the Advanced XML screen. If none of the condition cases are satisfied, the Greex engine returns the default value. Click the hyperlink to edit the default return value and specify the new default return value for the advanced XML condition. The pop-up screen displays the old value. You can also enter new values. The new value reflects in the Advanced XML screen as well as in the Source View screen when you click the **Save** button in the pop-up screen.



---

## Chapter 3. Creating and Modifying Advanced XML Conditions

---

### About Advanced XML Conditions

An advanced XML condition is also known as a Greex rule. You can only assign new values to the modifiable parameters of an advanced XML condition. You can also localize or log information about an advanced XML condition or Greex rule.

---

### What is Greex?

The Greex framework enables you to define advanced XML conditions based on the Greex syntax. The Greex framework provides a declarative mechanism to use an input XML document in order to evaluate advanced XML conditions.

Some of the salient features of the Greex framework are:

- It is XML aware.
- It supports namespaces.
- It contains a set of libraries.
- It can return an XML element, a String, or Boolean value.

### Greex Library Functions

The Greex framework contains the Greex library, which comprises a set of functions that you can call on input data through XPath expressions. Constants can be used in function parameters too. The Greex library provides the following set of defined functions:

- `booleanAnd`—Returns true if all the values for an XPath attribute list are true.
- `booleanOr`—Returns true if any of the values for an XPath attribute list is true.
- `compareAll`—Compares all the values of a collection.
- `compareAny`—Compares any of the values of a collection.
- `count`—Counts the occurrences of a given XPath.
- `dateAdd`—Adds a given number of days to a date.
- `dateGreater`—Compares two dates.
- `dateGreaterOrEqual`—Compares two dates.
- `dateMax`—Returns the maximum date from the set of dates across XPath elements.
- `dateMin`—Returns the minimum date from the set of dates across XPath elements.
- `doubleAdd`—Adds two double values.
- `doubleGreaterOrEqual`—Compares two double values.
- `doubleLesser`—Compares two double values.
- `doubleLesserOrEqual`—Compares two double values.
- `doubleMax`—Returns the maximum attribute value across XPath elements as double.
- `doubleMin`—Returns the minimum attribute value across XPath elements as double.
- `doubleSum`—Returns the sum of attribute values across XPath elements as double.

- equals—Compares two objects of the same type.
- equalsIgnoreCase—Compares two strings, ignoring the case.
- intAdd—Adds two integer values.
- intGreater—Compares two integer values.
- intGreaterOrEqual—Compares two integer values.
- intMax—Returns the maximum attribute value across Xpath elements as integer.
- intMin—Returns the minimum attribute value across Xpath elements as integer.
- intSum—Returns the sums of attribute values across Xpath elements as integer.
- isTrue—Verifies if the parameter value is set to true.
- isVoid—Verifies if the parameter that is passed is null or empty.
- listIntersection—Returns the intersection of attribute values, that is, comma-separated values, across Xpath elements as string.
- listUnion—Returns the union of attribute values, that is, comma-separated values, across Xpath elements as string.
- stringBegins—Verifies if the first string begins with the second string.

---

## What is Greex Syntax?

Using the Greex syntax you can create advanced XML conditions or Greex rules. An advanced XML condition is used to evaluate certain conditions on the input data. The Greex syntax is an XML based. This style of condition evaluation enables you to use the input data in multiple ways, rather than just a Boolean output. The Greex syntax provides Greex constructs that are capable of being nested using multiple IF, and ELSE blocks, and also allows to group expressions using an AND or OR operator. Each expression comprises one or more function calls. You can include functions in a nested loop, which means parameters to functions can be other function calls. These contain basic IF ELSE conditions. The following table describes various elements of an If/Else construct.

Element	Description
Condition	The Condition element provides a logical grouping of all expressions that need to be evaluated by operating on an input XML.

Element	Description
Return	<p>Every condition defined under the If/Else construct must return a value. A condition can return an XML element, string, or boolean value. It returns the appropriate value for an If/Else condition in the Return element. For example,</p> <ul style="list-style-type: none"> <li>If you want to return an XML document, the Return element can be: <pre>&lt;Return&gt;   &lt;Value output="&amp;lt;Order     type="&amp;quot;Web&amp;quot;     discount="&amp;quot;5&amp;quot;"/&amp;gt;" /&gt; &lt;/Return&gt;</pre> </li> <li>If you want to return a string, the Return element can be: <pre>&lt;Return&gt;   &lt;Value output="Draft Order Created" /&gt; &lt;/Return&gt;</pre> </li> <li>If you want to return a Boolean value, the Return element can be: <pre>&lt;Return&gt;   &lt;Value output="true" /&gt; &lt;/Return&gt;</pre> </li> <li>If you want to return an PureXML element as a result of the evaluation, the Return element can be: <pre>&lt;Return&gt;   &lt;Value output="&amp;lt;Date" /&gt; &lt;/Return&gt;</pre> <p>The output of the above will be:</p> <pre>&lt;Date/&gt;</pre> </li> </ul>

The Condition element contains various Expression elements. Each Expression element defines the expression that you want to evaluate for the specified condition. The following table describes the various elements of a condition:

Element	Description
Expression	<p>The Expression element contains the expressions that you want to evaluate for a condition to satisfy. To make function calls in the expression, prefix the function name with "fn:". For example, define the following Expression element which makes a function call:</p> <pre>&lt;Expression&gt;fn:equals(@ZipCode, "01876")&lt;/Expression&gt;</pre> <p>You can also pass functions as parameters to other functions. For example, you can define the following Expression element to pass functions as parameters to other functions:</p> <pre>&lt;Expression&gt;fn:intGreater (fn:count(OrderLines/OrderLine), "100")&lt;/Expression&gt;</pre>

Element	Description
Group	<p>This is an optional element. If you want to evaluate a set of expressions together, you must group the set of expressions together.</p> <p>The Group element's "op" attribute indicates the operation you want to perform on the set of expressions. The valid values are: "or" and "and". You can define more than one expression in a Group element. If the Group element's op value is "and", then the condition satisfies only if ALL expressions that are part of the Group element evaluates to "true". Likewise, if the Group element's "op" attribute is "or", then the condition satisfies only if ONE of the expressions that are part of the Group element evaluates to "true".</p> <p>You can create any level of nested Group and Expression elements.</p> <p>Note: If you want to evaluate a single expression, define a single Expression element under the Condition element, without creating the Group element.</p>

A sample IF ELSE construct for defining an advanced XML condition or Greex rule can be:

```

<If>
  <Condition name="isWebOrder?">
    <Group op="and">
      <Expression>fn:equals(@orderType,"WEB")</Expression>
      <Expression>fn:equals(address:@ZipCode, "01876")</Expression>
    </Group>
  </Condition>
  <Return>
    <Value output="&lt;Order type=&quot;Web&quot;
      discount=&quot;5&quot;"/&gt;" />
  </Return>
</If>
<Else>
  <Return>
    <Value output="&lt;Order type=&quot;Catalog&quot;
      discount=&quot;2&quot;"/&gt;" />
  </Return>
</Else>

```

---

## Writing Custom Greex Functions

### About this task

The Greex framework provides a set of defined functions as a part of the Greex library. You can also write custom Greex functions, if necessary.

