Windup Rules Development Guide

Table of Contents

1. Introduction

This guide is for engineers, consultants, and others who plan to create custom XML-based rules for Windup.

1.1. What is Windup?

1.1.1. Overview

Windup is an extensible and customizable rule-based tool that helps simplify migration of Java applications.

Windup can be run as a standalone application or as a plug-in to Red Hat JBoss Developer Studio. Running from a Forge environment, it examines application artifacts, including project source directories and applications archives, then

produces an HTML report highlighting areas that need changes. Windup can be used to migrate Java applications from previous versions of *Red Hat JBoss Enterprise Application Platform* or from other containers, such as *Oracle WebLogic Server* or *IBM® WebSphere® Application Server*.

1.1.2. How Does Windup Simplify Migration?

Windup looks for common resources and highlights technologies and known "trouble spots" when migrating applications. The goal is to provide a high level view into the technologies used by the application and provide a detailed report organizations can use to estimate, document, and migrate enterprise applications to Java EE and JBoss EAP.

1.2. Features of Windup

Shared Data Model

Windup creates a shared data model graph that provides the following benefits.

- It enables complex rule interaction, allowing rules to pass findings to other rules.
- It enables 3rd-party plug-ins to interact with other plugins, rules and reports.
- The data graph organizes the findings to be searchable and queryable.

Extensibility

Windup can be extended by developers, users, and 3rd-parties.

- It provides a plug-in API to inject other applications into Windup.
- It enables 3rd-parties to create simple POJO plug-ins that can interact with the data graph.
- Means we don't have to invent everything. Users with domain knowledge can implement their own rules.

Better Rules

Windup provides more powerful and complex rules.

- XML-based rules are simple to write and and easy to implement.
- Java-based rule add-ons are based on OCPsoft Rewrite and provide greater flexibility and power creating when rules.
- Rules can now be nested to handle more complex situations. This means you can nest simple statements rather than use complex XPATH or REGEX expressions. *Rules can be linked using and/or statements

Automation

Windup has the ability to automate some of the migration processes.

- Windup is integrated with Forge 2, meaning it can generate projects, libraries, and configuration files.
- Rules can create Forge inputs and put them into the data graph.
- During the automation phase, the data graph inputs can be processed to generate a new project.

Work Estimation

Estimates for the level of effort is based on the skills required and the classification of migration work needed.

- Lift and Shift - The code or file is standards-based and can be ported to the new environment with no changes.
- Mapped - There is a standard mapping algorithm to port the code or file to the new environment.
- Custom – The code or file must be rewritten or modified to work in the new environment.

Better Reporting	Windup reports are now targeted for specific audiences. • IT Management - Applications are ranked by cost of migration. • Project Management - Reports detail the type of work and estimation of effort to complete the tasks. • Developers - An Eclipse plug-in provides hints and suggested code changes within the IDE.

1.3. System Requirements

Software

- Java Platform, Enterprise Edition 7
- Windup is tested on Linux, Mac OS X, and Windows. Other Operating Systems with Java 7 support should work equally well.

Hardware

- A minimum of 4 GB RAM. For better performance, a 4-core processor with 8 GB RAM is recommended. This allows 3 - 4 GB RAM for use by the JVM.
- A minimum of 4 GB of free disk space. A fast disk, especially a Solid State Drive (SSD), will improve performance.

1.4. About the WINDUP_HOME Variable

This documentation uses the **WINDUP_HOME** *replaceable* value to denote the path to the Windup distribution. When you encounter this value in the documentation, be sure to replace it with the actual path to your Windup installation.

- If you download and install the latest distribution of Windup from the JBoss Nexus repository, WINDUP_HOME refers to the windup-distribution-2.2.0-Final folder extracted from the downloaded ZIP file.
- If you build Windup from GitHub source, WINDUP_HOME refers to the windup-distribution-2.2.0-Final folder extracted from the Windup source dist/target/windup-distribution-2.2.0-Final.zip file.

2. Get Started

2.1. Install Windup

1. If you installed previous versions of Windup, delete the `${user.home}/.windup/` directory. Otherwise you may see errors like the following when you execute Windup.

```
Command: windup-migrate-app was not found
```

2. Download the latest Windup ZIP distribution.
3. Extract the ZIP file in to a directory of your choice.

2.2. Write Your First Rule

2.2.1. Overview

This topic guides you through the process of creating and testing your first rule.

Windup XML-base rules use the following familiar rule pattern:

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

As you create your first rule, refer to theRules Schema for valid XML syntax.

For more information about XML rule construction, seeCreate a Basic XML Rule.

2.2.2. Rule Example Description

In previous releases of Red Hat JBoss Enterprise Application, you could ensure application class namespace isolation during deployment by defining a `<class-loading>` element in the `jboss-web.xml` file. Due to the change in JBoss EAP 6 to use modular class loading, this element is no longer necessary, and can result in `ParseError` and `XMLStreamException` errors in the server log. This issue is

described in the JBoss EAP Migration Guide.

In this example, you 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.

2.2.3. Create the Directory Structure for the Rule

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

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

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

2.2.4. Create Data to Test the Rule

1. Use your favorite editor or IDE to create a `jboss-web.xml` file in the `~/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>
```

2.2.5. Create the Rule

1. Use your favorite editor or IDE to create an XML file in the `~/migration-rules/rules/` subdirectory named `JBoss5-web-class-loading.windup.xml`. Copy in the following content.

```
<?xml version="1.0"?>
<ruleset xmlns="http://windup.jboss.org/v1/xml" id="UNIQUE_RULESET_ID">
    <rules>
        <rule id="UNIQUE_RULE_ID">
```

```
            <when>
                <!-- Test for a condition here -->
            </when>
            <perform>
                <!-- Perform an action -->
            </perform>
        </rule>
    </rules>
</ruleset>
```

> **NOTE** Windup identifies files with the `.windup.xml` extension as XML-based rules, so make 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 the file name: "JBoss5-web-class-loading"
   - Replace the UNIQUE_RULE_ID with the ruleset ID appended with '_001': "JBoss5-web-class-loading_001"
3. Complete the `when` condition.
   - Because this rule finds `jboss-web.xml` files containing the `class-loading` element, we use `xmlfile` 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>
     ```

4. Complete the `perform` action for this rule.
   - Provide an informative message.
   - Provide a link to documentation that describes the migration details.
   - Assign a level of effort of "1" to this task.

     ```
     <perform>
         <iteration>
     ```

```
        <classification classification="JBoss Web Application Descriptor"
effort="0"/>
        <hint message="The class-loading element is no longer valid in the
jboss-web.xml file." effort="1">
            <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_Loa
ding_in_JBoss_Enterprise_Application_Platform_6" description="Create or Modify
Files That Control Class Loading in JBoss EAP 6"/>
        </hint>
    </iteration>
</perform>
```

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

```
<?xml version="1.0"?>
<ruleset xmlns="http://windup.jboss.org/v1/xml" id="JBoss5-web-class-loading">
    <rules>
        <rule id="JBoss5-web-class-loading_001">
            <when>
                <xmlfile matches="jboss-web/class-loading" />
            </when>
            <perform>
                <iteration>
                    <classification classification="JBoss Web Application
Descriptor" effort="0"/>
                    <hint message="The class-loading element is no longer valid
in the jboss-web.xml file." effort="1">
                        <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_Loadi
ng_in_JBoss_Enterprise_Application_Platform_6" description="Create or Modify
Files That Control Class Loading in JBoss EAP 6"/>
                    </hint>
                </iteration>
            </perform>
        </rule>
     </rules>
</ruleset>
```

2.2.6. Install the Rule

A Windup rule is installed simply by copying the rule to the appropriate folder.

Copy the `JBoss5-web-class-loading.windup.xml` file to your `${user.home}/.windup/rules/` directory.

```
For Linux or Mac: ~/.windup/rules/
```

```
For Windows: "\Documents and Settings\USER_NAME\.windup\rules\" or
"\Users\USER_NAME\.windup\rules\"
```

2.2.7. Test the Rule

1. Open a terminal and navigate to the WINDUP_HOME/bin directory
2. Type the following command to start Windup:

```
For Linux:    windup/bin $ ./windup
For Windows:  C:\WINDUP_HOME\bin> windup
```

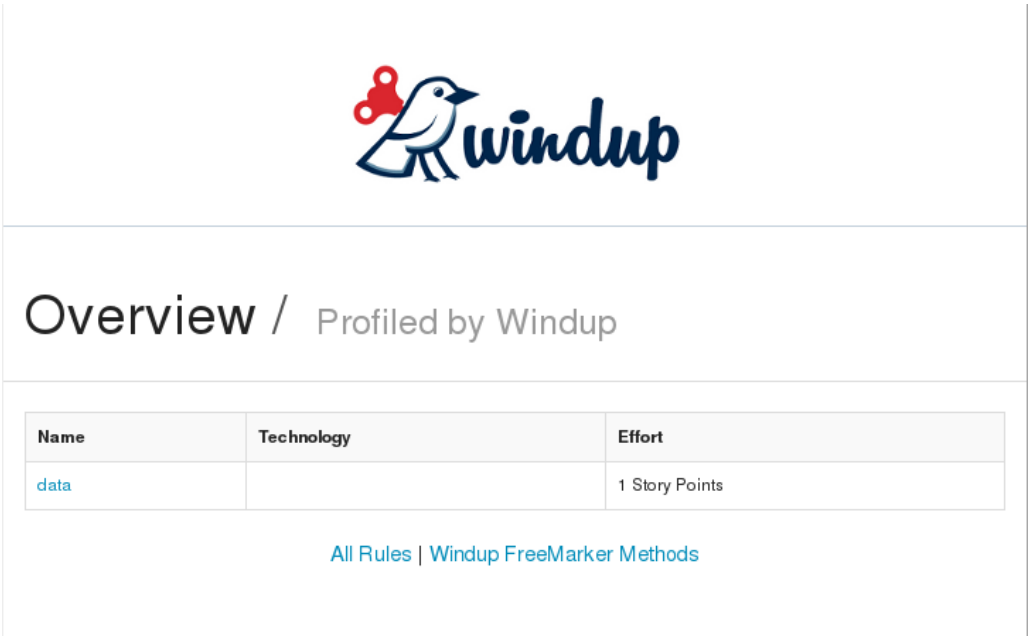
3. Execute the `windup-migrate-app` command, passing the test data file as the input parameter.

