Red Hat Application Migration Toolkit 4.2

Rules Development Guide

Create custom rules to enhance migration coverage.

Last Updated: 2018-10-31

Red Hat Application Migration Toolkit 4.2 Rules Development Guide

Create custom rules to enhance migration coverage.

Legal Notice

Abstract

This guide describes how to create custom XML rules for the Red Hat Application Migration Toolkit.

Table of Contents

CHAPTER 1. INTRODUCTION

1.1. ABOUT THE RULES DEVELOPMENT GUIDE

This guide is for engineers, consultants, and others who want to create custom XML-based rules for Red Hat Application Migration Toolkit (RHAMT) tools.

If you are new to RHAMT, it is recommended that you start with the *Getting Started Guide* for an overview of Red Hat Application Migration Toolkit features and system requirements. It is also recommended that you review the *CLI Guide*, which provides detailed instructions on how to install and execute the CLI.

If you would like to contribute to the RHAMT source code base or provide Java-based rule add-ons, see the *Core Development Guide*.

1.1.1. Use of RHAMT_HOME in This Guide

This guide uses the ***RHAMT_HOME*** replaceable variable to denote the path to your RHAMT installation. The installation directory is the **`rhamt-cli-4.2.0.Final`** directory where you extracted the RHAMT ZIP distribution.

When you encounter ***RHAMT_HOME*** in this guide, be sure to replace it with the actual path to your RHAMT installation.

1.2. RHAMT RULES

Red Hat Application Migration Toolkit (RHAMT) contains rule-based migration tools that analyze the APIs, technologies, and architectures used by the applications you plan to migrate. In fact, the RHAMT analysis process is implemented using RHAMT rules. RHAMT uses rules internally to extract files from archives, decompile files, scan and classify file types, analyze XML and other file content, analyze the application code, and build the reports.

RHAMT builds a data model based on the rule execution results and stores component data and relationships in a graph database, which can then be queried and updated as needed by the migration rules and for reporting purposes.

RHAMT rules use the following rule pattern:

```
when(condition)
  perform(action)
otherwise(action)
```

RHAMT provides a comprehensive set of standard migration rules out-of-the-box. Because applications may contain custom libraries or components, RHAMT allows you to write your own rules to identify use of components or software that may not be covered by the existing ruleset.

CHAPTER 2. GETTING STARTED

You can get starting creating custom RHAMT rules by walking through creating a rule or by reviewing the quickstarts.

2.1. CREATE YOUR FIRST XML RULE

This section guides you through the process of creating and testing your first RHAMT XML-based rule. This assumes that you have already installed RHAMT. See the *CLI Guide* for installation instructions.

In this example, you will write a rule to discover instances where an application defines a **`jboss-web.xml`** file containing a **`<class-loading>`** element and provide a link to the documentation that describes how to migrate the code.

Create the Directory Structure for the Rule

Create a directory structure to contain your first rule and the data file to use for testing.

```
$ mkdir -p /home/USER_NAME/migration-rules/rules
$ mkdir -p /home/USER_NAME/migration-rules/data
```

This directory structure will also be used to hold the generated RHAMT reports.

Create Data to Test the Rule

1. Create a **`jboss-web.xml`** file in the **`/home/USER_NAME/migration-rules/data/`** subdirectory.
2. Copy in the following content.

```
<!DOCTYPE jboss-web PUBLIC "-//JBoss//DTD Web Application 4.2//EN"
"http://www.jboss.org/j2ee/dtd/jboss-web_4_2.dtd">
<jboss-web>
    <class-loading java2ClassLoadingCompliance="false">
        <loader-repository>
            seam.jboss.org:loader=@projectName@
            <loader-repository-
config>java2ParentDelegation=false</loader-repository-config>
        </loader-repository>
    </class-loading>
</jboss-web>
```

Create the Rule

RHAMT XML-based rules use the following rule pattern:

```
when(condition)
  perform(action)
otherwise(action)
```

Ruleset and rule XML elements are covered in more detail in the XML Rule Structure section. See Create a Basic XML Rule for additional details about creating XML rules with example syntax.

1. Create an XML file in the **`/home/USER_NAME/migration-rules/rules/`** subdirectory named **`JBoss5-web-class-loading.windup.xml`**. Copy in the following content.

```
<?xml version="1.0"?>
<ruleset id="UNIQUE_RULESET_ID"
    xmlns="http://windup.jboss.org/schema/jboss-ruleset"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-
ruleset.xsd">
    <metadata>
        <description>
            <!-- Ruleset Description -->
        </description>
        <dependencies>
            <!-- Ruleset Dependencies -->
        </dependencies>
        <sourceTechnology id="SOURCE_ID"
versionRange="SOURCE_VERSION_RANGE"/>
        <targetTechnology id="TARGET_ID"
versionRange="TARGET_VERSION_RANGE"/>
        <tag>Reviewed-2015-05-01</tag>
    </metadata>
    <rules>
        <rule id="UNIQUE_RULE_ID">
            <when>
                <!-- Test for a condition here -->
            </when>
            <perform>
                <!-- Perform an action -->
            </perform>
        </rule>
     </rules>
</ruleset>
```

NOTE

RHAMT identifies files with the **`.windup.xml`** or **`.rhamt.xml`** extension as XML-based rules, so be sure to use this naming convention, otherwise the rule will not be evaluated!

2. Add the unique identifier for the ruleset and rule.
   - Replace the ***`UNIQUE_RULESET_ID`*** with an appropriate ruleset ID, for example, **`JBoss5-web-class-loading`**.
   - Replace the ***`UNIQUE_RULE_ID`*** with an appropriate rule ID, for example, **`JBoss5-web-class-loading_001`**.
3. Add the following ruleset add-on dependencies.

```
<dependencies>
  <addon id="org.jboss.windup.rules,windup-rules-
javaee,3.0.0.Final"/>
  <addon id="org.jboss.windup.rules,windup-rules-java,3.0.0.Final"/>
</dependencies>
```

4. Add the source and target technologies.
   - Replace ***SOURCE_ID*** with **eap**.
   - Replace ***TARGET_ID*** with **eap**.
5. Set the source and target technology versions.
   - Replace ***SOURCE_VERSION_RANGE*** with **(4,5)**.
   - Replace ***TARGET_VERSION_RANGE*** with **[6,)**.

   See the Apache Maven version range specification for help with this syntax.
6. Complete the **when** condition.
   Because this rule tests for a match in an XML file, **xmlfile** is used to evaluate the files.

   To match on the **class-loading** element that is a child of **jboss-web**, use the xpath expression **jboss-web/class-loading**.

```
<when>
    <xmlfile matches="jboss-web/class-loading" />
</when>
```

7. Complete the **perform** action for this rule.
   - Add a classification with a descriptive title and a level of effort of **1**.
   - Provide a hint with an informative message and a link to documentation that describes the migration details.

```
<perform>
    <iteration>
        <classification title="JBoss Web Application Descriptor"
effort="1"/>
        <hint title="JBoss Web XML class-loading element is no
longer valid">
          <message>
            The class-loading element is no longer valid in the
jboss-web.xml file.
          </message>
          <link href="https://access.redhat.com/documentation/en-
US/JBoss_Enterprise_Application_Platform/6.4/html-
single/Migration_Guide/index.html#Create_or_Modify_Files_That_Con
trol_Class_Loading_in_JBoss_Enterprise_Application_Platform_6"
title="Create or Modify Files That Control Class Loading in JBoss
EAP 6"/>
        </hint>
    </iteration>
</perform>
```

The rule is now complete and should look like the following example.

```
<?xml version="1.0"?>
<ruleset id="JBoss5-web-class-loading"
```

```
    xmlns="http://windup.jboss.org/schema/jboss-ruleset"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
    <metadata>
        <description>
            This ruleset looks for the class-loading element in a jboss-
web.xml file, which is no longer valid in JBoss EAP 6
        </description>
         <dependencies>
            <addon id="org.jboss.windup.rules,windup-rules-
javaee,3.0.0.Final"/>
            <addon id="org.jboss.windup.rules,windup-rules-
java,3.0.0.Final"/>
        </dependencies>
        <sourceTechnology id="eap" versionRange="(4,5)"/>
        <targetTechnology id="eap" versionRange="[6,)"/>
    </metadata>
    <rules>
        <rule id="JBoss5-web-class-loading_001">
            <when>
                <xmlfile matches="jboss-web/class-loading" />
            </when>
            <perform>
                <iteration>
                    <classification title="JBoss Web Application
Descriptor" effort="1"/>
                    <hint title="JBoss Web XML class-loading element is
no longer valid">
                      <message>
                        The class-loading element is no longer valid in
the jboss-web.xml file.
                      </message>
                      <link
href="https://access.redhat.com/documentation/en-
US/JBoss_Enterprise_Application_Platform/6.4/html-
single/Migration_Guide/index.html#Create_or_Modify_Files_That_Control_Clas
s_Loading_in_JBoss_Enterprise_Application_Platform_6" title="Create or
Modify Files That Control Class Loading in JBoss EAP 6"/>
                    </hint>
                </iteration>
            </perform>
        </rule>
     </rules>
</ruleset>
```

Install the Rule

An RHAMT rule is installed by placing the rule into the appropriate directory. See Add the Rule to RHAMT for the possible locations to place a custom rule.

Copy the **`JBoss5-web-class-loading.windup.xml`** file to the ***`RHAMT_HOME`*`/rules/`** directory.

```
$ cp /home/USER_NAME/migration-rules/rules/JBoss5-web-class-
loading.windup.xml RHAMT_HOME/rules/
```

Test the Rule

Open a terminal and execute the following command, passing the test file as an input argument and a directory for the output report.

```
$ RHAMT_HOME/bin/rhamt-cli --sourceMode --input /home/USER_NAME/migration-
rules/data --output /home/USER_NAME/migration-rules/reports --target eap:6
```

You should see the following result.