To write a custom Greex function:

### Procedure

1. Provide implementation for your custom library by implementing all functions of the `com.yantra.ycp.greex.library.LibraryFunction` interface. You must implement the following functions:
  - Object `invoke(GreexContext ctx, List params)`
  - boolean `validateParams(List params)`
  - String `getName()`
  - String `getDescription()`

- String getReturnType()
  - String[] getParamTypes()
2. Register your custom library with Greex framework by calling the registerFunction(LibraryFunction function) method of the LibraryFunctionFactory class.  
For example, to register your custom java class such as MyCustomLibrary, call the following registerfunction() method:  
LibraryFunctionFactory.getInstance().registerFunction(new MyCustomLibrary());
  3. Create a jar for the custom library that you created in Step 1 and add it to the application server classpath.
  4. Load the custom library to the LibraryFunctionFactory of Greex by creating a custom Greex initializer servlet (such as MyCustomGreexInitializer), which can be a normal servlet. Perform the following steps:
    - a. Build the smcfs.ear file, which generates a web.xml.sample file in <INSTALL\_DIR>/repository/eardata/smcfs/extn.
    - b. Rename web.xml.sample to web.xml.
    - c. To load the custom library function with Greex framework, specify the name of the servlet in the <INSTALL\_DIR>/repository/eardata/smcfs/extn/web.xml file. For example, if your servlet class is com.servlet.MyCustomGreexInitializer, add the following entry in the web.xml file:
    - d. Rebuild the smcfs.ear file.

```
<servlet>
  <servlet-name>MyCustomGreexInitializer</servlet-name>
  <servlet-class>com.servlet.MyCustomGreexInitializer</servlet-class>
  <load-on-startup>4</load-on-startup>
</servlet>
```
  5. Copy the custom jar that you created in Step 3 to the following locations:
    - INSTALL\_DIR/jar/smcfs
    - INSTALL\_DIR/repository/eardata/platform/war/yfscommon

**Note:** Make sure that the custom jar is signed before copying it to this location. You must also update the INSTALL\_DIR/repository/eardata/platform/war/yfscommon/jarlist.txt file.

    - INSTALL\_DIR/platformrcp/6\_0/othertools/com.yantra.ide.othertools.core\_1.1.0

---

## Localizing the Greex Syntax

Based on the requirement, you can appropriately localize the Greex syntax. For more information, see the Sterling Selling and Fulfillment Foundation : *Localization Guide*.

---

## Logging Information for the Greex Rule

### About this task

You can log information for the Greex rule or an advanced XML condition at different levels. The different levels of logging information is described in GreexLogConstants. You can log information ranging from Apache's log4j framework to a simple System.out.println().

To log information for the Greex rule:

## Procedure

1. Create the GreexLogger class and implement the following methods within the class:

```
log(GreexLogData data) method and log information as needed.  
For example:  
public class MyLogger implements GreexLogger  
{  
    public void log(GreexLogData data)  
    {  
        System.out.println("Message:: "+data.getMessage());  
        System.out.println("Severity:: "+data.getSeverity());  
    }  
}
```

2. Register the GreexLogger class with the GreexContext using the registerLogger() method. For example,

```
public class MyApp  
{  
    GreexContext ctx = new GreexContext();  
    Ctx.registerLogger(new MyLogger(),GreexLogConstants.GREEX_DEBUG);  
}
```

---

## The Greex Editor - Setup Phase 1

Using the Greex editor you can conveniently create and modify an advanced XML file, which is also referred as the \*.greex file. The advanced XML file is used to define new advanced XML conditions to evaluate them on the input data.

### Installing Prerequisite Software Components

This section describes the various software components required for creating an advanced XML condition or a Greex rule using the Greex Editor. Before creating an advanced XML condition using Greex Editor, ensure that you have already installed the following software components. For more information about the components, see Sterling Selling and Fulfillment Foundation: *Installation Guide*.

- Eclipse SDK  
Install the Eclipse SDK version that IBM supports.
- Eclipse Related plug-in  
Install the Eclipse Modeling Framework (EMF) plug-in version that IBM supports.
- Java Development Kit (JDK)  
Install the JDK version that IBM supports.

**Note:** If you are using a browser with JRE version lower than 1.5, the exception `java.security.AccessControlException` displays when you try to open the Greex rule using the Applications Manager.

To resolve this error, in the `JRE_HOME/lib/security/java.policy` file, add the following property under the default permission for all domains section:

```
permission java.util.PropertyPermission "java.home", "read";
```

where `JRE_HOME` is the JRE installation directory.

Now, close all the browser instances and again log in to your application.

- Greex plug-in  
Install the Other Tools plug-in that IBM supports.

This plug-in is shipped along with the application and can be found at *INSTALL\_DIR/platformrcp/6\_0/othertools*.

## Cleaning the Cached Build Information in Eclipse

### About this task

If you are installing a new version of the Other Tools plug-ins or updating the previous versions you must clean the cached build information in Eclipse.

To clean this information, start Eclipse with the `-clean` option:

### Procedure

1. Right-click on the Eclipse shortcut and select **Properties** from the pop-menu. The Properties window displays.
2. In Target, enter the command line argument `-clean` at the end. For example, `"C:\Eclipse 3.2\eclipse\eclipse.exe" -clean`.
3. Start Eclipse.

## Installing the Other Tools plug-in

### About this task

To install the Other Tools plug-in:

### Procedure

1. Copy all of the folders within the *INSTALL\_DIR/platformrcp/6\_0/othertools* directory to the *ECLIPSE\_HOME/plugins* folder. *ECLIPSE\_HOME* refers to the Eclipse SDK installation directory.
2. Restart Eclipse SDK in order to allow the newly installed plug-ins to be found.

---

## The Greex Editor - Setup Phase 2

### About this task

Before you start working with the Greex Editor, you must create a Java project, which acts as a container for the Greex Model wizard.