```
windup-migrate-app sourceMode true --input ~/migration-rules/data --output
~/migration-rules/reports
```

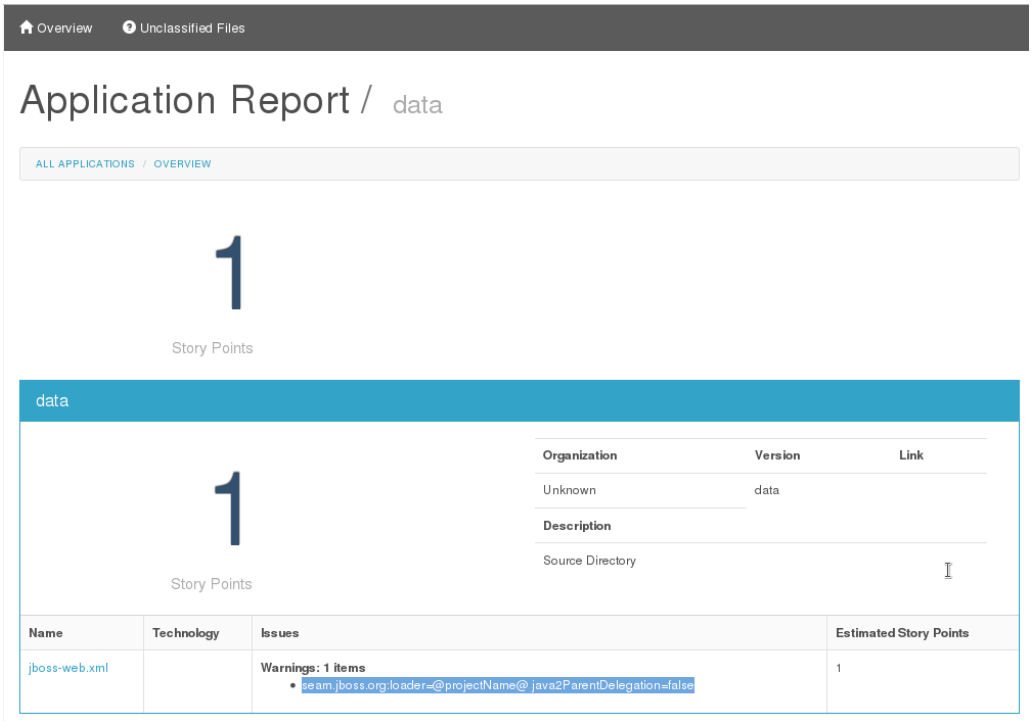
4. You should see this result.

```
***SUCCESS*** Windup report created: /home/your-username/migration-
rules/reports/index.html
              Access it at this URL: file:///home/your-username/migration-
rules/reports/index.html
```

5. Access the report at ~/migration-rules/reports/index.html to make sure it provides the expected results.
    - The *Overview* page displays the **Name** of the input folder, "data", along with the expected **Effort** of "1 Story Points".

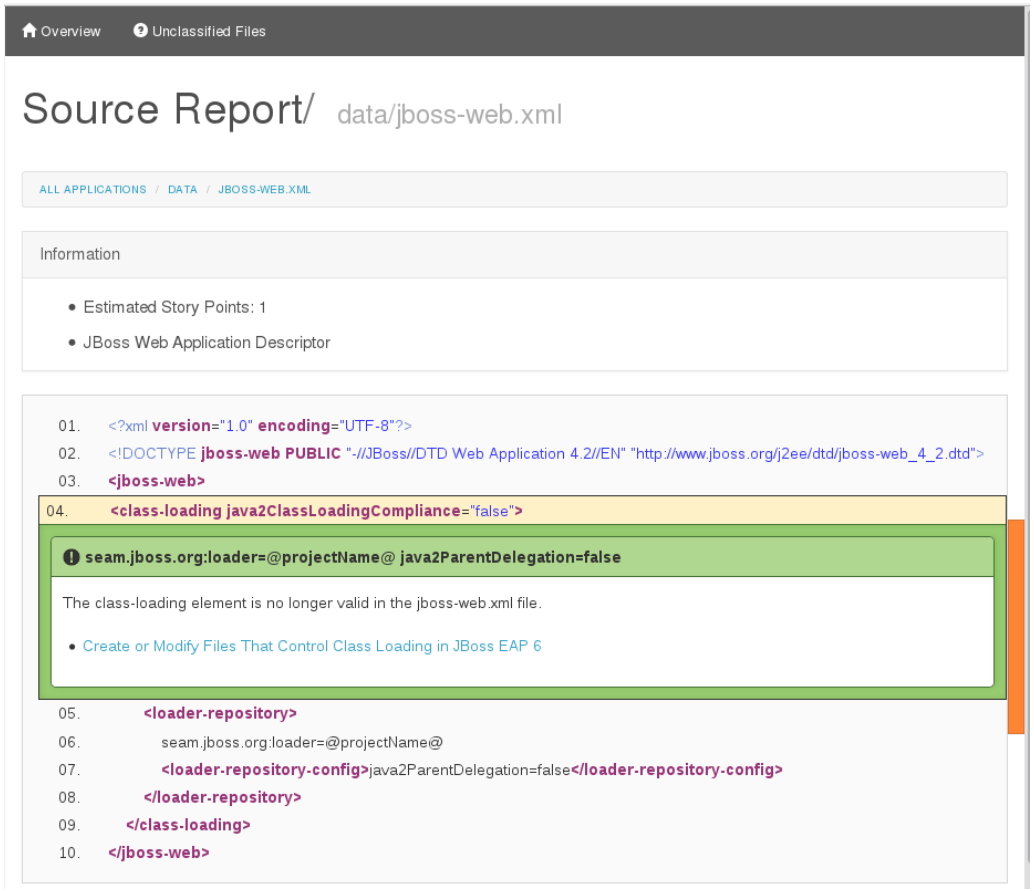


- Drill down into the *Application Report* detail by clicking on the "data" link. This report displays the **Name** of the file, "jboss-web.xml", the warning "seam.jboss.org:loader=@projectName@ java2ParentDelegation=false" in the **Issues** column, and displays "1" *Estimated Story Points", as expected.



- Drill down into *Source Report* by clicking on the "jboss-web.xml" file link. This report provides information about the file and summarizes the story points. It also highlights the `<class-loading>` line in the `jboss-web.xml` file, provides the message "The class-loading element is no longer valid in the jboss-web.xml file.", and provides a link to the Create or Modify Files That Control Class Loading in JBoss EAP 6 topic in the JBoss EAP 6 Migration Guide. Click on the

link to make sure the link to the documentation is valid.



3. Create and Test XML Rules

3.1. Difference Between XML-based and Java-based Rules

3.1.1. Summary

As mentioned before, Windup provides a core and a default set of rules to analyze and report on migration of application code. Windup also allows you to write your own custom rules. These rules can be written using either XML or Java. Rules written using XML are referred to as *XML-based* rules. Rules written using the Java API are referred to as *Java-based* rule add-ons. Both *XML-based* and *Java-based* rule add-ons can be used to inspect (classify) and report on Java source, XML, properties, archives, and other types of files,

3.1.2. Which one to choose?

XML-based rules provide a quick, simple way to create rules to analyze Java, XML, and properties files. If you simply need to highlight a specific section of Java code or

XML file content and provide migration hints for it, creation of *XML-based* rules is the recommended approach. Creation of custom *XML-based* rules is covered in the *Windup Rules Development Guide*.

Java-based rule add-ons provide the ability to create very complex rules, manipulate the shared data model graph, and customize the resulting reports. If you need to test or perform complex conditions and operations or want to manipulate the shared data model graph, create custom reports, or extend the functionality in any other way beyond what the *XML-based* rules provide, you must create *Java-based* rules. Creation of custom *Java-based* rules is covered in the *Windup Core Development Guide*.

3.1.3. Pros and Cons of XML-based Rules

Pros:

- XML rules are fairly easy to write and require less code.
- XML rules are not compiled so you do not need to configure Maven to build from source.
- XML rules are simple to deploy. You simply drop the rule into the appropriate path and Windup automatically scans the new rule.

Cons:

- XML rules only support a simple subset of conditions and operations.
- XML rules do not provide for direct custom graph data manipulation.
- XML rules do not support the ability to create custom reports.

3.1.4. Pros and Cons - Java

Pros:

- Java rule add-ons allow you to write custom conditions and operations and provide a lot of flexibility.
- Java rule add-ons allow you to access and manipulate the shared data model graph and to customize reports.

- You can set breakpoints and test Java rule add-ons using a debugger.
- IDEs provide code completion for the Windup API.

Cons:

- You must configure Maven to compile Java rule add-ons.
- Java rule add-ons that are not included in the Windup core code base must be a full Forge add-on.
- Java rule add-ons require that you write Java code.
- Writing Java rule add-ons can be complex and require knowledge of Windup internals.

3.1.5. Examples

The following is an example of a rule written in XML that classifies Java code:

```
<?xml version="1.0"?>
<ruleset xmlns="http://windup.jboss.org/v1/xml" id="EjbRules">
    <rules>
        <rule id="EjbRules_2fmb">
            <when>
                <javaclass references="javax.persistence.Entity" as="default">
                    <location>TYPE</location>
                </javaclass>
            </when>
            <perform>
                <iteration>
                    <classification classification="JPA Entity" effort="0"/>
                </iteration>
            </perform>
        </rule>
    </rules>
</ruleset>
```

The following is an example of a rule written in Java that classifies Java code:

```
/**
 * Scans for classes with EJB related annotations, and adds EJB related metadata for
these.
 */
public class DiscoverEjbAnnotationsRuleProvider extends AbstractRuleProvider
{
```

```
    @Override
    public Configuration getConfiguration(GraphContext context) {
        return ConfigurationBuilder.begin()
        .addRule()
        .when(JavaClass.references("javax.ejb.
{annotationType}").at(TypeReferenceLocation.ANNOTATION))
        .perform(new AbstractIterationOperation<JavaTypeReferenceModel>()
        {
            public void perform(GraphRewrite event, EvaluationContext context,
JavaTypeReferenceModel payload)
            {
                extractEJBMetadata(event, payload);
            };
        })
        .where("annotationType").matches("Stateless|Stateful")
        .withId(ruleIDPrefix + "_StatelessAndStatefulRule")
        .addRule()

.when(JavaClass.references("javax.ejb.MessageDriven").at(TypeReferenceLocation.ANNOTA
TION))
        .perform(new AbstractIterationOperation<JavaTypeReferenceModel>() {
            @Override
            public void perform(GraphRewrite event, EvaluationContext context,
JavaTypeReferenceModel payload) {
                extractMessageDrivenMetadata(event, payload);
            }
        })
        .withId(ruleIDPrefix + "_MessageDrivenRule")
        .addRule()

.when(JavaClass.references("javax.persistence.Entity").at(TypeReferenceLocation.ANNOT
ATION).as(ENTITY_ANNOTATIONS)

.or(JavaClass.references("javax.persistence.Table").at(TypeReferenceLocation.ANNOTATI
ON).as(TABLE_ANNOTATIONS_LIST)))
        .perform(Iteration.over(ENTITY_ANNOTATIONS).perform(new
AbstractIterationOperation<JavaTypeReferenceModel>() {
            @Override public void perform(GraphRewrite event, EvaluationContext
context, JavaTypeReferenceModel payload) {
                extractEntityBeanMetadata(event, payload);
            }
        }).endIteration())
        .withId(ruleIDPrefix + "_EntityBeanRule");
    }
    ...
}
```

3.1.6. Quick Comparison Summary

Requirement	XML Rule	Java Rule Add-on
Easy to write?	Yes	Depends on the complexity of the rule
Requires that you configure Maven?	No	Yes
Requires that you compile the rule?	No	Yes
Simple deployment?	No	Yes
Supports custom reports?	No	Yes
Ability to create complex conditions and operations?	No	Yes
Ability to directly manipulate the graph data?	No	Yes

3.2. XML-Rule-Construction

This section describes the basic construction of XML rules. All XML rules are defined as elements within rulesets. Rulesets also define the phase in which the rules are run.

3.2.1. Ruleset Element

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

- <**ruleset**>: This element defines this as a Windup ruleset.
  - <**rules**>: This element contains the individual rules.
    - <**rule**/>: This element is defines the rule.

      One or more rules can be defined for a ruleset. See Rule Elements in the following section for details on how to define `<rule>` elements.

- **<rule/>**

  Defines an individual rule.

- **<file-mapping/>**

  Map an extension to a graph type

- **<package-mapping/>**

  Maps from a package pattern (regular expression) to a organization name.

- **</rules>**

- **</ruleset>**:

3.2.2. Rule Elements

Rule elements follow the familiar construct:

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

The following section describes the more commonly used elements in a `<rule.`.

- **<when>**: This element defines the condition or conditions to match on.
  - **<javaclass/>**: Match on a Java class.

    This element can have the following attributes:

Attribute	Description
references="CLASS_NAME"	Match on the fully qualified class name. Wildcards can be specified using `{*}` syntax. For example: "org.apache.commons.{*}"
in="FILE_NAME"	The file name
as="VARIABLE_NAME"	An optional variable name that can be used in later processing.

This element can contain the following elements:

Element	Description
<location>	The location where the reference was found in a Java source file, for example, in the IMPORT, ANNOTATION, METHOD, VARIABLE_DECLARATION. You can specify multiple locations. See the TypeReferenceLocation Javadoc for the full list of valid values.

For more information, see the JavaClass JavaDoc.

- <**xmlfile**/>: Match on an XML file. The following table lists some of the most commonly used attributes and elements.

  This element can have the following attributes:

Attribute	Description
matches="XPATH_PATH"	Match on the XPath, for example: "/w:web-app/w:resource-ref/w:res-auth[text() = 'Container']". Wildcards can be specified using `{*}` syntax.
in="FILE_NAME"	The file name
as="VARIABLE_NAME"	An optional variable name that can be used in later processing.

  This element can contain the following elements:

Element	Description
<namespace>	The namespace prefix and URI

  For more information, see the XmlFile JavaDoc.

- <**project**/>: Match on a project. The following table lists some of the most

commonly used attributes and elements. For more information, see the Project JavaDoc.

- **</when>**
- **<perform>**: This element is invoked when the condition is met.
  - **<hint>**: This child element of **perform** is used to create a `hint` This element can have the following attributes:

Attribute	Description
effort=LEVEL_OF_EFFORT	A level of effort assigned to this rule
message="MESSAGE"	The message ??
in="VARIABLE_NAME"	A variable to use for substitution.
title="Title"	The title ??.

    This element can contain the following elements:

Element	Description
<message>	The message to display in the report.
<link>	An HREF link and description for further information.

  - **<xslt>**: This specifies how to transform the the specified XML file
  - **<log>**: This child element of **perform** is used to log a message. It takes the attribute **message** to define the text message.
- **</perform>**
- The **<otherwise/>** element is invoked when the condition is not met.

3.2.3. Predefined Rules

Windup provides some predefined rules for more common migration requirements, for example, mapping files from the source platform to target platform. The following is an example of the predefined "XmlFileMappings" rule.

```
<?xml version="1.0"?>
<ruleset xmlns="http://windup.jboss.org/v1/xml" id="XmlFileMappings">
  <rules>
    <file-mapping from=".*\.tld$" to="XmlFileModel"/>
    <file-mapping from=".*\.bpel$" to="XmlFileModel"/>
    <file-mapping from=".*\.wsdl$" to="XmlFileModel"/>
    <file-mapping from=".*\.wsdd$" to="XmlFileModel"/>
    <file-mapping from=".*\.bpelex$" to="XmlFileModel"/>
    <file-mapping from=".*\.mon$" to="XmlFileModel"/>
    <file-mapping from=".*\.xmi$" to="XmlFileModel"/>
    <file-mapping from=".*\.export$" to="XmlFileModel"/>
    <file-mapping from=".*\.import$" to="XmlFileModel"/>
    <file-mapping from=".*\.bcfg$" to="XmlFileModel"/>
    <file-mapping from=".*\.map$" to="XmlFileModel"/>
    <file-mapping from=".*\.brg$" to="XmlFileModel"/>
    <file-mapping from=".*\.brgt$" to="XmlFileModel"/>
    <file-mapping from=".*\.ruleset$" to="XmlFileModel"/>
    <file-mapping from=".*\.module$" to="XmlFileModel"/>
    <file-mapping from=".*\.modulex$" to="XmlFileModel"/>
    <file-mapping from=".*\.composite$" to="XmlFileModel"/>
    <file-mapping from=".*\.requirements$" to="XmlFileModel"/>
  </rules>
</ruleset>
```

3.3. Create a Basic XML Rule

You can create a Windup rule using Java, XML, or Groovy. This topic describes how to create a rule using XML.

3.3.1. Prerequisites

- You should have already installed Windup.
- Before you begin, you may also want to be familiar with the following documentation:
  - Windup rules are based on the ocpsoft **rewrite** project. You can find more information about ocpsoft **rewrite** here: http://ocpsoft.org/rewrite/
  - The JavaDoc for the Windup API is located here: http://windup.github.io/windup/docs/javadoc/latest/
  - The XML rule schema is located here: https://github.com/windup/windup/blob/master/config-xml/rule-schema.xsd.

3.3.2. File Naming Convention for XML Rules

You must name the file containing an XML rule with the `.windup.xml` extension. Windup identifies files with this extension as XML-base rules, so be sure to use this naming convention, otherwise the rule will not be evaluated!

3.3.3. Basic XML Rule Template

XML rules consist of *conditions* and *actions* and follow the familiar "if/then/else" construct:

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

The following is the basic syntax for XML rules.