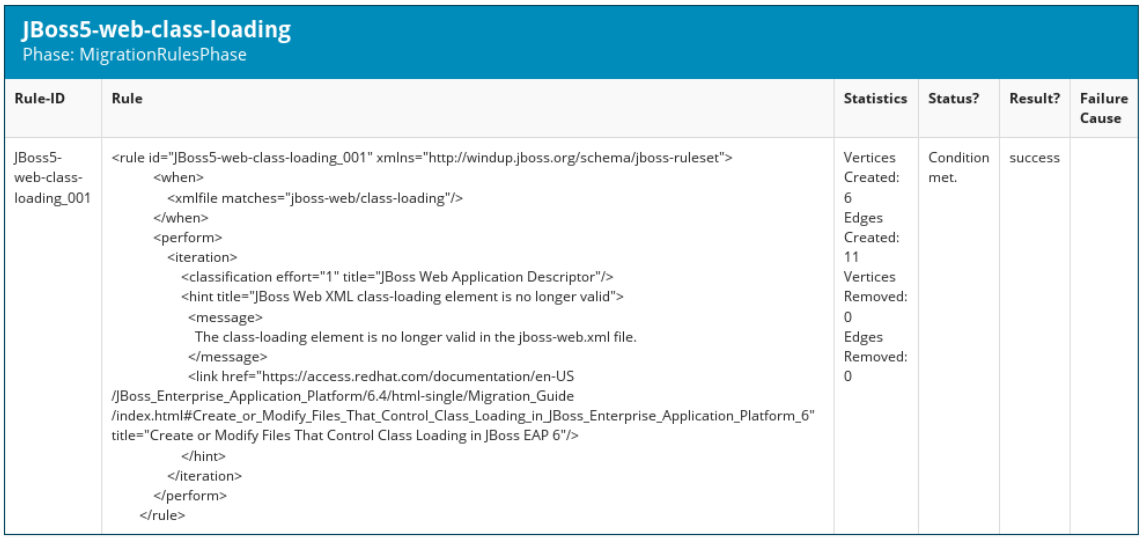
```
Report created: /home/USER_NAME/migration-rules/reports/index.html
              Access it at this URL: file:///home/USER_NAME/migration-
rules/reports/index.html
```

Review the Reports

Review the report to be sure that it provides the expected results. For a more detailed walkthrough of RHAMT reports, see the Review the Reports section of the RHAMT *CLI Guide*.

1. Open **`/home/USER_NAME/migration-rules/reports/index.html`** in a web browser.
2. Verify that the rule executed.
   a. From the main landing page, click the **Rule providers execution overview** link to open the Rule Providers Execution Overview.
   b. Find the **`JBoss5-web-class-loading_001`** rule and verify that its **Status?** is **`Condition met`** and its **Result?** is **`success`**.

   Figure 2.1. Test Rule Execution

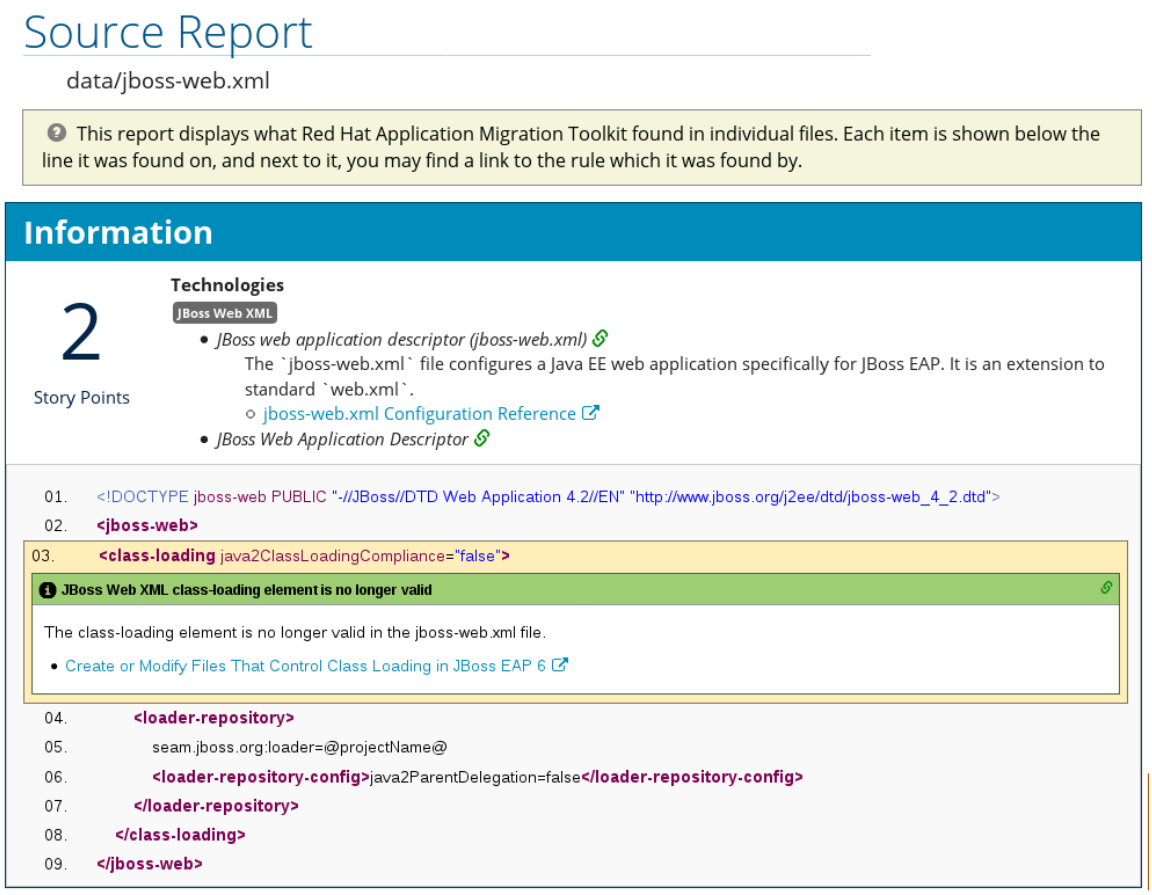


JBoss5-web-class-loading
Phase: MigrationRulesPhase

Rule-ID	Rule	Statistics	Status?	Result?	Failure Cause
JBoss5-web-class-loading_001	<rule id="JBoss5-web-class-loading_001" xmlns="http://windup.jboss.org/schema/jboss-ruleset"> <when> <xmlfile matches="jboss-web/class-loading"/> </when> <perform> <iteration> <classification effort="1" title="JBoss Web Application Descriptor"/> <hint title="JBoss Web XML class-loading element is no longer valid"> <message> The class-loading element is no longer valid in the jboss-web.xml file. </message> <link href="https://access.redhat.com/documentation/en-US /JBoss_Enterprise_Application_Platform/6.4/html-single/Migration_Guide /index.html#Create_or_Modify_Files_That_Control_Class_Loading_in_JBoss_Enterprise_Application_Platform_6" title="Create or Modify Files That Control Class Loading in JBoss EAP 6"/> </hint> </iteration> </perform> </rule>	Vertices Created: 6 Edges Created: 11 Vertices Removed: 0 Edges Removed: 0	Condition met.	success	

3. Verify that the rule matched on the test data.
   a. From the main landing page, click on the name of the application or input folder, which is **`data`** in this example.
   b. Click on the **Application Details** report link.
   c. Click on the **jboss-web.xml** link to view the **Source Report**.
   You can see that the **`<class-loading>`** line is highlighted, and the hint from the custom rule is shown inline.

Figure 2.2. Rule Match



The top of the file lists the classifications for matching rules. You can use the link icon to view the details for that rule. Notice that in this example, the **`jboss-web.xml`** file matched on another rule (**`JBoss web application descriptor (jboss-web.xml)`**) that produced **`1`** story point. This, combined with the **`1`** story point from our custom rule, brings the total story points for this file to **`2`**.

2.2. REVIEW THE RED HAT APPLICATION MIGRATION TOOLKIT QUICKSTARTS

The Red Hat Application Migration Toolkit quickstarts provide working examples of how to create custom Java-based rule add-ons and XML rules. You can use them as a starting point for creating your own custom rules.

You can download a ZIP file of the latest released version of the quickstarts. Or, if you prefer to work with the source code, you can fork and clone the **`windup-quickstarts`** project repository.

Each quickstart has a **`README.adoc`** file that contains instructions for that quickstart.

Download the Latest Quickstart ZIP

1. Open a browser and navigate to https://github.com/windup/windup-quickstarts/releases.
2. Click on the most recent release to download the ZIP file to your local file system.

Fork and Clone the Quickstart GitHub Project

You must have the **`git`** client installed on your machine.

1. Click the **Fork** link on the Red Hat Application Migration Toolkit quickstart GitHub page to create the project in your own Git. The forked GitHub repository URL created by the fork should look like this: **https://github.com/*YOUR_USER_NAME*/windup-quickstarts.git**.

2. Clone your Red Hat Application Migration Toolkit quickstart repository to your local file system:

```
$ git clone https://github.com/YOUR_USER_NAME/windup-quickstarts.git
```

3. This creates and populates a **windup-quickstarts** directory on your local file system. Navigate to the newly created directory, for example

```
$ cd windup-quickstarts/
```

4. If you want to be able to retrieve the latest code updates, add the remote **upstream** repository so you can fetch any changes to the original forked repository.

```
$ git remote add upstream https://github.com/windup/windup-
quickstarts.git
```

5. Get the latest files from the **upstream** repository.

```
$ git fetch upstream
```

CHAPTER 3. CREATING XML RULES

3.1. XML RULE STRUCTURE

This section describes the basic structure of XML rules. All XML rules are defined as elements within rulesets. For more details, see the RHAMT XML rule schema.

3.1.1. Rulesets

A ruleset is a group of one or more rules that targets a specific area of migration. This is the basic structure of the **<ruleset>** element.

- **<ruleset id="*UNIQUE_RULESET_ID*">**: Defines this as an RHAMT ruleset and gives it a unique ruleset ID.
  - **<metadata>**: The metadata about the ruleset.
    - **<description>**: The description of the ruleset.
    - **<dependencies/>**: The rule add-ons required by this ruleset.
    - **<sourceTechnology/>**: The source technology.
    - **<targetTechnology/>**: The target technology.
    - **<overrideRules/>**: Setting to **`true`** indicates that rules in this ruleset override rules with the same ID from the core ruleset distributed with RHAMT. Both the ruleset id and the rule id must match a rule within the core ruleset or the rule will be ignored. This is **`false`** by default.
  - **<rules>**: A set of individual rules.
    - **<rule id="*UNIQUE_RULE_ID*">**: Defines the rule and gives it a unique ID. It is recommended to include the ruleset ID as part of the rule ID, for example, ***`UNIQUE_RULESET_ID_UNIQUE_RULE_ID`***. One or more rules can be defined for a ruleset.
      - **<when>**: The conditions to match on. For a detailed description of the elements allowed in a **<when>**, see When Condition Syntax.
      - **<perform>**: The action to be performed when the rule condition is matched. For a detailed description of the elements allowed in a **`<perform>`**, see Perform Action Syntax.
      - **<otherwise>**: The action to be performed when the rule condition is not matched. This element takes the same child elements as the **`<perform>`** element.
      - **<where>**: A string pattern defined as a parameter, which can be used elsewhere in the rule definition. For more information, see Where Syntax.
    - **<file-mapping/>**: Maps an extension to a graph type.
    - **<package-mapping/>**: Maps from a package pattern (regular expression) to a organization name.

3.1.2. Predefined Rules

RHAMT provides predefined rules for common migration requirements. These core RHAMT rules are located in the RHAMT installation at ***RHAMT_HOME*/rules/migration-core/**.