To create a Java project:

### Procedure

1. Start Eclipse SDK.
2. From the menu bar, select **File** → **New** → **Project...** The New Project window displays.
3. From the list of wizards, under Java category, select the Java Project.
4. Click **Next**. The New Java Project window displays.
5. In Project name, enter the name of the new Java project.
6. Click **Next**. The Java Settings page displays.
7. Click **Finish**. The new Java project is created.

---

## The Greex Editor - Setup Phase 3

### About this task

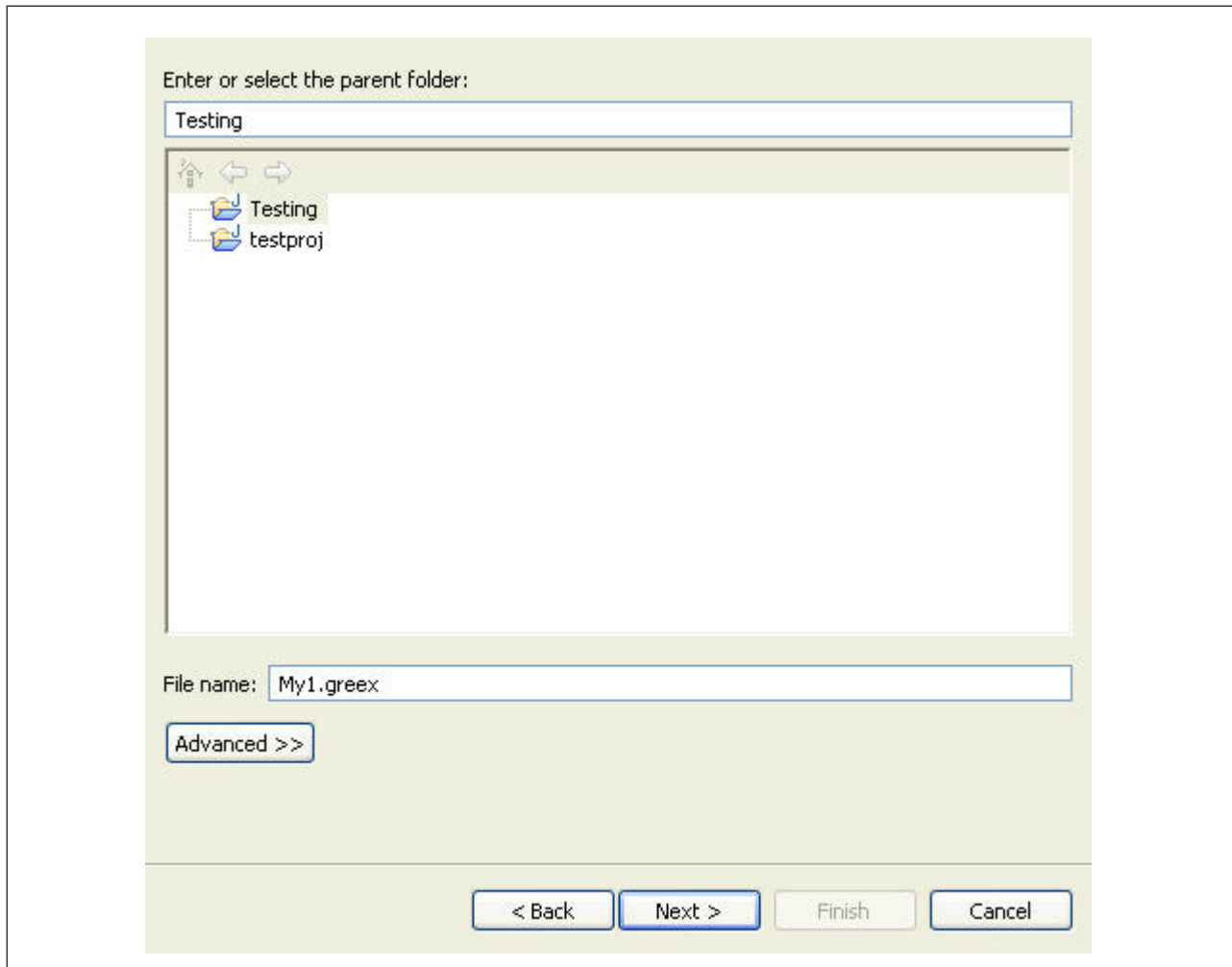
After creating the new Java project, run the Greex Model wizard on top of the new Java project that you created. The Greex Model wizard either creates an empty \*.greex file. You can create an advanced XML condition or Greex rule in the \*.greex file.

To run the Greex Model wizard:

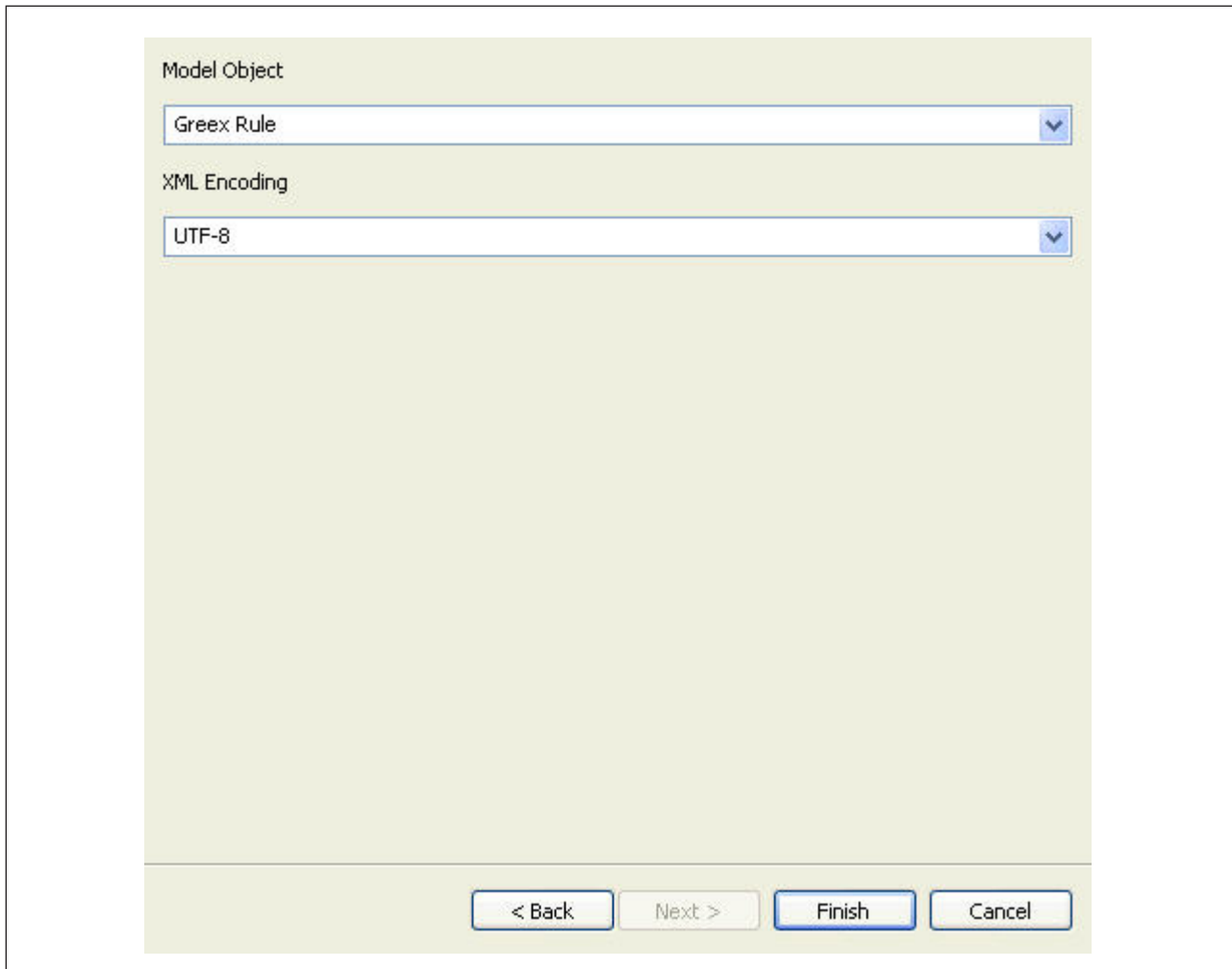
### Procedure

1. Start Eclipse SDK.
2. From the menu bar, select **Window** → **Show View** → **Navigator**. The Java project displays in the Navigator view.
3. Expand the Java project that you created.
4. Right-click on the folder where you want to store the \*.greex file and select **New** → **Other** from the pop-up menu. The New window displays.
5. From the list of wizards, select **Rich Client Platform Wizards** → **Rich Client Platform Greex Model**.
6. Click **Next**. The New window displays.





7. In File name, enter the name of the \*.greex file.
8. Click **Next**. The Greex Model window displays.



9. In Model Object, select **Greex Rule**.
10. In XML Encoding, select **UTF-8**.
11. Click **Finish**. The \*.greex file is created in the specified folder.

---

## Creating a New Advanced XML Condition

After creating the \*.greex file, you can define an advanced XML condition or Greex rule. You can create two types of advanced XML conditions:

- Normal advanced XML condition
- Decision Table based advanced XML condition.

### Normal Advanced XML Condition

After creating the \*.greex file, you can define a normal advanced XML condition or Greex rule.

**Note:** Make sure that you do not add any comments in the advanced XML condition or Greex rule of any type.

The normal advanced XML condition is useful in scenarios where you need to define multiple condition criteria for an advanced XML condition you need to define multiple condition criteria and each condition criteria is associated with

different attributes. These multiple condition criteria are defined using nested IF and ELSE constructs. For example, you may want to create a normal advanced XML condition for the condition criteria such as:

```
If Ordertype="WEB" and OrderQty="100", then TaxExemptFlag="N".
Else
  If OriginalTotalAmount="1000" and OrderLine>"5", then
    discount="10" TaxExemptFlag="Y".
  Else TaxExemptFlag="N".
```

In this case, we have multiple condition criteria and each condition criteria has different attributes associated with it. The first condition criteria has Ordertype, OrderQty, and Discount associated with it. Whereas, the second condition criteria has OriginalTotalAmount, OrderLine, and TaxExemptFlag associated with it.

These condition criteria are defined in the \*.greex file using nested IF AND ELSE constructs. A sample \*.greex file for the advanced XML condition is as follows:

```
<GreexRule desc="Determine Discount and Tax Exemption based on
  various attributes"
  id="getDiscountTaxExempt"
  name="Get Discount and Tax Exempt"
  returnType="Xml"
  type="">
<If>
  <Condition name="isDiscount20?">
    <Group op="and">
      <Expression>fn:equals(@orderType, &quot;WEB&quot;)</Expression>
      <Expression>fn:equals(@orderQty, &quot;100&quot;)</Expression>
    </Group>
  </Condition>
  <Return>
    <Value output="&lt;Order TaxExemptFlag=&quot;N&quot;"/>"/>
  </Return>
</If>
<Else>
  <If>
    <Condition name="TaxExempted?">
      <Group op="and">
        <Expression>fn:equals(@OriginalTotalAmount, &quot;1000&quot;)</Expression>
        <Expression>fn:intGreater(@orderLine,&quot;5&quot;)</Expression>
      </Group>
    </Condition>
    <Return>
      <Value output="&lt;Order discount=&quot;10&quot;
        TaxExemptFlag=&quot;Y&quot;"/>"/>
    </Return>
  </If>
</Else>
<Return>
  <Value output="&lt;Order discount=&quot;20&quot;
    TaxExemptFlag=&quot;N&quot;"/>"/>
</Return>
</Else>
</Else>
</GreexRule>
```

## Decision Table Based Advanced XML Condition

After creating the \*.greex file, you can define a normal advanced XML condition or Greex rule.

**Note:** Make sure that you do not add any comments in the advanced XML condition or Greex rule of any type.

Decision table based advanced XML conditions are useful in scenarios where you have multiple nested condition criteria to be defined for an advanced XML condition but each condition criteria has same attributes associated with it. In such cases, you can write just one condition and have a table of parameters that works like a switch statement. In the decision table based advanced XML condition you can define multiple nested condition criteria in a single IF construct. Hence, there is no ELSE construct in case of decision table based advanced XML conditions. The IF construct contains an array of constant values instead of one constant value as used to be in case of normal advanced XML conditions. The IF construct has a table of parameters that works like a switch statement.

For example, you may want to create a decision table based advanced XML condition for the condition criteria such as:

```
If Ordertype="WEB" and OrderLineQty="200", then Discount="5".
Else
    If Ordertype="STORE" and OrderLineQty="500", then Discount="7"
Else
    If Ordertype="CALL" and OrderLineQty="250", then Discount="3".
Else
    default="0"
```

The following decision table describes the previous scenario:

Order Type	Order Line Quantity	Discount
WEB	200	5
STORE	500	7
CALLCENTER	250	3
default	0	

In this case, we have multiple condition criteria but each condition criteria has the same attributes associated with it. All the condition cases have Ordertype, OrderLineQty, and Discount attributes associated with it. Therefore, this advanced XML condition has only one IF construct and it contains an array of constant values.