```
<?xml version="1.0"?>
<ruleset xmlns="http://windup.jboss.org/v1/xml" id="XmlFileMappings">
    <phase>
        <!-- The phase in which to run the rules -->
    </phase>
    <rules>
        <rule>
            <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>
```

3.3.4. Create the Rule When Condition

The syntax is dependent on whether you are creating a rule to evaluate Java class, an XML file, a project, or file content and is described in more detail here: [XML Rule - When Condition Syntax]

3.3.5. Create the Rule Perform Action

Operations allowed 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. The syntax is described in detail here: XML Rule - Perform Action Syntax.

3.4. Test multiple XML rulesets with .xsd schema

To test all the xml rules with the .xsd schema, follow this steps

- Download https://github.com/amouat/xsd-validator and unzip it into your selected folder (referred to as $XSD_VAL)
- prepare ruleset directory as $RULESET_DIRECTORY and assume the xsd file is placed in $XSD_SCHEMA path
- run the command

```
find $RULESET_DIRECTORY -type f -iname "*windup.xml" -exec $XSD_VAL./xsdv.sh
$XSD_SCHEMA {} \;
```

3.5. XML Rule - When Condition Syntax

Conditions allowed in the `when` portion of a rule must extendGraphOperation 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: https://github.com/windup/windup/blob/master/config-xml/rule-schema.xsd.

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

- javaclass Syntax
- xmlfile Syntax
- project Syntax
- filecontent Syntax

3.5.1. javaclass Syntax

Summary

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 a `javaclass`.

```
<rule>
    <when>
        <javaclass references="org.jboss.ws.api.annotation.WebContext"
in="org/jboss/{*}" as="webcontextclasses">
            <location>IMPORT</location>
            <location>TYPE</location>
        </javaclass>
    </when>
    <perform>
        <hint message="WebContext is deprecated." effort="0"/>
    </perform>
</rule>
```

Construct a javaclass Element

javaclass Element Attributes

references="CLASS_NAME"

The package or class name to match on. Wildcard characters can be used.

Example:

```
references="org.apache.commons.{*}"
```

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.

Example:

```
as="MyEjbRule"
```

in="PATH_FILTER"

Used to filter input files matching this regex (regular expression) naming pattern. Wildcard characters can be used.

Example:

```
in="{*}File1"
```

from="VARIABLE_NAME"

Begin the search query with the filtered result from a previous search identified by its `as` VARIABLE_NAME.

Example:

```
from="MyEjbRule"
```

JavaClass Element Child Elements

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.

Example:

```
<location>IMPORT</location>
```

3.5.2. xmlfile Syntax

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 `xmlfile`.

```
<rule>
    <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>
```

Construct an xmlfile Element

xmlfile Element: Attributes

matches="XPATH"

Match on an XML file condition.

Example:

```
matches="/w:web-app/w:resource-ref/w:res-auth[text() = 'Container']"
```

xpathResultMatch="XPATH_RESULT_STRING"

Return results that match the given regex.

Example:

```
xpathResultMatch=""
```

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.

Example:

```
as="MyEjbRule"
```

in="PATH_FILTER"

Used to filter input files matching this regex (regular expression) naming pattern. Wildcard characters can be used.

Example:

```
in="{*}File1"
```

from="VARIABLE_NAME"

Begin the search query with the filtered result from a previous search identified by its `as` VARIABLE_NAME.

Example:

```
from="MyEjbRule"
```

public-id="PUBLIC_ID"

The DTD public-id regex.

Example:

```
public-id="public"
```

xmlfile Element: Child Elements

namespace

The namespace to referenced in XML files. This element contains 2 attributes: The `prefix` and the `uri`.

Example:

```
<namespace prefix="abc" uri="http://maven.apache.org/POM/4.0.0"/>
```

3.5.3. project Syntax

Summary

Use the `project` element to query for the project charateristics. 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 a rule is dependent on the junit in the version between 2.0.0.Final and 2.2.0.Final.

```
<rule>
    <when>
        <project>
            <artifact groupId="junit" artifactId="junit" from="2.0.0.Final"
to="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>
```

Construct a project Element

project Element Attributes

The `project` element is used to match against the project as a whole. You can use this condition to query for dependencies of the project. It does not have any attributes itself.

project Element Child Elements

artifact

Subcondition used within `project` to query against project dependencies. This element contains the following attributes:

- 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`

3.5.4. filecontent Syntax

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.

3.6. XML Rule - 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: https://github.com/windup/windup/blob/master/config-xml/rule-schema.xsd.

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

- Classification Syntax
- Link Syntax
- Hint Syntax
- XSLT Syntax
- Lineitem Syntax
- Iteration Syntax

3.6.1. Classification Syntax

Summary

The `classification` element is used to identify or classify application resources. It provides a level of effort and 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.

Example:

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

`classification` Element: Attributes

classification="STRING"

Classify the resource as the specified string.

Example:

```
classification="JBoss Seam Components"
```

effort="NUMBER"

The level of effort assigned to this resource.

Example:

```
effort="2"
```

`classification` Element: Child Elements

xref

Provides a link URI and text description for additional information.

Example:

```
<classification classification="Websphere Startup Service" effort="4">
   <link href="http://docs.oracle.com/javaee/6/api/javax/ejb/Singleton.html"
description="EJB3.1 Singleton Bean"/>
```

```
   <link href="http://docs.oracle.com/javaee/6/api/javax/ejb/Startup.html"
description="EJB3.1 Startup Bean"/>
</classification>
```

3.6.2. Link Syntax

Summary

The `link` element is used in classifications or hints to identify or classify 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.

Example:

```
<rule id="SeamToCDIRules_2fmb">
    <when>
        <javaclass references="org.jboss.seam.{*}" as="default"/>
    </when>
    <perform>
        <iteration>
            <classification classification="SEAM Component" effort="1">
                <link
href="http://www.seamframework.org/Seam3/Seam2ToSeam3MigrationNotes"
description="Seam 2 to Seam 3 Migration Notes"/>
                <link href="http://docs.jboss.org/weld/reference/latest/en-
US/html/example.html" description="JSF Web Application Example"/>
                <link href="http://docs.jboss.org/weld/reference/latest/en-
US/html/contexts.html" description="JBoss Context Documentation"/>
                <link href="http://www.andygibson.net/blog/tutorial/cdi-
conversations-part-2/" description="CDI Conversations Blog Post"/>
            </classification>
        </iteration>
    </perform>
</rule>
```

`link` Element: Attributes

href="URI"

The URI for the referenced link/.

Example:

```
href="https://access.redhat.com/articles/1249423"
```

description="URI_DESCRIPTION"

A description for the link.

Example:

```
description="Migrate WebLogic Proprietary Servlet Annotations"
```

3.6.3. Hint Syntax

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.

Example:

```
<rule id="WebLogicWebServiceRules_8jyqn">
    <when>
        <javaclass
references="weblogic.wsee.connection.transport.http.HttpTransportInfo.setUsername({*}
)" as="default">
            <location>METHOD</location>
        </javaclass>
    </when>
    <perform>
        <iteration>
            <hint message="Replace proprietary web-service authentication with JAX-
WS standards." effort="0">
                <link href="http://java-x.blogspot.com/2009/03/invoking-web-services-
through-proxy.html" description="JAX-WS Proxy Password Example"/>
            </hint>
        </iteration>
    </perform>
</rule>
```

`hint` Element: Attributes

message="MESSAGE"

A message describing the migration hint

Example:

```
message="See this KnowledgeBase article on the Customer Portal: <some-url>"
```

effort="NUMBER"

The level of effort assigned to this resource.

Example:

```
effort="2"
```

`hint` Element: Child Elements

xref

Identify or classify links to informational content. See the section on Link Syntax for details.

Example:

```
link href="http://java-x.blogspot.com/2009/03/invoking-web-services-through-
proxy.html" description="JAX-WS Proxy Password Example"/>
```

3.6.4. XSLT Syntax

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.

Example:

```
<rule id="XmlWebLogicRules_6bcvk">
    <when>
        <xmlfile as="default" matches="/weblogic-ejb-jar"/>
    </when>
    <perform>
        <iteration>
```

```
            <classification classification="Weblogic EJB XML" effort="3"/>
            <xslt description="JBoss EJB Descriptor (Windup-Generated)"
template="transformations/xslt/weblogic-ejb-to-jboss.xsl" extension="-jboss.xml"/>
        </iteration>
    </perform>
</rule>
```

`xslt` Element: Attributes

of="STRING"

Create a new transformation for the given reference.

Example:

```
of="testVariable_instance"
```

description="String"

Sets the description of this XSLTTransformation.

Example:

```
description="XSLT Transformed Output"
```

extension="String"

Sets the extension for this XSLTTransformation.

Example:

```
extension="-result.html"
```

template=String

Sets the XSL template.

Example:

```
template="simpleXSLT.xsl"
```

`xslt` Element: Child Elements

xslt-parameter=Map<String,String>

Specify XSLTTransformation parameters as property value pairs

Example:

```
<xslt-parameter property="title" value="EJB Transformation"/>
```

3.6.5. Lineitem Syntax

Summary

The `lineitem` element is used to provide line item information about a hint 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.

Example:

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

`lineitem` Element: Attributes

message="MESSAGE"

A lineitem message

Example:

```
message="Proprietary code found."
```

3.6.6. Iteration Syntax

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 preforms an iteration.

Example:

```
<rule id="XmlWebLogicRules_14wscy">
    <when>
        <xmlfile as="1" matches="/wl:weblogic-webservices | /wl9:weblogic-
webservices">
            <namespace prefix="wl9" uri="http://www.bea.com/ns/weblogic/90"/>
            <namespace prefix="wl" uri="http://www.bea.com/ns/weblogic/weblogic-
webservices"/>
        </xmlfile>
        <xmlfile as="2" matches="//wl:webservice-type | //wl9:webservice-type"
from="1">
            <namespace prefix="wl9" uri="http://www.bea.com/ns/weblogic/90"/>
            <namespace prefix="wl" uri="http://www.bea.com/ns/weblogic/weblogic-
webservices"/>
        </xmlfile>
    </when>
    <perform>
        <iteration over="1">
            <classification classification="Weblogic Webservice Descriptor"
effort="0"/>
        </iteration>
        <iteration over="2">
            <hint message="Webservice Type" effort="0"/>
        </iteration>
    </perform>
</rule>
```

`iteration` Element: Attributes

over="VARIABLE_NAME"

Iterate over the condition identified by this VARIABLE_NAME.

Example:

```
over="2"
```

`iteration` Element: Child Elements

iteration child elements include a `when` condition, along with the actions `iteration`, `classification`, `hint`, `xslt`, `lineitem`, and `otherwise`.

3.7. Test an XML Rule in Windup

3.7.1. Add the Rule to Windup

A Windup rule is installed simply by copying the rule to the appropriate Windup folder. Windup scans for rules, which are files that end with either `*.windup.groovy` or `.windup.xml`, in the following locations:

- In the directory specified on the `windup-migrate-app` using the `--userRulesDirectory` argument.
- In the WINDUP_HOME/rules/ directory.

  WINDUP_HOME is the directory where you install and run the Windup executable.
- In the `${user.home}/.windup/rules/` directory.

  The `${user.home}/.windup` is a directory created by Windup at first run and contains rules, add-ons, and the Windup log.