The following is an example of a core RHAMT rule that matches on a proprietary utility class.

```
<?xml version="1.0"?>
<ruleset xmlns="http://windup.jboss.org/schema/jboss-ruleset"
id="weblogic" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">

    <metadata>
        <description>
            This ruleset provides analysis of WebLogic proprietary classes
and constructs that may require individual attention when migrating to
JBoss EAP 6+.
        </description>
        <dependencies>
            <addon id="org.jboss.windup.rules,windup-rules-
javaee,2.0.1.Final" />
            <addon id="org.jboss.windup.rules,windup-rules-
java,2.0.0.Final" />
        </dependencies>
        <sourceTechnology id="weblogic" />
        <targetTechnology id="eap" versionRange="[6,)" />
        <tag>reviewed-2015-06-02</tag>
        <tag>weblogic</tag>
    </metadata>
    <rules>
        ...
        <rule id="weblogic-02000">
            <when>
                <javaclass references="weblogic.utils.StringUtils.{*}" />
            </when>
            <perform>
                <hint title="WebLogic StringUtils usage" effort="1"
category-id="mandatory">
                    <message>Replace with the `StringUtils` class from
Apache Commons.</message>
                    <link
href="https://commons.apache.org/proper/commons-lang/" title="Apache
Commons Lang" />
                    <tag>weblogic</tag>
                </hint>
            </perform>
        </rule>
        ...
    </rules>
</ruleset>
```

3.2. CREATE A BASIC XML RULE

This section describes how to create an RHAMT XML rule. This assumes that you already have RHAMT installed. See the RHAMT *CLI Guide* for installation instructions.

3.2.1. Create the Basic XML Rule Template

RHAMT XML rules consist of *conditions* and *actions* and use the following rule pattern.

```
when(condition)
  perform(action)
otherwise(action)
```

Create a file with the following contents, which is the basic syntax for XML rules.

IMPORTANT

The RHAMT XML rule file must use the **`.windup.xml`** or **`.rhamt.xml`** extension, otherwise the rule will not be evaluated.

```
<?xml version="1.0"?>
<ruleset id="unique-ruleset-id"
    xmlns="http://windup.jboss.org/schema/jboss-ruleset"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
    <metadata>
        <!-- Metadata about the rule including a description,
             source technology, target technology, and any
             add-on dependencies -->
    </metadata>
    <rules>
        <rule id="unique-ruleset-id-01000">
            <when>
                <!-- Test a condition... -->
            </when>
            <perform>
                <!-- Perform this action when condition is satisfied -->
            </perform>
            <otherwise>
                <!-- Perform this action when condition is not satisfied
-->
            </otherwise>
        </rule>
    <rules>
</ruleset>
```

3.2.2. Create the Ruleset Metadata

The XML ruleset **`metadata`** element provides additional information about the ruleset such as a description, the source and target technologies, and add-on dependencies. The metadata also allows for specification of tags, which allow you to provide additional information about a ruleset. For more information about ruleset metadata, see XML Rule Structure.

Example Rule <metadata>

```
<ruleset id="unique-ruleset-id"
    xmlns="http://windup.jboss.org/schema/jboss-ruleset"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
    <metadata>
        <description>
                This is the description.
        </description>
        <dependencies>
                <addon id="org.jboss.windup.rules,windup-rules-
javaee,2.0.1.Final"/>
                <addon id="org.jboss.windup.rules,windup-rules-
java,2.0.0.Final"/>
        </dependencies>
        <sourceTechnology id="weblogic" versionRange="(10,12]"/>
        <sourceTechnology id="ejb" versionRange="(2,3]"/>
        <targetTechnology id="eap" versionRange="(5,6]"/>
        <targetTechnology id="ejb" versionRange="(2,3]"/>
        <tag>require-stateless</tag>
        <tag>require-nofilesystem-io</tag>
        <executeAfter>AfterRulesetId</executeAfter>
        <executeBefore>BeforeRulesetId</executeBefore>
    </metadata>
    <rules>
         ...
    </rules>
</ruleset>
```

3.2.3. Create the Rule

Individual rules are contained within the **<rules>** element and consist of one or more when conditions and perform actions.

See the XML rule schema for valid rule syntax.

3.2.3.1. Create the Rule When Condition

The XML rule **<when>** element tests for a condition. The following is a list of valid **<when>** conditions.

Element	Description
<and>	The standard logical *and* operator.
<filecontent>	Find strings or text within files, for example, properties files.
<file-mapping>	Define file names to internal stored File model.
<javaclass>	Test for a match in a Java class.
<javaclass-ignore>	Exclude javaclass which you would like to ignore in processing discovery.

Element	Description
<not>	The standard logical *not* operator.
<or>	The standard logical *or* operator.
<package-mapping>	Define package names to organization or libraries.
<project>	Test for project characteristics, such as dependencies.
<true>	Always match.
<xmlfile>	Test for a match in an XML file.

The specific syntax is dependent on whether you are creating a rule to evaluate Java class, an XML file, a project, or file content. This is described in more detail in When Condition Syntax.

3.2.3.2. Create the Rule Perform Action

The XML rule **<perform>** element performs the action when the condition is met. Operations allowed in this section of the rule include the classification of application resources, in-line hints for migration steps, links to migration information, and project lineitem reporting. The following is a list of valid **<perform>** actions.

Element	Description
<classification>	This operation adds metadata that is intended to apply to the entire file. For example, if the Java Class is a JMS Message Listener, you might want to add a Classification with the title "JMS Message Listener". Information that would apply to the entire file would go here. Also, if an effort level is set, that information would apply to the entire file.
<hint>	This operation adds metadata to a line within the file. This provides a hint or inline information about how to migrate a section of code.
<iteration>	This specifies to iterate over an implicit or explicit variable defined within the rule.
<lineitem>	This provides a high-level message that will appear in the application overview page.
<link>	This provides an HTML link to additional information or documentation that provides more information about the migration task.
<xslt>	This specifies how to transform an XML file.

The syntax is described in more detail in Perform Action Syntax.

3.3. XML RULE SYNTAX

3.3.1. When Condition Syntax

Conditions allowed in the **when** portion of a rule must extend GraphOperation and currently include evaluation of Java classes, XML files, projects, and file content. Because XML rules are modeled after the Java-based rule add-ons, links to JavaDocs for the related Java classes are provided for a better understanding of how they behave.

The complete XML rule schema is located here: http://windup.jboss.org/schema/windup-jboss-ruleset.xsd.

The following sections describe the more common XML **when** rule conditions.

- <javaclass> condition syntax
- <xmlfile> condition syntax
- <project> condition syntax
- <filecontent> condition syntax
- <file> condition syntax
- <has-hint> condition syntax
- <has-classification> condition syntax
- <graph-query> condition syntax

By default, if more than one **when** rule condition is provided, then all conditions must be met for the rule to match.

3.3.1.1. <javaclass> Syntax

3.3.1.1.1. Summary

Use the **<javaclass>** element to find imports, methods, variable declarations, annotations, class implementations, and other items related to Java classes. For a better understanding of the **<javaclass>** condition, see the JavaDoc for the JavaClass class.

The following is an example of a rule that tests for WebLogic-specific Apache XML packages.

```
<rule id="weblogic-03000">
    <when>
        <javaclass references="weblogic.apache.xml.{*}" />
    </when>
    <perform>
        <hint title="WebLogic Specific Apache XML Package" effort="1"
category-id="mandatory">
            <message>
                Code using this package should be replaced with code
using the org.apache.xml package from [Apache
                Xerces](http://xerces.apache.org/).
            </message>
        </hint>
    </perform>
</rule>
```

-

3.3.1.1.2. Construct a <javaclass> Element

3.3.1.1.2.1. <javaclass> Element Attributes

Attribute Name	Type	Description
references	CLASS_NAME	The package or class name to match on. Wildcard characters can be used. This attribute is required.  **NOTE** For performance reasons, you should not start the reference with wildcard characters. For example, use **`weblogic.apache.xml.{*}`** instead of **`{web}.apache.xml.{*}`**. `references="weblogic.apache.xml.{*}"`
matchesSource	STRING	An exact regex to match. This is useful to distinguish hard-coded strings. This attribute is required. `matchesSource="log4j.logger"`
as	VARIABLE_NAME	A variable name assigned to the rule so that it can be used as a reference in later processing. See the **`from`** attribute below. `as="MyEjbRule"`
from	VARIABLE_NAME	Begin the search query with the filtered result from a previous search identified by its **`as`** VARIABLE_NAME. `from="MyEjbRule"`
in	PATH_FILTER	Filter input files matching this regex (regular expression) naming pattern. Wildcard characters can be used. `in="{*}File1"`

3.3.1.1.2.2. <javaclass> Child Elements

Child Element	Description

Child Element	Description
<location>	The location where the reference was found in a Java class. Location can refer to annotations, field and variable declarations, imports, and methods. For the complete list of valid values, see the JavaDoc for TypeReferenceLocation. `<location>IMPORT</location>`

<annotation-literal>	Match on literal values inside of annotations.
	
	The below example would match on **`@MyAnnotation(myvalue="test")`**.
	
	`<javaclass references="org.package.MyAnnotation">`
	`  <location>ANNOTATION</location>`
	`  <annotation-literal name="myvalue" pattern="test"/>`
	`</javaclass>`
	
	Note that in this case, the **`<javaclass>`** refers to an annotation (**`@MyAnnotation`**), so the top-level annotation filter, **`<annotation-literal>`** must specify the **`name`** attribute. If the **`<javaclass>`** referred to a class that is annotated, then the top-level annotation filter used would be **`<annotation-type>`**.

<annotation-type>	Match on a particular annotation type. You can supply subconditions to be matched against the annotation elements.
	
	The below example would match on a **`Calendar`** field declaration annotated with **`@MyAnnotation(myvalue="test")`**.
	
	`<javaclass references="java.util.Calendar">`
	`  <location>FIELD_DECLARATION</location>`
	`  <annotation-type pattern="org.package.MyAnnotation">`
	`  <annotation-literal name="myvalue"`
	`pattern="test"/>`
	`  </annotation-type>`
	`</javaclass>`

Child Element	Description
<annotation-list>	Match on an item in an array within an annotation. If an array index is not specified, the condition will be matched if it applies to any item in the array. You can supply subconditions to be matched against this element. The below example would match on **`@MyAnnotation(mylist={"one","two"})`**. `<javaclass references="org.package.MyAnnotation" >   <location>ANNOTATION</location>   <annotation-list name="mylist">   <annotation-literal pattern="two"/>   </annotation-list> </javaclass>` Note that in this case, the **`<javaclass>`** refers to an annotation (**`@MyAnnotation`**), so the top-level annotation filter, **`<annotation-list>`** must specify the **`name`** attribute. If the **`<javaclass>`** referred to a class that is annotated, then the top-level annotation filter used would be **`<annotation-type>`**.

3.3.1.2. <xmlfile> Syntax

3.3.1.2.1. Summary

Use the **`<xmlfile>`** element to find information in XML files. For a better understanding of the **`<xmlfile>`** condition, see the XmlFile JavaDoc.