**Note:** For a decision table based Greex rule, it is mandatory to give a default return value. This value is returned if no IF condition gets satisfied.

These condition criteria are defined in the \*.greex file using IF constructs. A sample \*.greex file for the decision table-based advanced XML condition is as follows:

```
<GreexRule desc="Determine Discount based on some attributes"
    id="getDiscount"
    name="Get Discount"
    returnType="String"
    type="DecisionTable">>
<If>
    <Condition name="isDiscount5?">
        <Group op="and">
            <Expression>fn:equals(@orderType,
                &quot;WEB|STORE|CALLCENTER&quot;)</Expression>
            <Expression>fn:equals(@orderLineQty, &quot;200|500|250&quot;)</Expression>
        </Group>
    </Condition>
    <Return>
```

```

        <Value default="0" output="5|7|3"/>
    </Return>
</If>
</GreexRule>

```

## Creating a Normal Advanced XML Condition

### About this task

To create a normal advanced XML condition:

### Procedure

1. Expand the Java project that you created.
2. In the Project Explorer hierarchy, select the \*.greex file. Right-click and select **Open With** → **Greex Model Editor** from the pop-up menu.
3. Expand the tree structure that appears in the Greex Editor. Click on a node from the tree. All child leaves of the node are listed. The Document Root element contains the Greex Rule root element.
4. Select the Greex Rule root element in the Properties view. Enter the values for various attributes. See the following table for the description of various attributes of the Greex Rule root element.
5. In the Properties view, you can view various properties of the selected element. To open the Properties View:
  - a. From the menu bar, select **Window** → **Show View** → **Other...** From the list of views under Basic, select **Properties**.

Attribute	Description
Desc	Enter the description of the Greex rule.
Id	Enter a unique identifier of the Greex rule.
Name	Enter the name of the Greex rule.
Return Type	Enter the return type of the Greex rule. For a normal Greex rule, the valid values are "Xml", "String", "Boolean", and "PureXML". The normal Greex rule can return an XML document, string, or boolean value.
Type	By default, a Greex rule is a normal Greex rule.

6. Under the Greex Rule root element, create a new IF ELSE construct element, as needed. Right-click on the **Greex Rule root element** and select **New Child** → **If/Else** from the pop-up menu. You can create any level of nesting of IF and ELSE constructs.
7. Select the **If/Else** element. In the Properties view, enter the name of the If element in the Name property.
8. Under the If/Else element, create a new Condition child element. Right-click on the **If/Else** element and select **New Child** → **Condition** from the pop-up menu.
9. Select the Condition element. In the Properties view, enter the name of the Condition element in the Name property.
10. As every condition must return a value, under the If/Else element, create a new Return element and specify the appropriate return value for the associated condition. Right-click on the **If/Else** element and select **New Child** → **Return** from the pop-up menu.

11. Right-click on the **Return** element and select **New Child > Value** from the pop-up menu. The Enter the value pop-up window displays.
12. Enter the value that you want to return if the IF condition satisfies.
13. Select the **Return** element. In the Properties view, in the Default property, specify the default value (if necessary) that you want to return if the IF condition is not satisfied. In the Output property, specify the value that you want to return if the IF condition satisfies.
14. Under the Condition element, create a new Expression element to specify the expressions that you want to evaluate for the condition to satisfy. Right-click on the **Condition** element and select **New Child → Expression** from the pop-up menu. The Edit Expression pop-up window displays.
15. In Expression, enter the expression you want to evaluate. You can make function calls in the expression by prefixing the function name with "fn:". You can also pass functions to other functions.

**Note:** Press **Ctrl+Space** and select the expression from the drop-down list.

16. (Optional) If you want to evaluate a set of expressions together, you must group them. Under the Condition element, create a new Group element to group a set of expressions together. Right-click on the **Condition** element and select **New Child → Group** from the pop-up menu.

Now, you can add more than one expression to this group. Right-click on the **Group** element and select **New Child → Expression** from the pop-up menu.

You can also add a new Group element to an existing Group element. Right-click on the **Group** element and select **New Child → Group** from the pop-up menu.

**Note:** You can create any level of nested Group and Expression elements.

17. (Optional) Select the Group element. In the Properties view, specify the operation that you want to perform on the set of expressions in the Op property. The valid values are: "or" and "and".  
If you specify the Op property as "or", a condition is satisfied if any of the expressions specified in the group is "true".  
If you specify the Op property as "and", a condition is satisfied only if all expressions specified in the group is "true".
18. Click **Save**.

---

## Creating a Decision Table Based Advanced XML Condition

### About this task

To create a new decision table based advanced XML condition:

### Procedure

1. Expand the Java project that you created.
2. In the Project Explorer hierarchy, select the \*.greex file. Right-click **Open With → Greex Model Editor**.
3. Expand the tree structure that appears in the Greex editor panel. Click on a node from the tree. All the child leaves of the node are listed. The Document Root element contains the root element Greex Rule.
4. Select the root element Greex Rule and in the Properties view, enter the values for various attributes.

The following table describes various attributes of the root element Greex Rule.

In the Properties view, you can view various properties of the selected element. To open the Properties View, from the menu bar, select **Window** → **Show View** → **Other...** From the list of views under Basic, select **Properties**.

Attribute	Description
Desc	Enter the description of the Greex rule.
Id	Enter the unique identifier of the Greex rule
Name	Enter the name of the Greex rule.
Return Type	Enter the return type of the Greex rule. For a decision table based Greex rule, the only valid value is "String". The decision table based Greex rule can only return a String.
Type	Enter the Greex rule type. By default, it is a normal Greex rule. To make it a decision table based Greex rule, select "DecisionTable" from the drop-down list.

5. Under the root element Greex Rule, create a new IF construct element as per the requirement. Right-click on the **root element Greex Rule** and select **New Child** → **If** from the pop-up menu.

**Note:** A decision table based Greex rule cannot have an ELSE construct. It can only have a single IF construct.

6. Select the **If** element and in the Properties view, enter the name of the If element in the Name property.
7. Under the If element, create a new Condition child element. Right-click on the **If** element and select **New Child** → **Condition** from the pop-up menu.
8. Select the **Condition** element and in the Properties view, enter the name of the Condition element in the Name property.
9. As every condition must return a value, under the If element, create a new Return element and specify the appropriate return value for the associated condition in the Value element. Right-click on the **If** element and select **New Child** → **Return** from the pop-up menu.
10. Right-click on the **Return** element and select **New Child** → **Value** from the pop-up menu. Enter the value pop-up window displays.
11. Enter the value which you want to return if the IF condition gets satisfied.
12. Select the **Return** element. In the Properties view, in the Default property, specify the default value which you want to return if none of the IF conditions are satisfied and in the Output property, specify the value which you want to return if the IF condition gets satisfied.