```
For Linux or Mac:   ~/.windup/rules/
For Windows: "\Documents and Settings\USER_NAME\.windup\rules\"  -or-
"\Users\USER_NAME\.windup\rules\"
```

3.7.2. Test the XML Rule

1. If you have not started Windup, follow the instructions to Execute Windup.
2. Test the XML rule against your application file by running the `windup-migrate-app` command in the Windup console prompt.

The command uses this syntax:

```
windup-migrate-app [--sourceMode true] --input INPUT_ARCHIVE_OR_FOLDER --output
OUTPUT_REPORT_DIRECTORY --packages PACKAGE_1 PACKAGE_2 PACKAGE_N
```

You should see the following result:

```
***SUCCESS*** Windup report created: OUTPUT_REPORT_DIRECTORY/index.html
```

3.7.3. Additional Resources

- For more information and examples of how to run Windup, see:Execute Windup
- Working examples of XML-based rules can be found on GitHub in theWindup source code GitHub repository and the Windup quickstarts GitHub repository or latest release ZIP download.

---

4. Windup Processing

4.1. Execute Windup

4.1.1. Prerequisites

Before you begin, you must gather the following information.

1. The fully qualified path of the application archive or folder you plan to migrate.
2. The fully qualified path to a folder that will contain the resulting report information.
   - If the folder does not exist, it is created by Windup.
   - If the folder exists, you are prompted with the message:

     ```
     Overwrite all contents of <OUTPUT_DIRECTORY> (anything already in the directory
     will be deleted)? [y/N]
     ```

     Choose "y" if you want Windup to delete and recreate the directory.
   - If you are confident you want to overwrite the output directory, you can specify `--overwrite` on the command line to automatically delete and

recreate the directory.

> **NOTE** Be careful not to specify a directory that contains important information!

3. You must also provide a list of the application packages to be evaluated.
   - In most cases, you are interested only in evaluating the custom application class packages and not the standard Java EE or 3rd party packages. For example, if the *MyCustomApp* application uses the package `com.mycustomapp`, you provide that package using the `--packages` argument on the command line. It is not necessary to provide the standard Java EE packages, like `java.util` or `javax.ejb`.
   - While you can provide package names for standard Java EE 3rd party software like `org.apache`, it is usually best not to include them as they should not impact the migration effort.
   - If you omit the `--packages` argument, every package in the application is scanned, resulting in very slow performance. It is best to provide the argument with one or more packages.

4.1.2. Start Windup

For information about the use of WINDUP_HOME in the instructions below, see [About the WINDUP_HOME Variable](#).

1. Open a terminal and navigate to the WINDUP_HOME/bin directory
2. Type the following command to start Windup:

```
For Linux:    windup/bin $ ./windup
For Windows:  C:\WINDUP_HOME\bin> windup
```

3. You are presented with the following prompt.

```
Using Windup at WINDUP_HOME


 _       ___                __
| |     / (_)___  ____/ /_  ______
| | /| / / / __ \/ __  / / / / __ \
| |/ |/ / / / / / /_/ / /_/ / /_/ /
```

```
|__/|__/_/_/ /_/\__,_/\__,_/ .___/
                          /_/

JBoss Windup, version [ 2.2.0.Final ] - JBoss, by Red Hat, Inc. [
http://windup.jboss.org ]

[windup-distribution-2.2.0.Final]$
```

4.1.3. Run Windup

1. The command to run Windup is `windup-migrate-app`.
2. This command can take arbitrary options processed by different addons. The list of options in the core Windup distribution can be found in Javadoc. Most commonly used command line arguments are:

   --input INPUT_ARCHIVE_OR_FOLDER

   This is the fully qualified application archive or source path.

   --output OUTPUT_REPORT_DIRECTORY

   The fully qualified path to the folder that will contain the the report information produced by Windup.

   > NOTE
   >
   > If the **OUTPUT_REPORT_DIRECTORY** directory exists and you do not specify the `--overwrite` argument, you are prompted to overwrite the contents. If you respond "y", it is deleted and recreated by Windup, so be careful not to specify an output directory that contains important information!

   --overwrite (optional)

   Specify this optional argument only if you are certain you want to force Windup to delete the existing **OUTPUT_REPORT_DIRECTORY**. The default value is `false`.

   --userRulesDirectory

   Points to a directory to load XML rules from. (Search pattern: *.windup.groovy and *.windup.xml)

   --packages PACKAGE_1, PACKAGE_2, PACKAGE_N (optional)

   This is a comma-delimited list of the packages to be evaluated by Windup.

--excludePackages PACKAGE_1, PACKAGE_2, PACKAGE_N (optional)

This is a comma-delimited list of the packages to be excluded by Windup.

--source-mode true (optional)

This argument is optional and is only required if the application to be evaluated contains source files rather than compiled binaries. The default value is `false`.

3. To evaluate an application archive, use the following syntax:

```
windup-migrate-app --input INPUT_ARCHIVE_OR_FOLDER --output
OUTPUT_REPORT_DIRECTORY --packages PACKAGE_1 PACKAGE_2 PACKAGE_N
```

   To run Windup against application source code, you must add the `--sourceMode true` argument:

```
windup-migrate-app --sourceMode true --input INPUT_ARCHIVE_OR_FOLDER --output
OUTPUT_REPORT_DIRECTORY --packages PACKAGE_1 PACKAGE_2 PACKAGE_N
```

   See Windup Command Examples below for concrete examples of commands that use source code directories and archives located in the Windup GitHub repository.

4. You should see the following result upon completion of the command:

```
***SUCCESS*** Windup execution successful!
```

5. To exit Windup, type:

```
exit
```

6. Open the `OUTPUT_REPORT_DIRECTORY/index.html` file in a browser to access the report. The following subdirectories in the `OUTPUT_REPORT_DIRECTORY` contain the supporting information for the report:

```
OUTPUT_REPORT_DIRECTORY/
    graph/
    renderedGraph/
    reports/
    stats/
```

```
    index.html
```

7. For details on how to evaluate the report data, see Review the Report.

4.1.4. Run Windup in Batch Mode

Windup can be also executed in batch mode within a shell or batch script using the `--evaluate` argument as follows.

1. Open a terminal and navigate to the WINDUP_HOME directory.
2. Type the following command to run Windup in batch mode:

```
For Linux:     $ bin/windup --evaluate "windup-migrate-app --input INPUT_ARCHIVE
--output OUTPUT_REPORT --packages PACKAGE_1 PACKAGE_2 PACKAGE_N"
For Windows:   > bin\windup.bat --evaluate "windup-migrate-app --input
INPUT_ARCHIVE --output OUTPUT_REPORT --packages PACKAGE_1 PACKAGE_2 PACKAGE_N"
```

4.1.5. Windup Command Line Help

To see the complete list of available arguments for the `windup-migrate-app` command, execute the following command in the Windup prompt:

```
man windup-migrate-app
```

4.1.6. Command Examples

The following command examples report against applications located in the Windup source test-files directory.

Source Code Example

The following command runs against the seam-booking-5.2 application source code. It evaluates all `org.jboss.seam` packages and creates a folder named 'seam-booking-report' in the `/home/username/windup-reports/` directory to contain the reporting output.

```
windup-migrate-app --sourceMode true --input /home/username/windup-source/test-
files/seam-booking-5.2/ --output /home/username/windup-reports/seam-booking-report --
packages org.jboss.seam
```

Archive Example

The following command runs against the jee-example-app-1.0.0.ear EAR archive. It evaluates all `com.acme` and `org.apache` packages and creates a folder named 'jee-example-app-1.0.0.ear-report' in the `/home/username/windup-reports/` directory to contain the reporting output.

```
windup-migrate-app --input /home/username/windup-source/test-files/jee-example-app-1.0.0.ear/ --output /home/username/windup-reports/jee-example-app-1.0.0.ear-report --packages com.acme org.apache
```

Windup Batch Example

The following Windup batch command runs against the jee-example-app-1.0.0.ear EAR archive. It evaluates all `com.acme` and `org.apache` packages and creates a folder named 'jee-example-app-1.0.0.ear-report' in the `/home/username/windup-reports/` directory to contain the reporting output.