The following is an example of a rule that tests for an XML file.

```
<rule id="UNIQUE_RULE_ID">
    <when>
        <xmlfile matches="/w:web-app/w:resource-ref/w:res-auth[text() =
'Container']">
            <namespace prefix="w"
uri="http://java.sun.com/xml/ns/javaee"/>
        </xmlfile>
    </when>
    <perform>
        <hint title="Title for Hint from XML">
            <message>Container Auth</message>
        </hint>
        <xslt description="Example XSLT Conversion" extension="-converted-
example.xml"
              template="/exampleconversion.xsl"/>
    </perform>
</rule>
```

3.3.1.2.2. Construct an <xmlfile> Element

3.3.1.2.2.1. <xmlfile> Element Attributes

Attribute Name	Type	Description
matches	XPATH	Match on an XML file condition. `matches="/w:web-app/w:resource-ref/w:res-auth[text() = 'Container']"`
xpathResultMatch	XPATH_RESULT_STRING	Return results that match the given regex. `<xmlfile matches="//foo/text()" xpathResultMatch="Text from foo."/>`
as	VARIABLE_NAME	A variable name assigned to the rule so that it can be used as a reference in later processing. See the **`from`** attribute below. `as="MyEjbRule"`
in	PATH_FILTER	Used to filter input files matching this regex (regular expression) naming pattern. Wildcard characters can be used. `in="{*}File1"`
from	VARIABLE_NAME	Begin the search query with the filtered result from a previous search identified by its **`as`** VARIABLE_NAME. `from="MyEjbRule"`
public-id	PUBLIC_ID	The DTD public-id regex. `public-id="public"`

3.3.1.2.2.2. <xmlfile> `matches` Custom Functions

The **`matches`** attribute may use several built-in custom XPath functions, which may have useful side effects, like setting the matched value on the rule variables stack.

Function	Description
`windup:matches()`	Match a XPath expression against a string, possibly containing RHAMT parameterization placeholders. `matches="windup:matches(//foo/@class, '{javaclassname}'"` This will match all **`<foo/>`** elements with a **`class`** attribute and store their value into **`javaclassname`** parameter for each iteration.

3.3.1.2.2.3. <xmlfile> Child Elements

Child Element	Description
<namespace>	The namespace referenced in XML files. This element contains two optional attributes: The **`prefix`** and the **`uri`**. `<namespace prefix="abc" uri="http://maven.apache.org/POM/4.0.0"/>`

3.3.1.3. <project> Syntax

3.3.1.3.1. Summary

Use the **`<project>`** element to query the Maven POM file for the project characteristics. For a better understanding of the **`<project>`** condition, see the JavaDoc for the Project class.

The following is an example of a rule that checks for a JUnit dependency version between 2.0.0.Final and 2.2.0.Final.

```
<rule id="UNIQUE_RULE_ID">
    <when>
        <project>
            <artifact groupId="junit" artifactId="junit"
fromVersion="2.0.0.Final" toVersion="2.2.0.Final"/>
        </project>
    </when>
    <perform>
        <lineitem message="The project uses junit with the version between
2.0.0.Final and 2.2.0.Final"/>
    </perform>
</rule>
```

3.3.1.3.2. Construct a <project> Element

3.3.1.3.2.1. <project> Element Attributes

The **`<project>`** element is used to match against the project's Maven POM file. You can use this condition to query for dependencies of the project. It does not have any attributes itself.

3.3.1.3.2.2. <project> Child Elements

Child Element	Description
<artifact>	Subcondition used within **`<project>`** to query against project dependencies. The **`<artifact>`** element attributes are described below.

3.3.1.3.2.3. <artifact> Element Attributes

Attribute Name	Type	Description
groupId	PROJECT_GROUP_ID	Match on the project **<groupId>** of the dependency.
artifactId	PROJECT_ARTIFACT_ID	Match on the project **<artifactId>** of the dependency.
fromVersion	FROM_VERSION	Specify the lower version bound of the artifact. For example **2.0.0.Final**.
toVersion	TO_VERSION	Specify the upper version bound of the artifact. For example **2.2.0.Final**.

3.3.1.4. <filecontent> Syntax

3.3.1.4.1. Summary

Use the **<filecontent>** element to find strings or text within files, for example, a line in a Properties file. For a better understanding of the **<filecontent>** condition, see the JavaDoc for the FileContent class.

3.3.1.4.2. Construct a <filecontent> Element

3.3.1.4.2.1. <filecontent> Element Attributes

Attribute Name	Type	Description
pattern	String	Match the file contents against the provided parameterized string. This attribute is required.
filename	String	Match the file names against the provided parameterized string.
as	VARIABLE_NAME	A variable name assigned to the rule so that it can be used as a reference in later processing. See the **from** attribute below. `as="MyEjbRule"`
from	VARIABLE_NAME	Begin the search query with the filtered result from a previous search identified by its **as** VARIABLE_NAME. `from="MyEjbRule"`

3.3.1.5. <file> Syntax

3.3.1.5.1. Summary

Use the **<file>** element to find the existence of files with a specific name, for example, an **ibm-webservices-ext.xmi** file. For a better understanding of the **<file>** condition, see the JavaDoc for the File class.

3.3.1.5.2. Construct a <file> Element

3.3.1.5.2.1. <file> Element Attributes

Attribute Name	Type	Description
filename	String	Match the file names against the provided parameterized string. This attribute is required.
as	VARIABLE_NAME	A variable name assigned to the rule so that it can be used as a reference in later processing. See the **from** attribute below. `as="MyEjbRule"`
from	VARIABLE_NAME	Begin the search query with the filtered result from a previous search identified by its **as** VARIABLE_NAME. *Example:* `from="MyEjbRule"`

3.3.1.6. <has-hint> Syntax

3.3.1.6.1. Summary

Use the **<has-hint>** element to test whether a file or line has a hint already associated with it. It is primarily used to prevent firing if a hint already exists, or to implement rules for default execution when no other conditions apply. For a better understanding of the **<has-hint>** condition, see the JavaDoc for the HasHint class.

The following is an example of a rule that checks to see if a hint exists for an IBM JMS destination message, and if not includes it.

```
<rule id="websphere-jms-eap7-03000">
  <when>
    <javaclass references="{package}.{prefix}{type}Message" />
  </when>
  <perform>
    <iteration>
      <when>
        <not>
          <has-hint />
        </not>
      </when>
      <perform>
        <hint title="IBM JMS destination message" effort="1" category-
id="mandatory">
```

```
          <message>
            JMS `{package}.{prefix}{type}Message` messages represent the
actual data passed through JMS destinations. This reference should be
            replaced with the Java EE standard API `javax.jms.
{type}Message`.
          </message>
          <link href="https://docs.oracle.com/javaee/7/tutorial/jms-
concepts003.htm#sthref2271" title="Java EE 7 JMS Tutorial - Message API"
/>
          <tag>jms</tag>
          <tag>websphere</tag>
        </hint>
      </perform>
    </iteration>
  </perform>
  <where param="type">
    <matches pattern="(Text|Stream|Object|Map|Bytes)?" />
  </where>
  <where param="prefix">
    <matches pattern="(JMS|MQe|MQ)" />
  </where>
  <where param="package">
    <matches pattern="com.ibm(\..*)?\.jms" />
  </where>
</rule>
```

3.3.1.6.2. Construct a <has-hint>

The **<has-hint>** element is used to determine if a hint exists for a file or line. It does not have any child elements.

3.3.1.6.2.1. <has-hint> Element Attributes

Attribute Name	Type	Description
message	String	An optional argument allowing you to match the hint against the provided message string.

3.3.1.7. <has-classification> Syntax

3.3.1.7.1. Summary

Use the **<has-classification>** element to test whether a file or line has a classification. It is primarily used to prevent firing if a classification already exists, or to implement rules for default execution when no other conditions apply. For a better understanding of the **<has-classification>** condition, see the JavaDoc for the HasClassification class.

3.3.1.7.2. Construct a <has-classification>

The **has-classification** element is used to determine if a specified classification exists. It does not have any child elements.

3.3.1.7.2.1. <has-classification> Element Attributes

Attribute Name	Type	Description
title	String	An optional title to match the classification against.

3.3.1.8. <graph-query> Syntax

3.3.1.8.1. Summary

Use the **<graph-query>** element to search the generated graph for any elements. This element is primarily used to search for specific archives. For a better understanding of the **<graph-query>** condition, see the JavaDoc for the QueryHandler class.

The following is an example of a rule that tests to determine if any **ehcache** packages are found.

```
<rule id="embedded-cache-libraries-01000">
    <when>
        <graph-query discriminator="JarArchiveModel">
            <property name="fileName"
searchType="regex">.*ehcache.*\.jar$</property>
        </graph-query>
    </when>
    <perform>
        <classification title="Caching - Ehcache embedded library"
category-id="cloud-mandatory" effort="5">
            <description>
                The application embeds an Ehcache library.

                Cloud readiness issue as potential state information that
is not persisted to a backing service.
            </description>
        </classification>
        <technology-tag level="INFORMATIONAL">Ehcache
(embedded)</technology-tag>
    </perform>
</rule>
```

3.3.1.8.2. Construct a <graph-query>

3.3.1.8.2.1. <graph-query> Element Attributes

Attribute Name	Type	Description
discriminator	MODEL_TYPE	The type of model to use for searching. This can be any valid model; however, it is recommended to use the **JarArchiveModel** for examining archives. This attribute is required.

Attribute Name	Type	Description
as	VARIABLE_NAME	A variable name assigned to the rule so that it can be used as a reference in later processing. See the **`from`** attribute below. `as="MyEjbRule"`
from	VARIABLE_NAME	Begin the search query with the filtered result from a previous search identified by its **`as`** VARIABLE_NAME. `from="MyEjbRule"`

3.3.1.8.2.2. <graph-query> Properties

Property Name	Type	Description
name	String	The name of the attribute to match against within the chosen model. When using any file-based models it is recommended to match on **`fileName`**. This attribute is required.
type	property-type	Defines the expected type of property, either **`STRING`** or **`BOOLEAN`**.
searchType	property-search-type	Defines how the condition is matched. If set to **`equals`**, then an exact match must be made. If using **`regex`**, then regular expressions can be used.

3.3.2. Perform Action Syntax

Operations available in the **`perform`** section of the rule include the classification of application resources, in-line hints for migration steps, links to migration information, and project lineitem reporting. Because XML rules are modeled after the Java-based rule add-ons, links to JavaDocs for the related Java classes are provided for a better understanding of how they behave.

The complete XML rule schema is located here: http://windup.jboss.org/schema/windup-jboss-ruleset.xsd.

The following sections describe the more common XML rule perform actions.

- <classification> Syntax
- <link> Syntax
- <hint> Syntax
- <xslt> Syntax
- <lineitem> Syntax

- <iteration> Syntax

3.3.2.1. <classification> Syntax

3.3.2.1.1. Summary

The **<classification>** element is used to identify or classify application resources that match the rule. It provides a title that is displayed in the report, a level of effort, and it can also provide links to additional information about how to migrate this resource classification. For a better understanding of the **<classification>** action, see the JavaDoc for the Classification class.