**Note:** For a decision table based Greex rule, it is mandatory to give a default return value.

For a decision table based Greex rule, you can define an array of constant values in the Value element. The multiple values are separated using the "|" operator. For example, 5|7|3.

13. Under the Condition element, create a new Expression element to specify the expression(s) that you want to evaluate for the condition to be satisfied. Right-click on the **Condition** element and select **New Child** → **Expression** from the pop-up menu. The Edit Expression pop-up window displays.
14. In Expression, enter the expression that you want to evaluate.

You can make function calls in the expression by prefixing the function name with "fn:". You can also pass functions to other functions.

**Note:** You can press **Ctrl+Space** and select the expression from the drop-down list.

For a decision table based Greex rule, you can define an array of constant values for a particular XML attribute in the Expression element. The multiple values are separated using the "|" operator. For example, `fn:equals(@orderType, "WEB|STORE|CALLCENTER")`.

15. (Optional) If you want to evaluate a set of expressions together, you must group the set of expressions. Under the Condition element, create a new Group element to group a set of expressions together. Right-click on the **Condition** element and select **New Child** → **Group** from the pop-up menu. Now, you can add more than one expression to this group. Right-click on the **Group** element and select **New Child** → **Expression** from the pop-up menu. You can also add a new Group element to a Group element. Right-click on the Group element and select **New Child** → **Group** from the pop-up menu.

**Note:** Any level of nesting of Group and Expression elements is allowed.

16. (Optional) Select the **Group** element and in the Properties view, specify the operation that you want to perform on the set of expressions in the "op" property. The valid values are: "or" and "and".  
If you specify the Op property as "or" then a condition gets satisfied if any of the expressions specified in the group evaluates to true.  
If you specify the Op property as "and" then a condition gets satisfied only if all of the expressions specified in the group evaluate to true.
17. Save your changes by clicking on the **Save** button.

---

## Validating an Advanced XML Condition

### About this task

After creating an advanced XML condition, you must validate its syntax and structure. For example, if an advanced XML condition is of decision table type, it must contain only one IF construct and no ELSE constructs. Therefore, you must validate an advanced XML condition before loading it into the database.

To validate an advanced XML condition:

### Procedure

1. Expand the Java project that you created.
2. In the Project Explorer hierarchy, select the \*.greex file. Right-click and select **Open With** → **Greex Model Editor** from the pop-up menu.
3. Expand the Greex rule that appears in the Greex editor panel. Click on a node from the tree. All child leaves of the node are listed. The Document Root element contains the Greex Rule root element.
4. Select the Greex Rule root element and right-click. Select **Validate Greex Rule** from the pop-up menu. If the Greex rule that you created uses correct syntax and structure, the message "Greex rule validation succeeded" displays. Otherwise, the system displays an appropriate error message.



---

## Loading Advanced XML Conditions into the Database

After creating an advanced XML condition or Greex Rule using the Rich Client Platform Greex Editor, load the \*.greex file into the database. This file contains the newly created advanced XML condition that you want to load into the database. Using the Applications Manager you can modify appropriate parameters of the advanced XML condition (if necessary).

**Note:** Before loading an advanced XML condition or Greex rule into the database, you must validate the advanced XML condition for a valid structure or syntax. For more information, see “Validating an Advanced XML Condition” on page 26.

### Before Importing or Loading an Advanced XML Condition

#### Procedure

1. Modify the *ECLIPSE\_HOME*/plugins/com.yantra.ide.othertools.greex.editor\_ *VERSION\_NUMBER*/lib/greexide.properties file and specify the JDBC connection properties based on the database you are using. *ECLIPSE\_HOME* is the directory where Eclipse SDK is installed. *VERSION\_NUMBER* is the current version number of the Greex editor plug-in.
2. Depending on the database, copy the required database driver jar file to the *ECLIPSE\_HOME*/plugins/com.yantra.ide.othertools.greex.editor\_ *VERSION\_NUMBER*/lib directory. For example, if you are using an Oracle database, copy the ojdbc14.jar file.
3. Edit the *ECLIPSE\_HOME*/plugins/com.yantra.ide.othertools.greex.editor\_ *VERSION\_NUMBER*/META-INF/MANIFEST.MF file and add the relative path (relative to the *ECLIPSE\_HOME*/plugins/com.yantra.ide.othertools.greex.editor\_ *VERSION\_NUMBER* directory) of the required database driver jar file in the Bundle-ClassPath property.  
For example, if you are using an Oracle database, add lib/ojdbc14.jar to the Bundle-ClassPath property, as follows:  
Bundle-ClassPath: greexeditor.jar, lib/ojdbc14.jar
4. If Eclipse SDK is already running, restart Eclipse in clean mode, as follows:  
eclipse.exe -clean

### Steps to Load an Advanced XML Condition

#### About this task

To load an advanced XML condition into the database:

#### Procedure

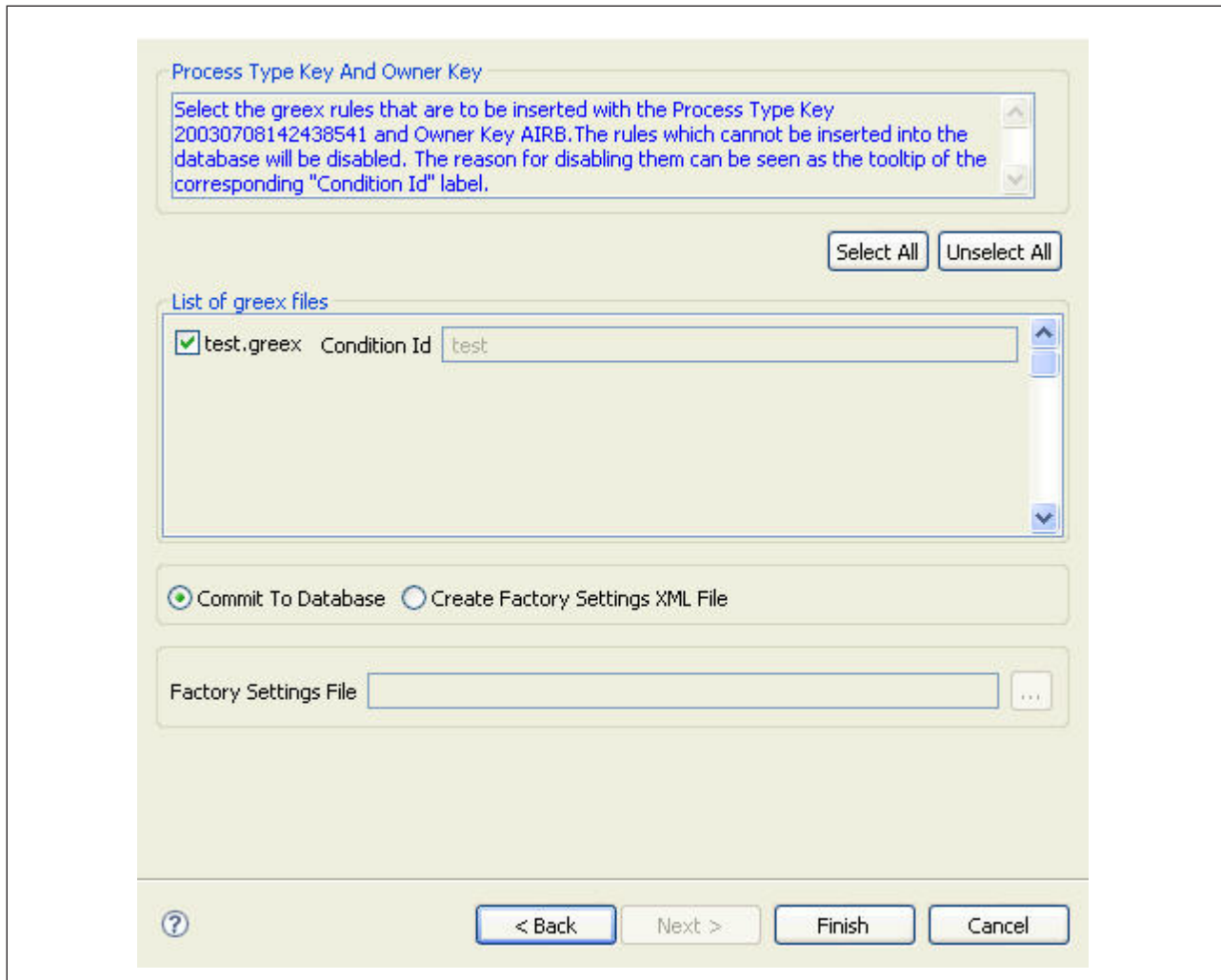
1. Start Eclipse SDK.
2. From the menu bar, select **Window** → **Show View** → **Navigator**. The Java project displays in the Navigator view.
3. Expand the Java project that you created.
4. Right-click the folder that contains all \*.greex files, which contains an advanced XML condition that you want to load into the database. Select **Greex** → **Export Greex Rule to Database** from the pop-up menu. The Export Greex Rule to Database window displays.

The screenshot shows a software window with a light beige background. At the top, there are two dropdown menus. The first is labeled "Process Type Key" and contains the value "20030708142438541". The second is labeled "Owner Key" and contains the value "AIRB". Below these menus is a large empty space. At the bottom of the window, there are four buttons: "< Back", "Next >", "Finish", and "Cancel". The "Next >" button is highlighted with a black border.


Field	Description
Process Type Key	Select the key of the appropriate process type from the drop-down list.
Owner Key	Select the key of the appropriate owner from the drop-down list.

5. Click **Next**.

The Select Greex Rules window displays.



Field	Description
Process Type Key and Owner Key	Displays the process type and owner keys that you selected.
Select All	Click this button if you want to select all *.greex files listed in List of Greex Files panel.
Unselect All	Click this button if you want to unselect all *.greex files listed in the List of Greex Files panel.
List of Greex files	Select the *.greex files that you want to load into the database.
Connection Id	Displays the unique identifier of a Greex rule defined in each *.greex file that you select.
Commit To Database	Choose this option to commit the selected Greex rules to the database.
Create Factory Settings XML File	Choose this option to create the factory setup XML file for the selected Greex rules.

Field	Description
Factory Settings File	<p>Enter the path and name of the new factory settings XML file in which you want to store the YFS_CONDITION Factory Settings.</p> <p>Click . The Select XML File to Store the YFS_CONDITION Factory Settings pop-up window displays. Select the factory settings XML file in which you want to store the YFS_CONDITION Factory Settings.</p>

6. Click **Finish**.

---

## Importing Advanced XML Conditions From the Database

You can import the existing advanced XML conditions or Greex rules from the database to your local machine. This is useful if you want to modify the structure of an advanced XML condition, such as by adding new expressions or IF and ELSE constructs to an existing advanced XML condition.

### Before Importing or Loading an Advanced XML Condition Procedure

1. Modify the `ECLIPSE_HOME/plugins/com.yantra.ide.othertools.greex.editor_VERSION_NUMBER/lib/greexide.properties` file and specify the JDBC connection properties based on the database you are using. `ECLIPSE_HOME` is the directory where Eclipse SDK is installed. `VERSION_NUMBER` is the current version number of the Greex editor plug-in.
2. Depending on the database, copy the required database driver jar file to the `ECLIPSE_HOME/plugins/com.yantra.ide.othertools.greex.editor_VERSION_NUMBER/lib` directory. For example, if you are using an Oracle database, copy the `ojdbc14.jar` file.
3. Edit the `ECLIPSE_HOME/plugins/com.yantra.ide.othertools.greex.editor_VERSION_NUMBER/META-INF/MANIFEST.MF` file and add the relative path (relative to the `ECLIPSE_HOME/plugins/com.yantra.ide.othertools.greex.editor_VERSION_NUMBER` directory) of the required database driver jar file in the `Bundle-ClassPath` property. For example, if you are using an Oracle database, add `lib/ojdbc14.jar` to the `Bundle-ClassPath` property, as follows:  

```
Bundle-ClassPath: greexeditor.jar, lib/ojdbc14.jar
```
4. If Eclipse SDK is already running, restart Eclipse in clean mode, as follows:  

```
eclipse.exe -clean
```