```
For Linux: $ bin/windup --evaluate "windup-migrate-app --input /home/username/windup-source/test-files/jee-example-app-1.0.0.ear/ --output /home/username/windup-reports/jee-example-app-1.0.0.ear-report --packages com.acme org.apache"
For Windows: > bin\windup.bat --evaluate "windup-migrate-app --input \windup-source\test-files\jee-example-app-1.0.0.ear --output \windup-reports\jee-example-app-1.0.0.ear-report --packages com.acme org.apache
```

Windup Quickstart Examples

For more concrete examples, see the Windup quickstarts located on GitHub here: https://github.com/windup/windup-quickstarts. If you prefer, you can download the latest release ZIP or TAR distribution of the quickstarts.

The quickstarts provide examples of Java-based and XML-based rules you can run and test using Windup. The README instructions provide a step-by-step guide to run the quickstart example. You can also look through the code examples and use them as a starting point for creating your own rules.

4.2. Review the Report

4.2.1. About the Report

When you execute Windup, the report is generated in the `OUTPUT_REPORT_DIRECTORY` you specify for the `--output` argument in the command line. This output directory contains the following files and subdirectories:

- `index.html` : This is the landing page for the report.
- `archives/` : Contains the archives extracted from the application
- `graph/` : Contains binary graph database files
- `reports/` : This directory contains the generated HTML report files
- `stats/` : Contains Windup performance statistics

The examples below use the test-files/jee-example-app-1.0.0.ear located in the Windup GitHub source repository for input and specify the `com.acme` and `org.apache` package name prefixes to scan. For example:

```
windup-migrate-app --input /home/username/windup-source/test-files/jee-example-app-
1.0.0.ear/ --output /home/username/windup-reports/jee-example-app-1.0.0.ear-report --
packages com.acme org.apache
```

4.2.2. Open the Report

Use your favorite browser to open the `index.html` file located in the output report directory. You should see something like the following:

Click on the link under the **Name** column to view the Windup application report page.

4.2.3. Report Sections

Application Report Page

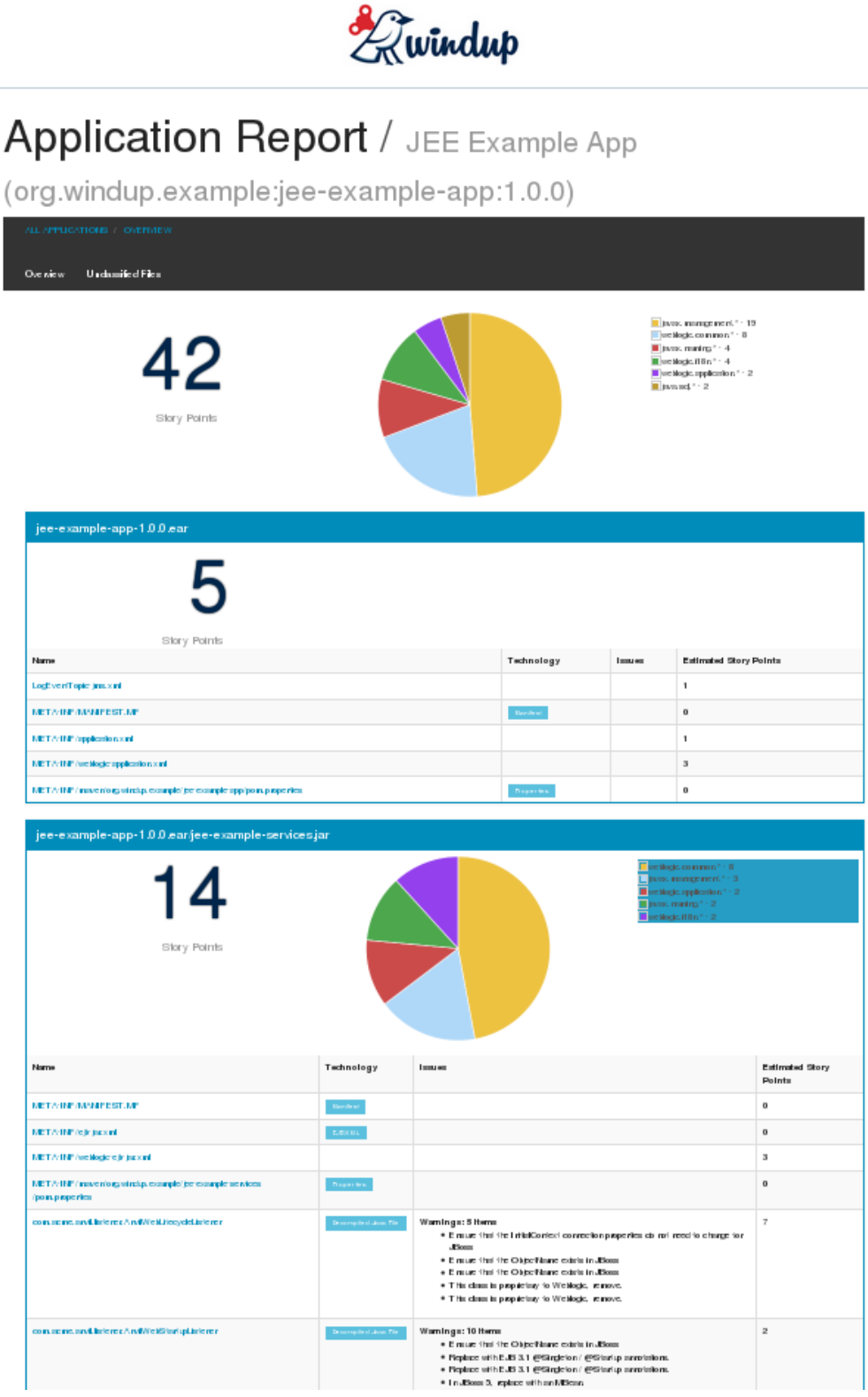
The first section of the application report page summarizes the migration effort. It

displays the following information both graphically and in list form by application artifact for each file that is analyzed.

- The file name
- The file type
- A list of issues, if any, that were found in the file
- The estimated total *Story Points* to migrate the file. A *Story Point* is a term commonly used in Scrum Agile software development methodology to estimate the level of effort needed to implement a feature or change. It does not necessarily translate to man-hours, but the value should be consistent across tasks.

In the following Windup Application Report example, the migration of the **JEE Example App** EAR is assigned a total of 42 story points. A pie chart shows the breakdown of story points by package. This is followed by a section for each of the archives contained in the EAR. It provides the total of the story points assigned to the archive and lists the files contained in archive along with the warnings and story point assigned to each file.

Example of a Windup Application Report



Archive Analysis Sections

Each archive summary begins with a total of the story points assigned to its migration, followed by a table detailing the changes required for each file in the archive. The report contains the following columns.

Name

The name of the file being analyzed

Technology

The type of file being analyzed. For example:

- Java Source
- Decompiled Java File
- Manifest
- Properties
- EJB XML
- Spring XML
- Web XML
- Hibernate Cfg
- Hibernate Mapping

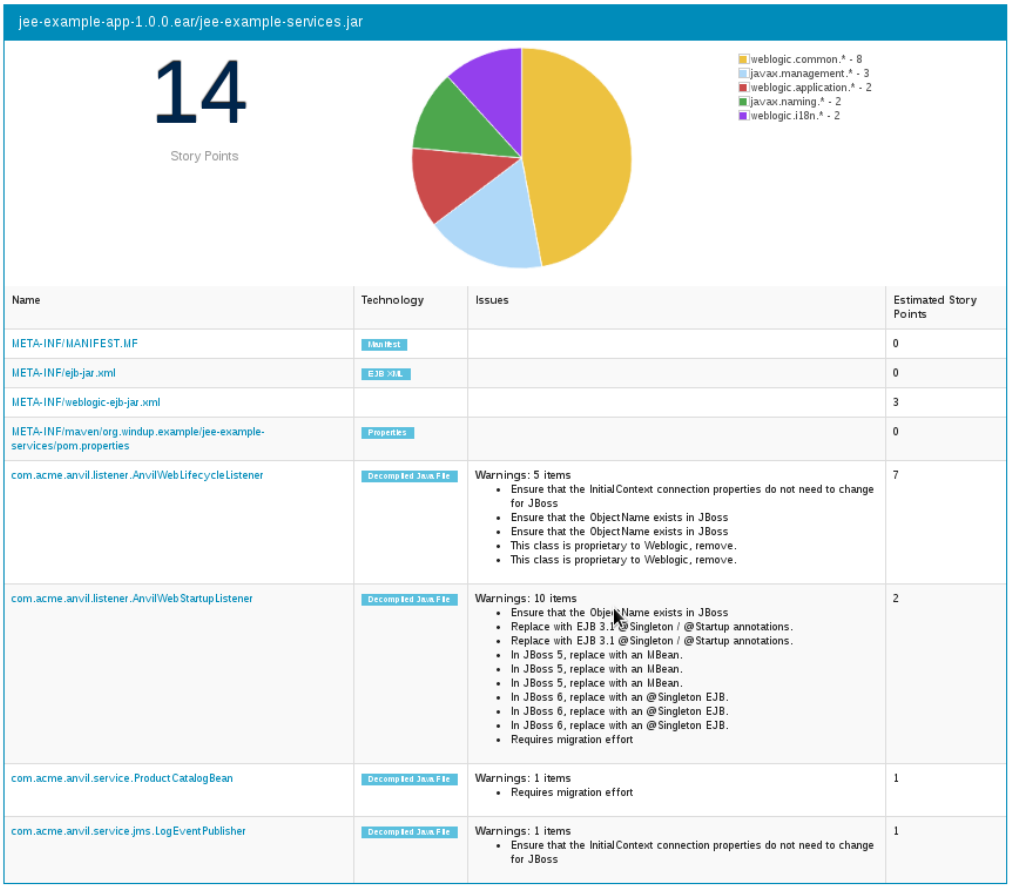
Issues

Warnings about areas of code that need review or changes.

Estimated Story Points

Level of effort required for migrating the file.

The following is an example of the archive analysis summary section of a Windup Report. In this example, it's the analysis of the `WINDUP_SOURCE/test-files/jee-example-app-1.0.0.ear/jee-example-services.jar`.

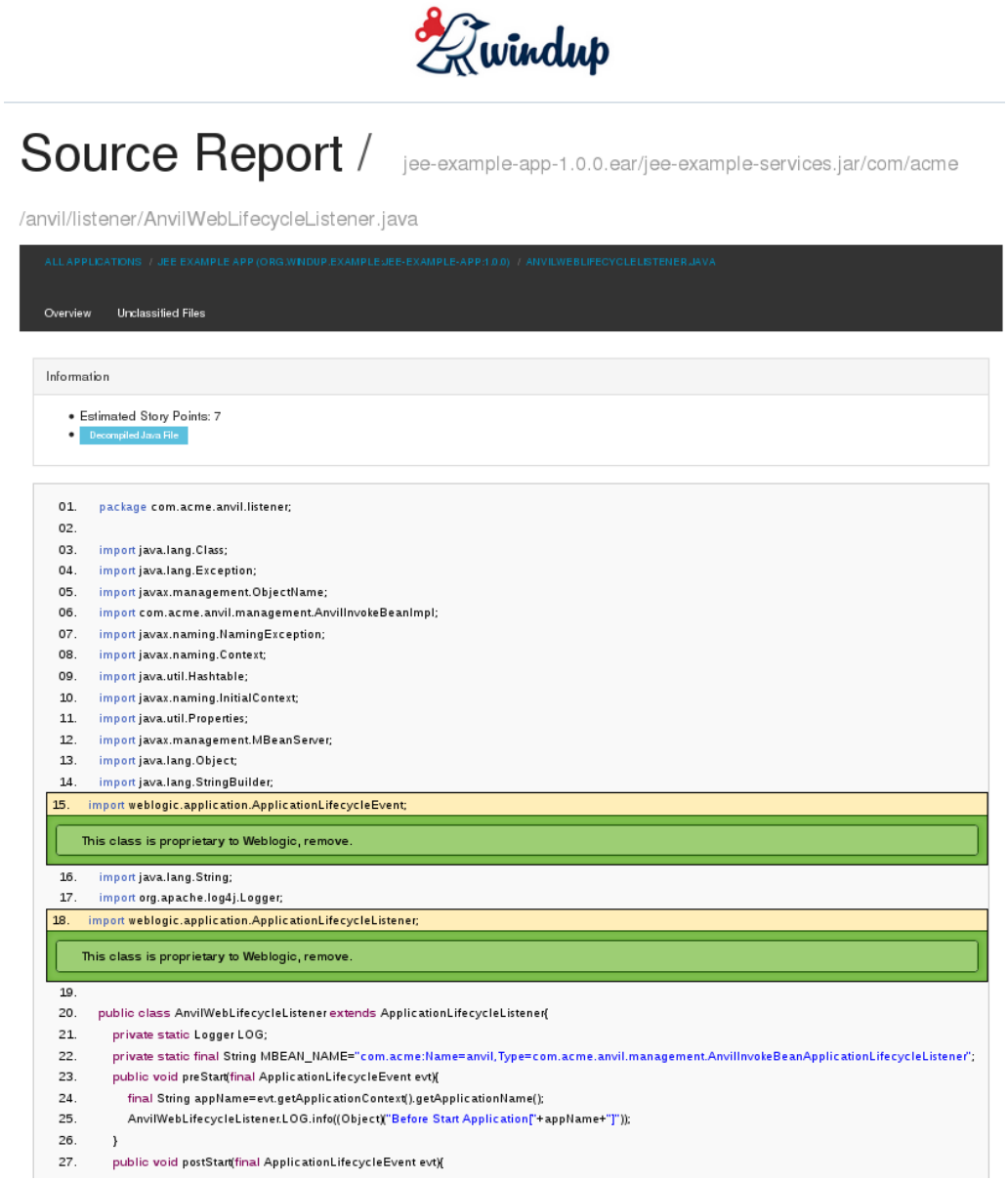


File Analysis Pages

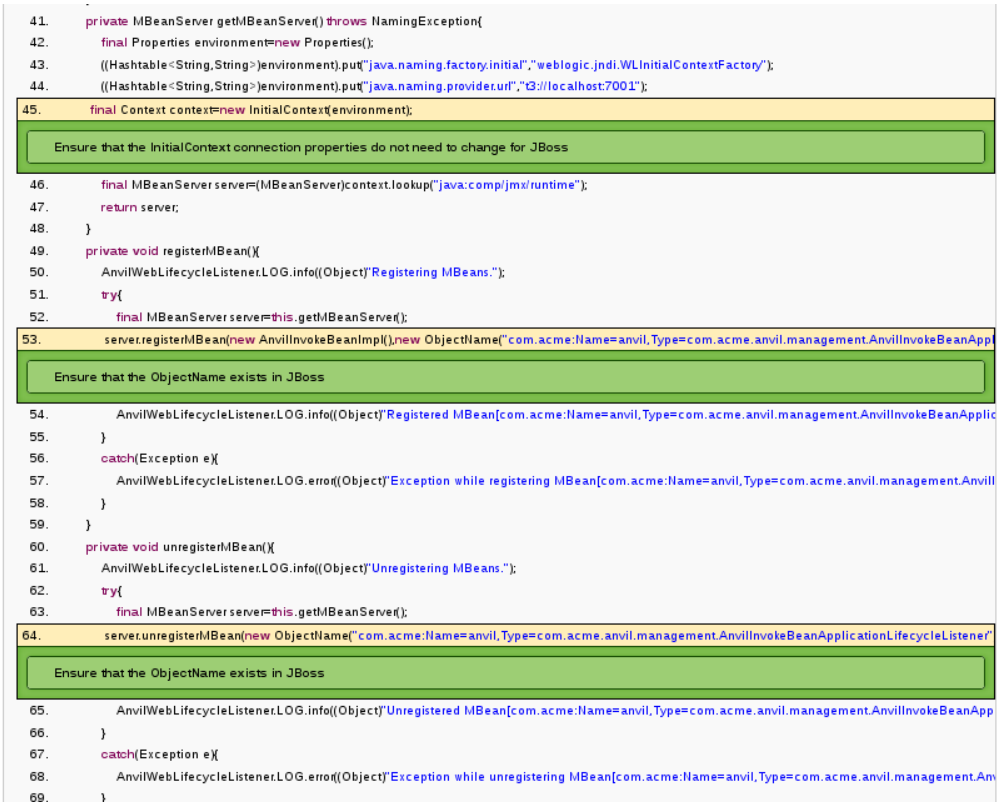
The analysis of the `jee-example-services.jar` lists the files in the JAR and the warnings and story points assigned to each one. Notice the `com.acme.anvil.listener.AnvilWebLifecycleListener` file has 5 warnings and is assigned 7 story points. Click on the file to see the detail.

- The **Information** section provides a summary of the story points and notes that the file was decompiled by Windup.
- This is followed by the file source code listing. Warnings appear in the file at the point where migration is required.

In this example, warnings appear at the import of `weblogic.application.ApplicationLifecycleEvent` and report that the class is proprietary to WebLogic and must be removed.



Later in the code, warnings appear for the creation of the InitialContext and for the object name when registering and unregistering an MBeans

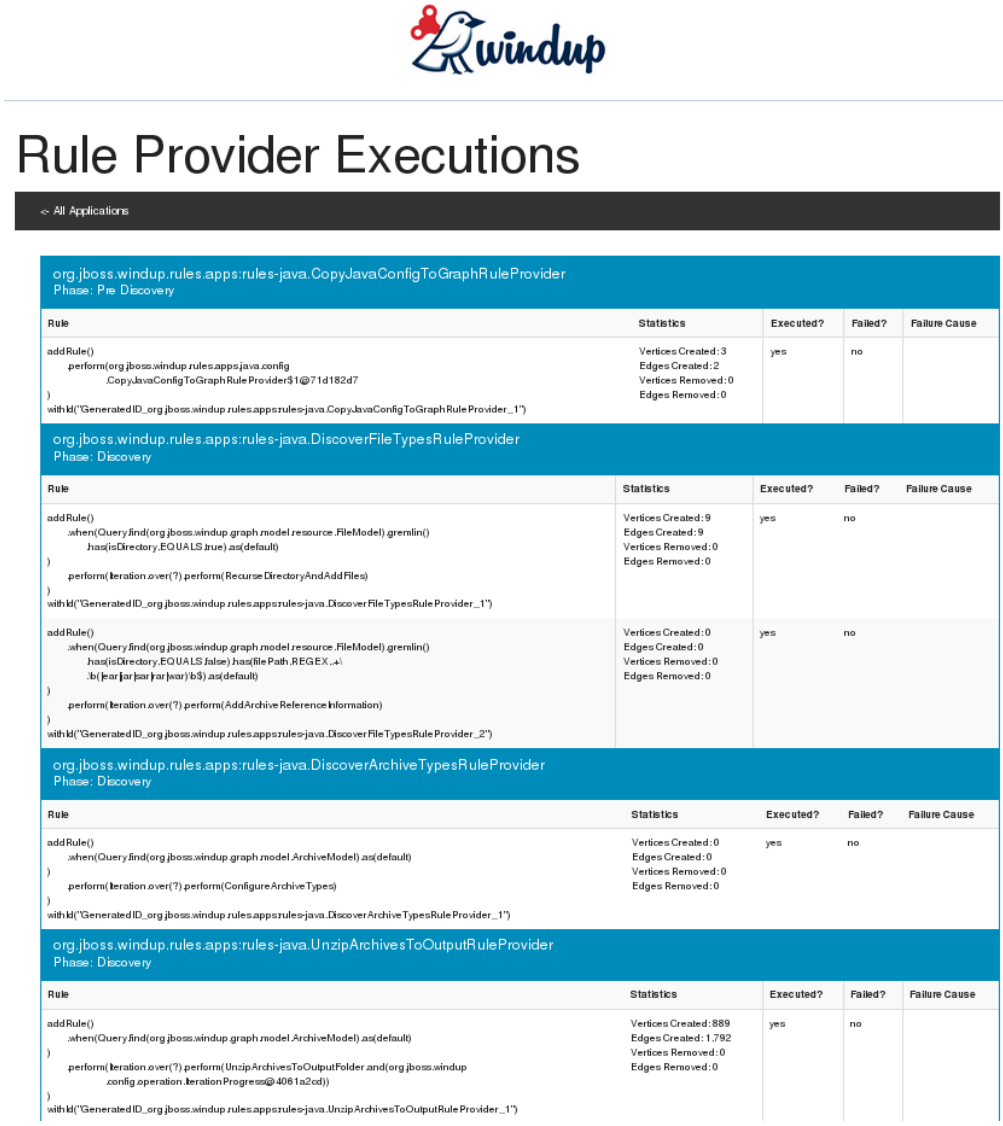


4.2.4. Additional Reports

Explore the Windup `OUTPUT_REPORT_DIRECTORY/reports` folder to find additional reporting information.

Rule Provider Execution Report

The `OUTPUT_REPORT_DIRECTORY/reports/windup_ruleproviders.html` page provides the list of rule providers that executed when running the Windup migration command against the application.



Rule Provider Executions

<- All Applications

org.jboss.windup.rules.apps:rules-java.CopyJavaConfigToGraphRuleProvider
Phase: Pre Discovery

Rule	Statistics	Executed?	Failed?	Failure Cause
addRule() .perform(org.jboss.windup.rules.apps.java.config .CopyJavaConfigToGraphRuleProvider$1@71d182d7) withId("GeneratedID_org.jboss.windup.rules.apps:rules-java.CopyJavaConfigToGraphRuleProvider_1")	Vertices Created: 3 Edges Created: 2 Vertices Removed: 0 Edges Removed: 0	yes	no	

org.jboss.windup.rules.apps:rules-java.DiscoverFileTypesRuleProvider
Phase: Discovery

Rule	Statistics	Executed?	Failed?	Failure Cause
addRule() .when(Query.find(org.jboss.windup.graph.model.resource.FileModel).gremlin() .has(isDirectory,EQUALS,true).as(default)) .perform(Iteration.over(?).perform(RecurseDirectoryAndAddFiles)) withId("GeneratedID_org.jboss.windup.rules.apps:rules-java.DiscoverFileTypesRuleProvider_1")	Vertices Created: 9 Edges Created: 9 Vertices Removed: 0 Edges Removed: 0	yes	no	
addRule() .when(Query.find(org.jboss.windup.graph.model.resource.FileModel).gremlin() .has(isDirectory,EQUALS,false).has(filePath,REGEX,.+\ .b(ear|jar|sar|rar|war)$).as(default)) .perform(Iteration.over(?).perform(AddArchiveReferenceInformation)) withId("GeneratedID_org.jboss.windup.rules.apps:rules-java.DiscoverFileTypesRuleProvider_2")	Vertices Created: 0 Edges Created: 0 Vertices Removed: 0 Edges Removed: 0	yes	no	

org.jboss.windup.rules.apps:rules-java.DiscoverArchiveTypesRuleProvider
Phase: Discovery

Rule	Statistics	Executed?	Failed?	Failure Cause
addRule() .when(Query.find(org.jboss.windup.graph.model.ArchiveModel).as(default)) .perform(Iteration.over(?).perform(ConfigureArchiveTypes)) withId("GeneratedID_org.jboss.windup.rules.apps:rules-java.DiscoverArchiveTypesRuleProvider_1")	Vertices Created: 0 Edges Created: 0 Vertices Removed: 0 Edges Removed: 0	yes	no	