The following is an example of a rule that classifies a resource as a WebLogic EAR application deployment descriptor file.

```
<rule id="XmlWebLogicRules_10vvyf">
    <when>
        <xmlfile as="default" matches="/*[local-name()='weblogic-
application']"></xmlfile>
    </when>
    <perform>
        <iteration>
            <classification title="Weblogic EAR Application Descriptor"
effort="3"/>
        </iteration>
    </perform>
</rule>
```

3.3.2.1.2. <classification> Element Attributes

Attribute Name	Type	Description
title	STRING	The title given to this resource. This attribute is required. `title="JBoss Seam Components"`
effort	BYTE	The level of effort assigned to this resource. `effort="2"`
category-id	STRING	A reference to a category as defined in **RHAMT_HOME/rules/migration-core/core.windup.categories.xml**. The default categories are **mandatory**, **optional**, **potential**, and **information**. `category-id="mandatory"`
of	VARIABLE_NAME	Create a new classification for the given reference. `of="MySeamRule"`

Attribute Name	Type	Description

3.3.2.1.3. <classification> Child Elements

Child Element	Description
<link>	Provides a link URI and text title for additional information. `<classification title="Websphere Startup Service" effort="4">   <link href="http://docs.oracle.com/javaee/6/api/javax/ejb/Singleton.html" title="EJB3.1 Singleton Bean"/>   <link href="http://docs.oracle.com/javaee/6/api/javax/ejb/Startup.html" title="EJB3.1 Startup Bean"/> </classification>`
<tag>	Provides additional custom information for the classification. `<tag>Seam3</tag>`
<description>	Description of this resource `<description>JBoss Seam components must be replaced</description>`

3.3.2.2. <link> Syntax

3.3.2.2.1. Summary

The **<link>** element is used in classifications or hints to provide links to informational content. For a better understanding of the **<link>** action, see the JavaDoc for the Link class.

The following is an example of a rule that creates links to additional information.

```
<rule id="SeamToCDIRules_2fmb">
    <when>
        <javaclass references="org.jboss.seam.{*}" as="default"/>
    </when>
    <perform>
        <iteration>
            <classification title="SEAM Component" effort="1">
                <link
```

```
href="http://www.seamframework.org/Seam3/Seam2ToSeam3MigrationNotes"
title="Seam 2 to Seam 3 Migration Notes"/>
                <link
href="http://docs.jboss.org/weld/reference/latest/en-US/html/example.html"
title="JSF Web Application Example"/>
                <link
href="http://docs.jboss.org/weld/reference/latest/en-
US/html/contexts.html" title="JBoss Context Documentation"/>
                <link href="http://www.andygibson.net/blog/tutorial/cdi-
conversations-part-2/" title="CDI Conversations Blog Post"/>
            </classification>
        </iteration>
    </perform>
</rule>
```

3.3.2.2.2. <link> Element Attributes

Attribute Name	Type	Description
href	URI	The URI for the referenced link. `href="https://access.redhat.com/articles/1249423"`
title	STRING	A title for the link. `title="Migrate WebLogic Proprietary Servlet Annotations"`

3.3.2.3. <hint> Syntax

3.3.2.3.1. Summary

The **<hint>** element is used to provide a hint or inline information about how to migrate a section of code. For a better understanding of the **<hint>** action, see the JavaDoc for the Hint class.

The following is an example of a rule that creates a hint.

```
<rule id="WebLogicWebServiceRules_8jyqn">
    <when>
        <javaclass
references="weblogic.wsee.connection.transport.http.HttpTransportInfo.setU
sername({*})" as="default">
            <location>METHOD</location>
        </javaclass>
    </when>
    <perform>
        <iteration>
            <hint title="Proprietary web-service" category-id="mandatory"
effort="3">
                <message>Replace proprietary web-service authentication
```

```
with JAX-WS standards.</message>
                <link href="http://java-x.blogspot.com/2009/03/invoking-
web-services-through-proxy.html" title="JAX-WS Proxy Password Example"/>
            </hint>
        </iteration>
    </perform>
</rule>
```

3.3.2.3.2. <hint> Element Attributes

Attribute Name	Type	Description
title	STRING	Title this hint using the specified string. This attribute is required. `title="JBoss Seam Component Hint"`
category-id	STRING	A reference to a category as defined in **`RHAMT_HOME/rules/migration-core/core.windup.categories.xml`**. The default categories are **`mandatory`**, **`optional`**, **`potential`**, and **`information`**. `category-id="mandatory"`
in	VARIABLE_NAME	Create a new Hint in the FileLocationModel resolved by the given variable. `in="Foo"`
effort	BYTE	The level of effort assigned to this resource. `effort="2"`

3.3.2.3.3. <hint> Child Elements

Child Element	Description
<message>	A message describing the migration hint. `<message>EJB 2.0 is deprecated</message>`
<link>	Identify or classify links to informational content. See the section on <link> Syntax for details. `<link href="http://docs.oracle.com/javaee/6/api/" title="Java Platform, Enterprise Edition 6 API Specification" />`

Child Element	Description
<tag>	Define a custom tag for this **hint**. `<tag>Needs review</tag>`

<quickfix>	Contains information to be used by the Eclipse plugin to perform quick fixes when the rule condition is met.
	`<quickfix name="slink-qf" type="REPLACE">`
	`<replacement>h:link</replacement>`
	`<search>s:link</search>`
	`</quickfix>`

3.3.2.4. <xslt> Syntax

3.3.2.4.1. Summary

The **<xslt>** element specifies how to transform an XML file. For a better understanding of the **<xslt>** action, see the JavaDoc for the XSLTTransformation class.

The following is an example of rule that defines an XSLT action.

```
<rule id="XmlWebLogicRules_6bcvk">
    <when>
        <xmlfile as="default" matches="/weblogic-ejb-jar"/>
    </when>
    <perform>
        <iteration>
            <classification title="Weblogic EJB XML" effort="3"/>
            <xslt title="JBoss EJB Descriptor (Windup-Generated)"
template="transformations/xslt/weblogic-ejb-to-jboss.xsl" extension="-
jboss.xml"/>
        </iteration>
    </perform>
</rule>
```

3.3.2.4.2. <xslt> Element Attributes

Attribute Name	Type	Description
title	STRING	Sets the title for this XSLTTransformation in the report. This attribute is required. `title="XSLT Transformed Output"`
of	STRING	Create a new transformation for the given reference. `of="testVariable_instance"`

Attribute Name	Type	Description
extension	STRING	Sets the extension for this XSLTTransformation. This attribute is required. `extension="-result.html"`
template	STRING	Sets the XSL template. This attribute is required. `template="simpleXSLT.xsl"`
effort	BYTE	The level of effort required for the transformation.

3.3.2.4.3. <xslt> Child Elements

Child Element	Description
<xslt-parameter>	Specify XSLTTransformation parameters as property value pairs `<xslt-parameter property="title" value="EJB Transformation"/>`

3.3.2.4.3.1. <lineitem> Syntax

3.3.2.4.4. Summary

The **<lineitem>** element is used to provide general migration requirements for the application, such as the need to replace deprecated libraries or the need to resolve potential class loading issues. This information is displayed on the project or application overview page. For a better understanding of the **<lineitem>** action, see the JavaDoc for the LineItem class.

The following is an example of a rule that creates a lineitem message.

```
<rule id="weblogic_servlet_annotation_1000">
    <when>
        <javaclass references="weblogic.servlet.annotation.WLServlet"
as="default">
            <location>ANNOTATION</location>
        </javaclass>
    </when>
    <perform>
        <hint effort="1">
            <message>Replace the proprietary WebLogic @WLServlet
annotation with the Java EE 6 standard @WebServlet annotation.</message>
            <link href="https://access.redhat.com/articles/1249423"
title="Migrate WebLogic Proprietary Servlet Annotations" />
            <lineitem message="Proprietary WebLogic @WLServlet annotation
found in file."/>
        </hint>
```

```
    </perform>
</rule>
```

3.3.2.4.5. <lineitem> Element Attributes

Attribute Name	Type	Description
message	STRING	A lineitem message. `message="Proprietary code found."`

3.3.2.5. <iteration> Syntax

3.3.2.5.1. Summary

The **<iteration>** element specifies to iterate over an implicit or explicit variable defined within the rule. For a better understanding of the **<iteration>** action, see the JavaDoc for the Iteration class.

The following is an example of a rule that performs an iteration.

```
<rule id="jboss-eap5-xml-19000">
    <when>
        <xmlfile as="jboss-app" matches="/jboss-app"/>
        <xmlfile as="jboss-app-no-DTD" matches="/jboss-app" public-id=""/>
    </when>
    <perform>
        <iteration over="jboss-app">
            <classification title="JBoss application Descriptor"
effort="5"/>
        </iteration>
        <iteration over="jboss-app-no-DTD">
            <classification title="JBoss application descriptor with
missing DTD" effort="5"/>
        </iteration>
        <iteration over="jboss-app-no-DTD">
            <xslt title="JBoss application descriptor - JBoss 5 (Windup-
generated)" template="transformations/xslt/jboss-app-to-jboss5.xsl"
extension="-jboss5.xml"/>
        </iteration>
    </perform>
</rule>
```

3.3.2.5.2. <iteration> Element Attributes

Attribute Name	Type	Description
over	VARIABLE_NAME	Iterate over the condition identified by this VARIABLE_NAME. `over="jboss-app"`

3.3.2.5.3. <iteration> Child Elements

Child Element	Description
<iteration>	Child elements include a **when** condition, along with the actions **iteration**, **classification**, **hint**, **xslt**, **lineitem**, and **otherwise**.

3.3.3. Where Syntax

You can define parameters that specify a matching pattern to be used in other elements of an XML rule. This can help simplify the patterns for complex matching expressions.

Use the **<where>** element to define a parameter. Specify the parameter name using the **param** attribute and supply the pattern using the **<matches>** element. This parameter can then be referenced elsewhere in the rule definition using the syntax ***{PARAM_NAME}***.

The complete XML rule schema is located here: http://windup.jboss.org/schema/windup-jboss-ruleset.xsd.

The following example rule defines a parameter named **subpackage** that specifies a pattern of **(activeio|activemq)**.

```
<rule id="generic-catchall-00600">
  <when>
    <javaclass references="org.apache.{subpackage}.{*}">
    </javaclass>
  </when>
  <perform>
    ...
  </perform>
  <where param="subpackage">
    <matches pattern="(activeio|activemq)" />
  </where>
</rule>
```

The pattern defined by **subpackage** will then be substituted in the **<javaclass> references** attribute. This causes the rule to match on **org.apache.activeio.*** and **org.apache.activemq.*** packages.

3.4. ADD THE RULE TO RED HAT APPLICATION MIGRATION TOOLKIT

A Red Hat Application Migration Toolkit rule is installed by copying the rule to the appropriate RHAMT folder. RHAMT scans for rules, which are files that end with **.windup.xml** or **.rhamt.xml**, in the following locations.

- The directory specified by the **--userRulesDirectory** argument on the RHAMT command line.
- The ***RHAMT_HOME*/rules/** directory. ***RHAMT_HOME*** is the directory where you install and run the Red Hat Application Migration Toolkit executable.
- The **${user.home}/.rhamt/rules/** directory. This directory is created by RHAMT the first time it is executed and contains rules, add-ons, and the RHAMT log.