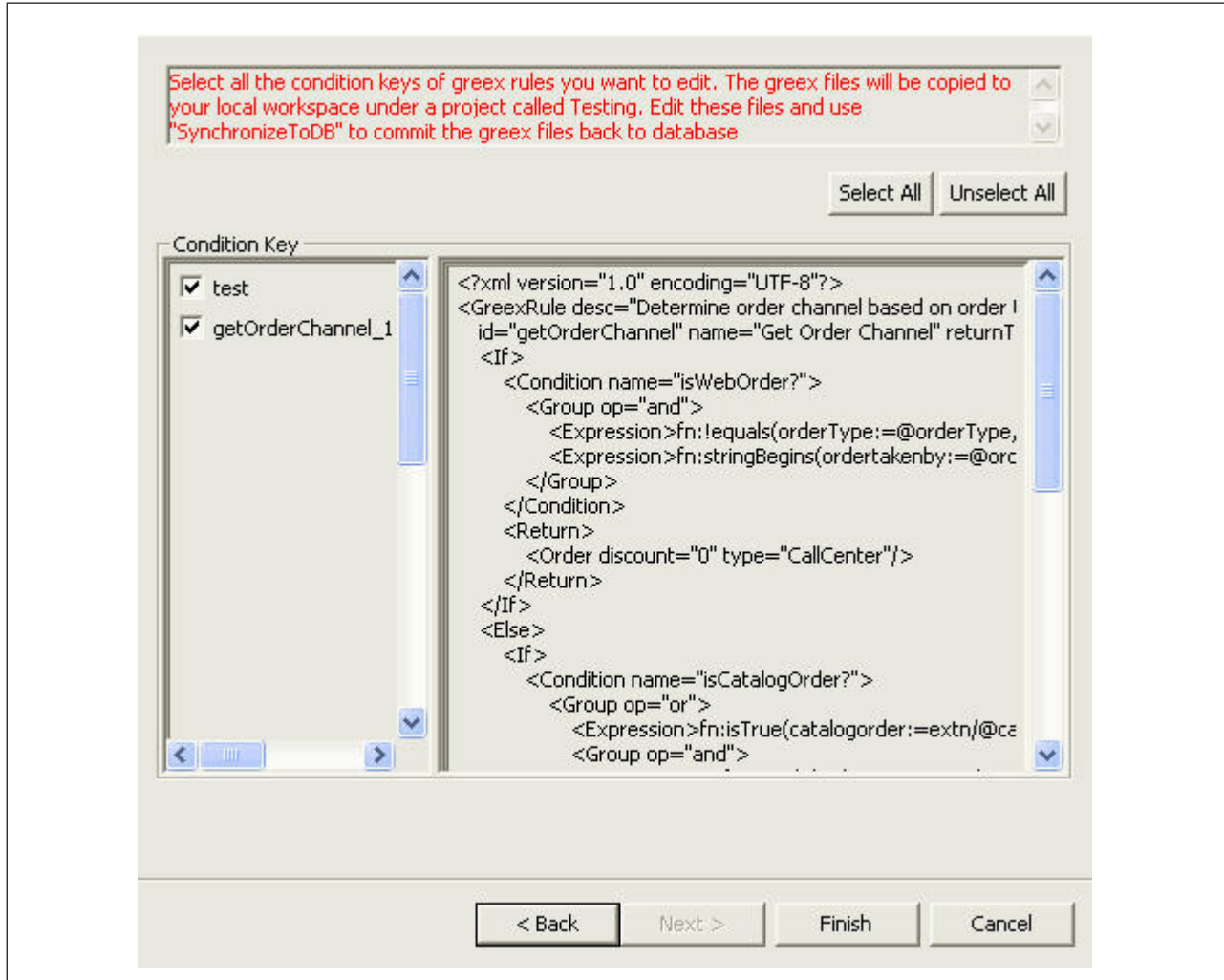
### Steps to Import Advanced XML Conditions About this task

To import advanced XML conditions from the database:

#### Procedure

1. Start Eclipse SDK.
2. From the menu bar, select **Window** → **Show View** → **Navigator**. The Java projects display in the Navigator view.

- Right-click the Java project in which you want to import the \*.greex files from the database.
- Select **Greex > Import Greex Rule from Database** from the pop-up menu. The Import Greex Rule from Database window displays. Each \*.greex file contains an advanced XML condition or Greex rule.



Field	Description
Select All	Click this button if you want to select all *.greex files listed in the Condition Key panel.
Unselect All	Click this button if you want to unselect all *.greex files listed in the Condition Key panel.
Condition Key	Displays the list of condition identifiers specified for each *.greex file loaded in the database. If you position the mouse on a particular condition identifier, in the right panel, the contents of the selected advanced XML condition or Greex rule displays.  For example, Greex rule or the advanced XML condition identifier, name, description, and so forth. All IF and ELSE constructs defined for that advanced XML condition also displays. Select or unselect the appropriate *.greex files that you do not want to copy to your local machine.

5. Click **Finish**. The selected \*.greex files are imported into the Java project.
6. Open the \*.greex files using the Greex editor and modify the advanced XML condition or Greex rule as needed.
7. Reload the modified \*.greex files into the database as described in Steps to Load an Advanced XML Condition.

---

## Modifying an Advanced XML Condition

Using the Applications Manager, you can only change the modifiable parameters of an advanced XML condition. You cannot change the structure of an advanced XML condition. For more information about modifying advanced XML conditions, see the Sterling Selling and Fulfillment Foundation: Configuration Guide .

---

## Notices

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

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

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

*IBM Director of Licensing*

*IBM Corporation*

*North Castle Drive*

*Armonk, NY 10504-1785*

*U.S.A.*

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

*Intellectual Property Licensing*

*Legal and Intellectual Property Law*

*IBM Japan Ltd.*

*1623-14, Shimotsuruma, Yamato-shi*

*Kanagawa 242-8502 Japan*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be

incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

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

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

*IBM Corporation*

*J46A/G4*

*555 Bailey Avenue*

*San Jose, CA 95141-1003*

*U.S.A.*

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

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

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

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.



This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

#### COPYRIGHT LICENSE:

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

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

© IBM 2011. Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 2011.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

#### **Trademarks**

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

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

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

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

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium and the Ultrium Logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Connect Control Center<sup>®</sup>, Connect:Direct<sup>®</sup>, Connect:Enterprise<sup>™</sup>, Gentran<sup>®</sup>, Gentran<sup>®</sup>:Basic<sup>®</sup>, Gentran:Control<sup>®</sup>, Gentran:Director<sup>®</sup>, Gentran:Plus<sup>®</sup>, Gentran:Realtime<sup>®</sup>, Gentran:Server<sup>®</sup>, Gentran:Viewpoint<sup>®</sup>, Sterling Commerce<sup>™</sup>, Sterling Information Broker<sup>®</sup>, and Sterling Integrator<sup>®</sup> are trademarks or registered trademarks of Sterling Commerce<sup>™</sup>, Inc., an IBM Company.

Other company, product, and service names may be trademarks or service marks of others.





Product Number: xxxx-xxx

Printed in USA