NOTE

For Windows, this directory would be **\Documents and Settings*USER_NAME*\.rhamt\rules** or **\Users*USER_NAME*\.rhamt\rules**.

CHAPTER 4. TESTING XML RULES

4.1. CREATE A TEST RULE

Once a rule has been created, it is recommended to create a test rule to ensure the functionality meets the desired expectation. Test rules are created using a process similar to Creating XML Rules with the following differences.

- Test rules should be placed in a **`tests/`** directory beneath the rule to be tested.
- Any data, such as test classes, should be placed in a **`data/`** directory beneath the **`tests/`** directory.
- Test rules should use the **`.rhamt.test.xml`** extension.
- These rules use the structure defined in Test XML Rule Structure.

In addition, it is recommended to create a test rule that follows the name of the rule it tests. For instance, if a rule were created with a filename of **`proprietary-rule.rhamt.xml`**, the test rule should be called **`proprietary-rule.rhamt.test.xml`**.

4.1.1. Test XML Rule Structure

All test XML rules are defined as elements within **`ruletests`** which contain one or more **`rulesets`**. For more details, see the RHAMT XML rule schema.

A ruletest is a group of one or more tests that targets a specific area of migration. This is the basic structure of the **`<ruletest>`** element.

- **`<ruletest id="`*`RULE_TOPIC`*`-test">`**: Defines this as a unique RHAMT ruletest and gives it a unique ruletest id.
  - **`<testDataPath>`**: Defines the path to any data, such as classes or files, used for testing.
  - **`<sourceMode>`**: Indicates if the passed in data only contains source files. If an archive, such as an EAR, WAR, or JAR, is in use, then this should be set to **`false`**. Defaults to **`true`**.
  - **`<rulePath>`**: The path to the rule to be tested. This should end in the name of the rule to test.
  - **`<ruleset>`**: Rulesets containing the logic of the test cases. These are identical to the ones defined in Rulesets.

4.1.2. Test XML Rule Syntax

In addition to the tags in XML Rule Syntax, the following **when** conditions are commonly used for creating test rules.

- **`<not>`**
- **`<iterable-filter>`**
- **`<classification-exists>`**

- **<hint-exists>**

In addition to the tags in Perform Action Syntax, the following **perform** conditions are commonly used as actions in test rules.

- **<fail>**

4.1.2.1. <not> Syntax

Summary
The **<not>** element is the standard logical *not* operator, and is commonly used to perform a **<fail>** if the condition is not met.

The following is an example of a test rule that fails if only a specific message exists at the end of the analysis.

```
<ruletest xmlns="http://windup.jboss.org/schema/jboss-ruleset"
          id="proprietary-servlet-test"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
  <testDataPath>data/</testDataPath>
  <rulePath>../proprietary-servlet.rhamt.xml</rulePath>
  <ruleset>
    <rules>
      <rule id="proprietary-servlet-01000-test">
        <when>
          <!--
     The <not> will perform a logical not operator on the elements within.
   -->
          <not>
            <!--
       The defined <iterable-filter> has a size of 1. This rule will only
match on a single instance of the defined hint.
     -->
            <iterable-filter size="1">
              <hint-exists message="Replace the proprietary
@ProprietaryServlet annotation with the Java EE 7 standard @WebServlet
annotation*" />
            </iterable-filter>
          </not>
        </when>
        <!--
   This <perform> element is only executed if the previous <when>
condition is false.
          This ensures that it only executes if there is not a single
instance of the defined hint.
        -->
        <perform>
          <fail message="Hint for @ProprietaryServlet was not found!" />
        </perform>
      </rule>
    </rules>
  </ruleset>
</ruletest>
```

The **<not>** element has no unique attributes or child elements.

4.1.2.2. <iterable-filter> Syntax

Summary
The **<iterable-filter>** element counts the number of times a condition is verified. For additional information, see the IterableFilter class.

The following is an example that looks for four instances of the specified message.

```
<ruletest xmlns="http://windup.jboss.org/schema/jboss-ruleset"
          id="proprietary-servlet-test"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
  <testDataPath>data/</testDataPath>
  <rulePath>../proprietary-servlet.rhamt.xml</rulePath>
  <ruleset>
    <rules>
      <rule id="proprietary-servlet-03000-test">
        <when>
          <!--
     The <not> will perform a logical not operator on the elements within.
   -->
          <not>
     <!--
       The defined <iterable-filter> has a size of 4. This rule will only
match on four instances of the defined hint.
     -->
            <iterable-filter size="4">
              <hint-exists message="Replace the proprietary
@ProprietaryInitParam annotation with the Java EE 7 standard @WebInitParam
annotation*" />
            </iterable-filter>
          </not>
        </when>
 <!--
   This <perform> element is only executed if the previous <when>
condition is false.
   In this configuration, it only executes if there are not four instances
of the defined hint.
 -->
        <perform>
          <fail message="Hint for @ProprietaryInitParam was not found!" />
        </perform>
      </rule>
    </rules>
  </ruleset>
</ruletest>
```

The **<iterable-filter>** element has no unique child elements.

<iterable-filter> Element Attributes

Attribute Name	Type	Description
size	integer	The number of times to be verified.

4.1.2.3. <classification-exists> Syntax

The **<classification-exists>** element determines if a specific classification title has been included in the analysis. For additional information, see the ClassificationExists class.

IMPORTANT

When testing for a message that contains special characters, such as **[** or **'**, you must escape each special character with a backslash (\) to correctly match.

The following is an example that searches for a specific classification title.

```
<ruletest xmlns="http://windup.jboss.org/schema/jboss-ruleset"
          id="proprietary-servlet-test"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
  <testDataPath>data/</testDataPath>
  <rulePath>../weblogic.rhamt.xml</rulePath>
  <ruleset>
    <rules>
      <rule id="weblogic-01000-test">
        <when>
          <!--
     The <not> will perform a logical not operator on the elements within.
   -->
          <not>
     <!--
       The defined <classification-exists> is attempting to match on the
defined title.
       This classification would have been generated by a matching
<classification title="WebLogic scheduled job" .../> rule.
     -->
            <classification-exists classification="WebLogic scheduled
job" />
          </not>
        </when>
 <!--
   This <perform> element is only executed if the previous <when>
condition is false.
   In this configuration, it only executes if there is not a matching
classification.
 -->
        <perform>
          <fail message="Triggerable not found" />
        </perform>
      </rule>
    </rules>
  </ruleset>
```

```
</ruletest>
```

The **<classification-exists>** has no unique child elements.

<classification-exists> Element Attributes

Attribute Name	Type	Description
classification	String	The **<classification> title** to search for.
in	String	An optional argument that restricts matching to files that contain the defined filename.

4.1.2.4. <hint-exists> Syntax

The **<hint-exists>** element determines if a specific hint has been included in the analysis. It searches for any instances of the defined message, and is typically used to search for the beginning or a specific class inside of a **<message>** element. For additional information, see the HintExists class.

IMPORTANT

When testing for a message that contains special characters, such as **[** or **'**, you must escape each special character with a backslash (****) to correctly match.

The following is an example that searches for a specific hint.

```
<ruletest xmlns="http://windup.jboss.org/schema/jboss-ruleset"
          id="proprietary-servlet-test"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
  <testDataPath>data/</testDataPath>
  <rulePath>../weblogic.windup.xml</rulePath>
  <ruleset>
    <rules>
      <rule id="weblogic-eap7-05000-test">
        <when>
          <!--
     The <not> will perform a logical not operator on the elements within.
   -->
          <not>
     <!--
       The defined <hint-exists> is attempting to match on the defined
message.
       This message would have been generated by a matching <message>
element on the <hint> condition.
     -->
            <hint-exists message="Replace with the Java EE standard
method .javax\.transaction\.TransactionManager\.resume\(Transaction tx\)."
/>
          </not>
        </when>
 <!--
```

```
   This <perform> element is only executed if the previous <when>
condition is false.
   In this configuration, it only executes if there is not a matching
hint.
 -->
        <perform>
          <fail message="Note to replace with standard
TransactionManager.resume is missing!" />
        </perform>
      </rule>
    </rules>
  </ruleset>
</ruletest>
```

The **<hint-exists>** element has no unique child elements.

<hint-exists> Element Attributes

Attribute Name	Type	Description
message	String	The **<hint> message** to search for.
in	String	An optional argument that restricts matching to **InLineHintModels** that reference the given filename.

4.1.2.5. <fail> Syntax

The **<fail>** element reports the execution as a failure and displays the associated message. It is commonly used in conjunction with the **<not>** condition to display a message only if the conditions are not met.

An example is included in the **<not>** Syntax section.

The **<fail>** element has no unique child elements.

<fail> Element Attributes

Attribute Name	Type	Description
message	String	The message to be displayed.

4.2. MANUALLY TEST THE XML RULE

Instead of writing a test case, it is possible to simply run the XML rule against your application file by running RHAMT in a terminal.

```
$ RHAMT_HOME/bin/rhamt-cli [--sourceMode] --input INPUT_ARCHIVE_OR_FOLDER
--output OUTPUT_REPORT_DIRECTORY --target TARGET_TECHNOLOGY --packages
PACKAGE_1 PACKAGE_2 PACKAGE_N
```

You should see the following result:

```
Report created: OUTPUT_REPORT_DIRECTORY/index.html
```

```
              Access it at this URL:
file:///OUTPUT_REPORT_DIRECTORY/index.html
```

More examples of how to run RHAMT are located in the Red Hat Application Migration Toolkit *CLI Guide*.

4.3. TEST THE RULES USING JUNIT

Once a test rule has been created, it can be analyzed as part of a JUnit test to confirm that the rule meets all criteria for execution. The **`WindupRulesMultipleTests`** class in the RHAMT rules repository is designed to test multiple rules simultaneously, and provides feedback on any missing requirements.

Prerequisites

- Fork and Clone the RHAMT XML Rules. The location of this repository will be referred to as *RULESETS_REPO*.
- Create a test XML rule.

Create the JUnit Test Configuration

The following instructions detail creating a JUnit test using the Red Hat Developer Studio. When using a different IDE it is recommended to consult your IDE's documentation for instructions on creating a JUnit test.

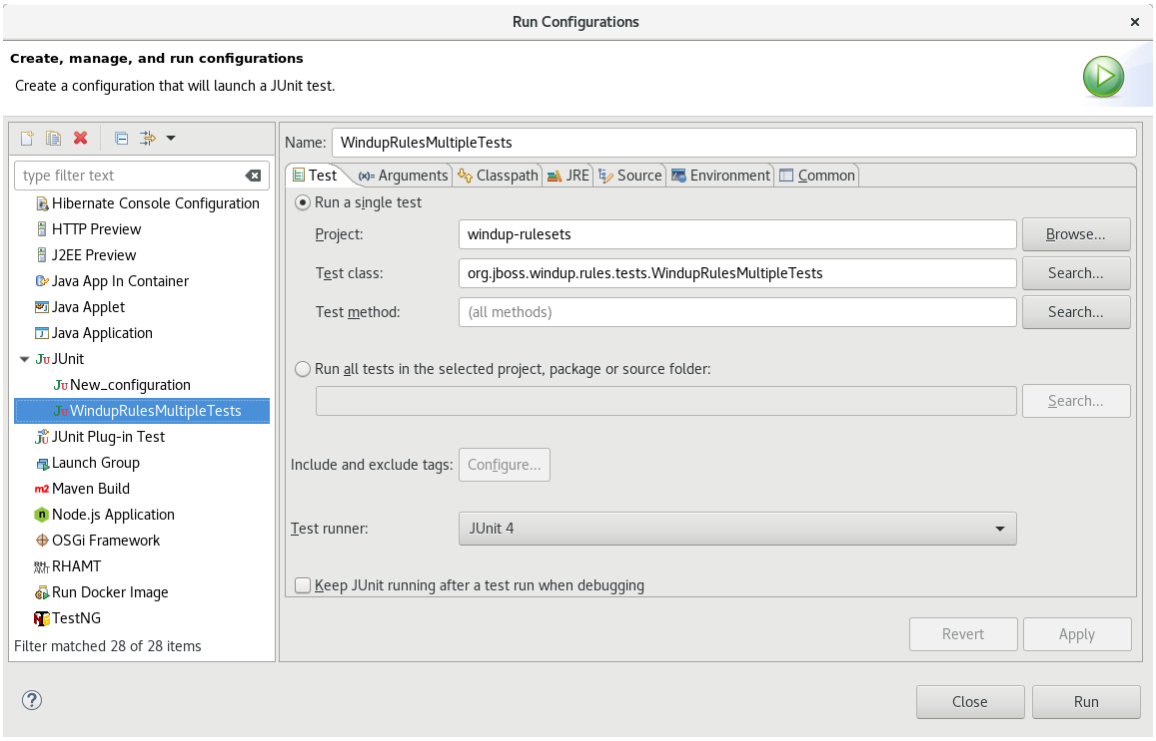
1. Import the RHAMT rulesets repository into your IDE.
2. Copy the custom rules, along with the corresponding tests and data, into ***`/path/to/RULESETS_REPO`*`/rules-reviewed/`*`RULE_NAME/`***. This should create the following directory structure.

   Directory Structure

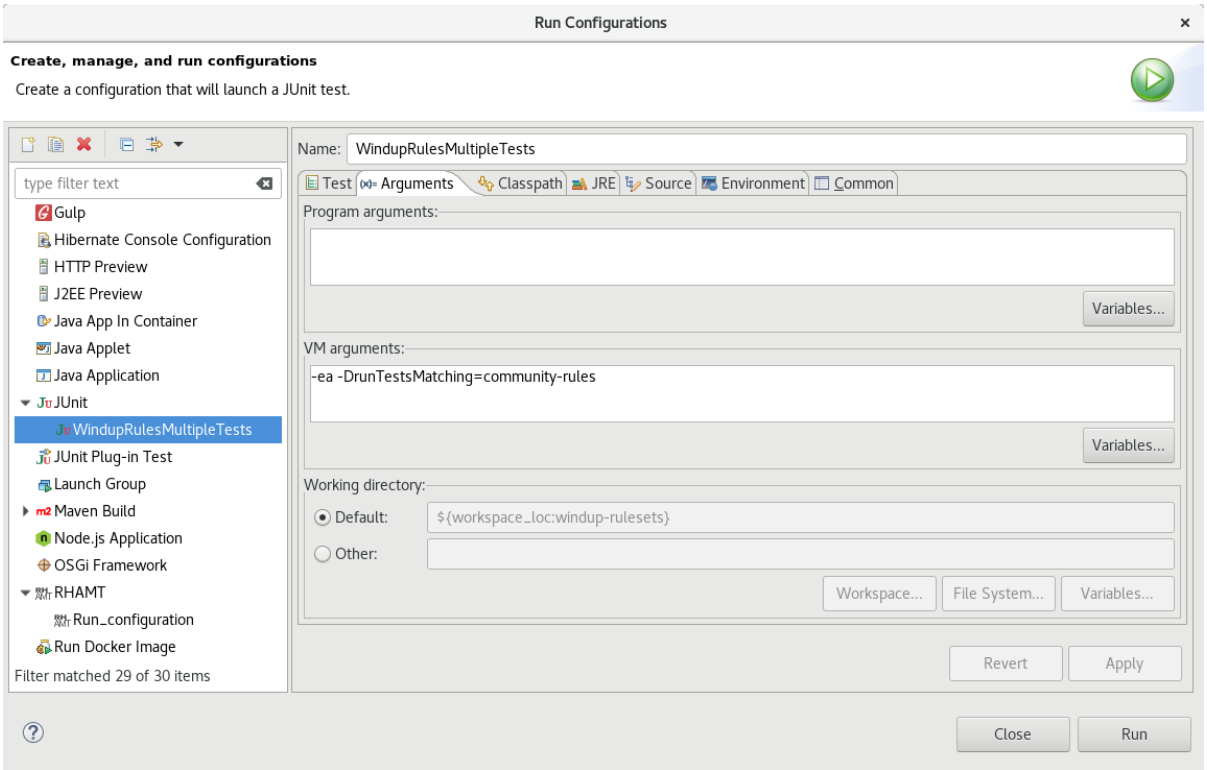
   ```
   ├── rules-reviewed/  (Root directory of the rules found within the
   project)
   │   ├── RULE_NAME/  (Directory to contain the newly developed rule
   and tests)
   │   │   ├── RULE_NAME.rhamt.xml  (Custom rule)
   │   │   ├── tests/  (Directory that contains any test rules and
   data)
   │   │   │   ├── RULE_NAME.rhamt.test.xml (Test rule)
   │   │   │   └── data/  (Optional directory to contain test rule
   data)
   ```

3. Select **Run** from the top menu bar.
4. Select **Run Configurations…** from the drop down that appears.
5. Right-click **JUnit** from the options on the left side and select **New**.
6. Enter the following
   - **Name**: A name for your JUnit test, such as **`WindupRulesMultipleTests`**.
   - **Project**: Ensure this is set to **`windup-rulesets`**.

- **Test class**: Set this to **`org.jboss.windup.rules.tests.WindupRulesMultipleTests`**.



7. Select the **Arguments** tab, and add the **`-DrunTestsMatching=`*`RULE_NAME`*** VM argument. For instance, if your rule name was **`community-rules`**, then you would add **`-DrunTestsMatching=community-rules`** as seen in the following image.



8. Click **Run** in the bottom right corner to begin the test.

Once the execution completes the results will be available for analysis. If all tests passed, then the test rule is correctly formatted; otherwise, it is recommended to address each of the issues raised in the test failures. For additional information on the errors that might occur, see Reported Errors When Running the Validation Report.

4.4. VALIDATION REPORT

Introduced in RHAMT 4.2, the validation report allows rule developers to confirm the test rules function as expected. It produces a report that provides details on each test rule, reporting any failures and where they occur.

Prerequisites

- You must fork and clone the RHAMT XML Rules.
- You must have one or more test XML rules to be validated.

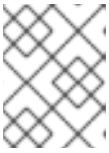
Create the Validation Report

1. Navigate to the local **`windup-rulesets`** repository.
2. Copy your custom rules and tests into the a subdirectory in the **`rules-reviewed`** directory. This should result in
3. Run the following command from the root of the **`windup-rulesets`** repository:

```
$ mvn -Dtest=WindupRulesMultipleTests -
DrunTestsMatching=CUSTOM_RULES clean surefire-report:report
```

For instance, if your rules were stored at the **`/path/to/windup-rulesets/rules-reviewed/myTests/`** directory, then the following command would be used:

```
$ mvn -Dtest=WindupRulesMultipleTests -DrunTestsMatching=myTests
clean surefire-report:report
```



NOTE

Leaving off the **`runTestsMatching`** argument will result in all tests being included in the validation report, and will drastically increase the runtime.

4. The validation report, **`surefire-report.html`**, will be generated in the **`target/site/`** subdirectory of the **`windup-rulesets`** repository.

4.4.1. Understanding the Validation Report

The validation report is split into sections that organize any test failures into the packages and tests executed. This report contains the following sections:

- Summary
- Package List
- Test Cases

Summary
The summary contains the overall number of tests executed and reports the number of errors and failures encountered. It also provides an overall success rate, and the time taken, in seconds, for the report to be generated.

Package List
The package list contains the number of tests executed per package, and reports the number of errors and failures encountered. It also provides a success rate, and the time taken, in seconds, for each package to be analyzed.

It is expected to see a single package, **`org.jboss.windup.rules.tests`**, unless additional test cases have been defined.

Test Cases
This section details each executed test. Each failure includes an optional **Details** section that can be expanded to show the stack trace for the assertion, including a line indicating the human readable source of the error. For additional information on the errors that might occur, see Reported Errors When Running the Validation Report.

4.4.2. Reported Errors When Running the Validation Report

The validation report will report any errors encountered while running the rules and tests. The following includes a list of these errors and how to resolve them.

Error Message	Description	Resolution
No test file matching rule	This error occurs when a rule file exists without a corresponding test file.	Create a test file for the existing rule, as outlined in Testing XML Rules .
Test rule Ids *RULE_NAME* not found!	This error is thrown when a rule exists without a corresponding ruletest.	Create a test for the existing rule, as outlined in Test XML Rule Structure.
XML parse fail on file *FILE_NAME*	The syntax in the XML file is invalid, and unable to be parsed successfully by the rule validator.	Correct the invalid syntax.
Test file path from **`<testDataPath>`** tag has not been found. Expected path to test file is: *RULE_DATA_PATH*	No files are found in the path defined in the **`<testDataPath>`** tag within the test rule.	Create the path defined in the **`<testDataPath>`** tag, and ensure all necessary data files are located within this directory.
The rule with id="*RULE_ID*" has not been executed.	The rule with the provided id has not been executed during this validation.	Ensure that a test data file exists that matches the conditions defined in the specified rule.

CHAPTER 5. OVERRIDING RULES

You can override core rules distributed with RHAMT or even custom-written rules. For example, you might want to change the matching conditions, effort, or hint text for a rule. This is done by making a custom copy of the original rule, marking it as a rule override, and making the necessary adjustments. You can also disable a rule in this same manner.

5.1. OVERRIDE A RULE

You can override a rule using the following steps.

1. Copy the XML file that contains the rule you want to override to the custom rules directory. Custom rules can be placed in either ***RHAMT_HOME*/rules**, **${user.home}/.rhamt/rules/**, or the directory specified by the **--userRulesDirectory** command-line argument.

2. Edit the XML file and only keep the **<rule>** elements for the rules that you want to override. Note that the rules from the original ruleset that are not overridden in the new ruleset will be executed as normal.

3. Ensure that you keep the same rule and ruleset IDs. When you copy the original rule XML, this ensures that the IDs match.

4. Add the **<overrideRules>true</overrideRules>** entry to the ruleset metadata.

5. Make the adjustments to the rule as necessary.
   You can change anything in the rule definition. This rule will override the entire original rule.

The below is an example of an overridden rule. Notice the **<overrideRules>true</overrideRules>** element in the ruleset metadata. The rule and ruleset IDs match the original. This rule override changed the effort needed from a **1** to a **3**.

```
<?xml version="1.0"?>
<ruleset id="weblogic"
    xmlns="http://windup.jboss.org/schema/jboss-ruleset"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
    <metadata>
        ...
        <overrideRules>true</overrideRules>
    </metadata>
    <rules>
        <rule id="weblogic-02000"
xmlns="http://windup.jboss.org/schema/jboss-ruleset">
            <when>
                <javaclass references="weblogic.utils.StringUtils.{*}"/>
            </when>
            <perform>
                <hint effort="3" category-id="mandatory" title="WebLogic
StringUtils Usage">
                    <message>Replace with the StringUtils class from
Apache Commons.</message>
                    <link
href="https://commons.apache.org/proper/commons-lang/" title="Apache
Commons Lang"/>
```

```
                    <tag>weblogic</tag>
                </hint>
            </perform>
        </rule>
    </rules>
</ruleset>
```

When you run RHAMT, this rule will now be used in place of the original rule with the matching rule ID. You can verify that the new rule was used by viewing the contents of the Rule Provider Executions Overview.

5.2. DISABLE A RULE

To disable a rule, follow the same steps as above to override a rule, except provide an empty **<rule>** element.

```
<?xml version="1.0"?>
<ruleset id="weblogic"
    xmlns="http://windup.jboss.org/schema/jboss-ruleset"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
    <metadata>
        ...
        <overrideRules>true</overrideRules>
    </metadata>
    <rules>
        <rule id="weblogic-02000"
xmlns="http://windup.jboss.org/schema/jboss-ruleset">
            <!-- Empty so that the original rule is disabled -->
        </rule>
    </rules>
</ruleset>
```

CHAPTER 6. USING CUSTOM RULE CATEGORIES

A new category field was added in RHAMT 3.0 that replaces the concept of the severity field in RHAMT rules. The default categories are the same that were available in the severity field: **`mandatory`**, **`optional`**, and **`potential`**. Additional categories, such as **`information`**, have also been included in subsequent RHAMT releases. You can now add your own custom rule categories and assign RHAMT rules to them.

IMPORTANT

Although RHAMT can still process rules that use the legacy **`severity`** field, it is recommended to update your custom rules to use the new **`category-id`** field.

Add a Custom Category

1. Edit the rule category file, which is located at ***`RHAMT_HOME`*`/rules/migration-core/core.windup.categories.xml`**.
2. Add a new **`<category>`** element and fill in the following fields:
   - **`id`**: The ID that RHAMT rules will use to reference the category.
   - **`priority`**: The sorting priority compared to other categories. The category with the lowest value is displayed first.
   - **`name`**: The display name of the category.
   - **`description`**: The description of how the category is intended to be used.

   Example Custom Rule Category

```
<?xml version="1.0"?>
<categories>
    ...
    <category id="custom-category" priority="20000">
        <name>Custom Category</name>
        <description>This is a custom category.</description>
    </category>
</categories>
```

This category is now ready to be referenced by RHAMT rules.

Assign a Rule to a Custom Category

In your RHAMT rule, use the custom category's **`id`** value in the rule's **`category-id`** field.

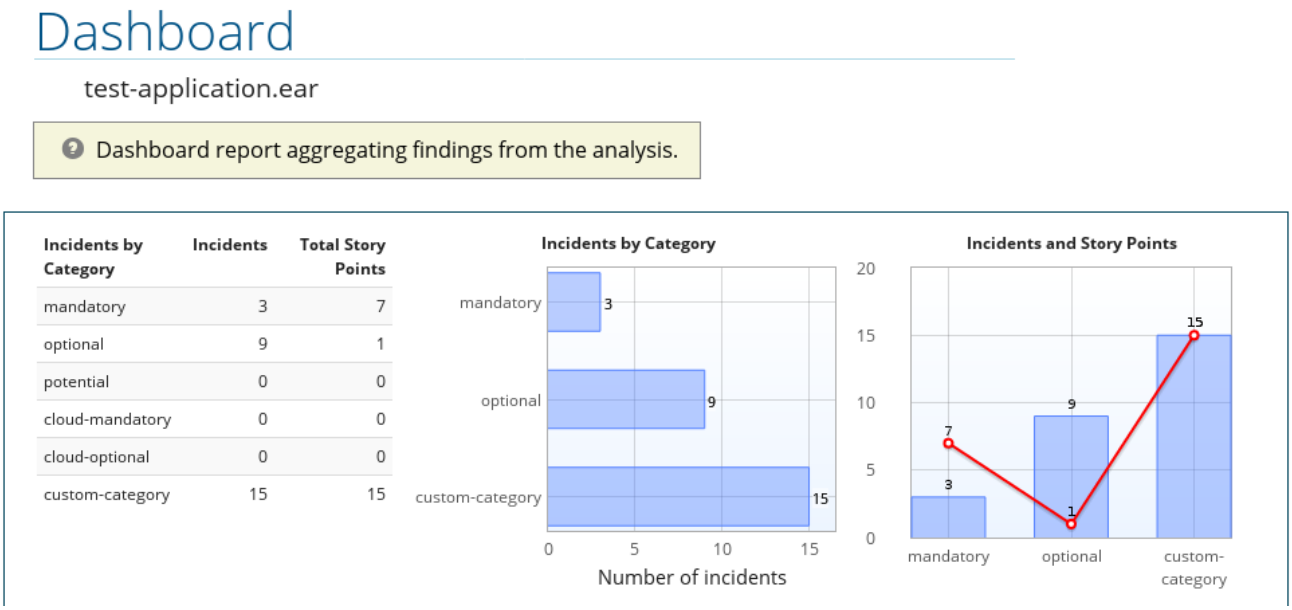
Example Rule Using a Custom Rule Category

```
<rule id="rule-id">
    <when>
        ...
    </when>
    <perform>
        <hint title="Rule Title" effort="1" category-id="custom-
category">
```

```
            <message>Hint message.</message>
        </hint>
    </perform>
</rule>
```

Now when you run RHAMT and this rule's condition is met, incidents identified by this rule will use your custom category. The custom category can be seen in the RHAMT report in places such as the dashboard and Issues report.

Figure 6.1. Custom Category on the Dashboard



APPENDIX A. REFERENCE MATERIAL

A.1. RULE STORY POINTS

A.1.1. What are Story Points?

Story points are an abstract metric commonly used in Agile software development to estimate the *level of effort* needed to implement a feature or change.

Red Hat Application Migration Toolkit uses story points to express the level of effort needed to migrate particular application constructs, and the application as a whole. It does not necessarily translate to man-hours, but the value should be consistent across tasks.

A.1.2. How Story Points are Estimated in Rules

Estimating the level of effort for the story points for a rule can be tricky. The following are the general guidelines RHAMT uses when estimating the level of effort required for a rule.

Level of Effort	Story Points	Description
Information	0	An informational warning with very low or no priority for migration.
Trivial	1	The migration is a trivial change or a simple library swap with no or minimal API changes.
Complex	3	The changes required for the migration task are complex, but have a documented solution.
Redesign	5	The migration task requires a redesign or a complete library change, with significant API changes.
Rearchitecture	7	The migration requires a complete rearchitecture of the component or subsystem.
Unknown	13	The migration solution is not known and may need a complete rewrite.

A.1.3. Task Category

In addition to the level of effort, you can categorize migration tasks to indicate the severity of the task. The following categories are used to group issues to help prioritize the migration effort.

Mandatory

The task must be completed for a successful migration. If the changes are not made, the resulting application will not build or run successfully. Examples include replacement of proprietary APIs that are not supported in the target platform.

Optional

If the migration task is not completed, the application should work, but the results may not be optimal. If the change is not made at the time of migration, it is recommended to put it on the schedule soon after your migration is completed. An example of this would be the upgrade of EJB 2.x code to EJB 3.

Potential

The task should be examined during the migration process, but there is not enough detailed information to determine if the task is mandatory for the migration to succeed. An example of this would be migrating a third-party proprietary type where there is no directly compatible type.

Information

The task is included to inform you of the existence of certain files. These may need to be examined or modified as part of the modernization effort, but changes are typically not required. An example of this would be the presence of a logging dependency or a Maven **`pom.xml`**.

For more information on categorizing tasks, see Using Custom Rule Categories.

A.2. ADDITIONAL RESOURCES

A.2.1. Review the Existing RHAMT XML Rules

RHAMT XML-based rules are located on GitHub at the following location: https://github.com/windup/windup-rulesets/tree/master/rules-reviewed.

See Fork and Clone the RHAMT XML Rules for instructions on obtaining these rules on your local machine.

Rules are grouped by target platform and function. When you create a new rule, it is helpful to find a rule that is similar to the one you need and use it as a starting template.

New rules are continually added, so it is a good idea to check back frequently to review the updates.

A.2.1.1. Fork and Clone the Red Hat Application Migration Toolkit XML Rules

The Red Hat Application Migration Toolkit **`windup-rulesets`** repository provides provide working examples of how to create custom Java-based rule add-ons and XML rules. You can use them as a starting point for creating your own custom rules.

You must have the **`git`** client installed on your machine.

1. Click the **`Fork`** link on the Red Hat Application Migration Toolkit Rulesets GitHub page to create the project in your own Git. The forked GitHub repository URL created by the fork should look like this: **`https://github.com/`*`YOUR_USER_NAME`*`/windup-rulesets.git`**.

2. Clone your Red Hat Application Migration Toolkit rulesets repository to your local file system:

```
$ git clone https://github.com/YOUR_USER_NAME/windup-rulesets.git
```

3. This creates and populates a **`windup-rulesets`** directory on your local file system. Navigate to the newly created directory, for example

```
$ cd windup-rulesets/
```

4. If you want to be able to retrieve the latest code updates, add the remote **`upstream`** repository so you can fetch any changes to the original forked repository.

```
$ git remote add upstream https://github.com/windup/windup-
rulesets.git
```

5. Get the latest files from the **`upstream`** repository.

```
$ git fetch upstream
```

A.2.2. Important Links

- RHAMT Javadoc: http://windup.github.io/windup/docs/latest/javadoc
- RHAMT forums: https://developer.jboss.org/en/windup
- RHAMT JIRA issue trackers
  - Core RHAMT: https://issues.jboss.org/browse/WINDUP
  - RHAMT Rules: https://issues.jboss.org/browse/WINDUPRULE
- RHAMT mailing list: jboss-migration-feedback@redhat.com
- RHAMT on Twitter: @JBossWindup
- RHAMT IRC channel: Server FreeNode (**`irc.freenode.net`**), channel **`#windup`** (transcripts)

Revised on 2018-10-31 18:24:33 UTC