org.jboss.windup.rules.apps:rules-java.UnzipArchivesToOutputRuleProvider
Phase: Discovery

Rule	Statistics	Executed?	Failed?	Failure Cause
addRule() .when(Query.find(org.jboss.windup.graph.model.ArchiveModel).as(default)) .perform(Iteration.over(?).perform(UnzipArchivesToOutputFolder.and(org.jboss.windup .config.operation.IterationProgress@4061a2cd))) withId("GeneratedID_org.jboss.windup.rules.apps:rules-java.UnzipArchivesToOutputRuleProvider_1")	Vertices Created: 889 Edges Created: 1,792 Vertices Removed: 0 Edges Removed: 0	yes	no	

Rule Provider Execution Report

The `OUTPUT_REPORT_DIRECTORY/reports/windup_ruleproviders.html` page provides the list of rule providers that executed when running the Windup migration command against the application.

Individual File Analysis Reports

You can directly access the the file analysis report pages described above by browsing for them by name in the `OUTPUT_REPORT_DIRECTORY/reports/` directory. Because the same common file names can exist in multiple archives, for example "manifest.mf" or "web.xml", Windup adds a unique numeric suffix to each report file name.

5. Additional Resources

5.1. Get Involved

5.1.1. How can you help?

To help us make Windup cover most application constructs and server configurations, including yours, you can help with any of the following items. Many require only a few minutes of your time!

- Send an email to windup-users@lists.jboss.org and let us know what Windup migration rules should cover.
- Provide example applications to test migration rules.
- Identify application components and problem areas that may be difficult to migrate.
  - Write a short description of these problem migration areas.
  - Write a brief overview describing how to solve the problem migration areas.
- Try Windup on your application. Be sure to report any issues you encounter.
- You can contribute to the Windup rules repository.

    - Write a Windup rule to identify or automate a migration process.
    - Create a test for the new rule.
    - Details are provided in the *Windup Rules Development Guide*.
- You can also contribute to the project source code.
    - Create a core rule.
    - Improve Windup performance or efficiency.
    - See the_Windup Core Development Guide_ for information about how to configure your environment and set up the project.

Any level of involvement is greatly appreciated!

5.1.2. Important links

See the list of links to Windup resources.

5.2. Important Links

- Windup wiki: https://github.com/windup/windup/wiki
- Windup documentation (generated from the Wiki documentation at the link above):
    - Windup User Guide
    - Windup Rules Development Guide
    - Windup Core Development Guide
    - Windup Javadoc
- Windup forums: https://community.jboss.org/en/windup
    - Windup 0.x legacy forums: https://developer.jboss.org/en/windup
- Windup issue tracker: https://issues.jboss.org/browse/WINDUP
- Windup users mailing List: windup-users@lists.jboss.org
- Windup developers mailing list: windup-dev@lists.jboss.org
- Windup commits mailing list: windup-commits@lists.jboss.org
- Windup on Twitter: @JBossWindup

- Windup IRC channel: Server FreeNode (`irc.freenode.net`), channel `#windup`.
  - Windup IRC Chat transcripts: http://bit.ly/windup-transcripts
  - Windup meeting IRC Chat transcripts: transcripts: http://bit.ly/windup-meetings

5.3. Review the Windup Quickstarts

The Windup 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. The quickstarts are available on GitHub here: https://github.com/windup/windup-quickstarts

You can fork and clone the project to have access to regular updates or you can download a ZIP file of the latest version.

5.3.1. Download the Latest Quickstart ZIP

To download the latest quickstart ZIP file, browse to: https://github.com/windup/windup-quickstarts/releases

Click on the most recent release to download the ZIP to your local file system.

5.3.2. Fork and Clone the Quickstart GitHub Project

If you don't have the GitHub client (`git`), download it from: http://git-scm.com/

1. Click the `Fork` link on the Windup 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 Windup 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 lates 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. To get the latest files from the `upstream` repository.

```
git reset --hard upstream/master
```

5.4. Known Windup Issues

Windup known issues are tracked here: Open Windup issues

5.5. Report Issues with Windup

Windup uses JIRA as its issue tracking system. If you encounter an issue executing Windup, please file a JIRA Issue.

5.5.1. Create a JIRA Account

If you do not yet have a JIRA account, create one using the following procedure.

1. Open a browser to the following URL: https://issues.jboss.org/secure/Dashboard.jspa
2. Click the *Sign Up* link in the top right side of the page.
3. Enter your email address and click the `Confirm address` button.
4. Follow the instructions sent to your email address.

5.5.2. Create a JIRA Issue

1. Open a browser to the following URL: https://issues.jboss.org/secure/CreateIssue!default.jspa.
   - If you have not yet logged in, click the *Log In* link at the top right side of the page.
   - Enter your credentials and click the `LOGIN` button.
   - You are then redirected back to the **Create Issue** page.
2. Choose the following options and click the `Next` button.

- **Project**: *Windup*
- **Issue Type**: *Bug*

3. On the next screen complete the following fields:
   - **Summary**: Enter a brief description of the problem or issue.
   - **Environment**: Provide the details of your operating system, version of Java, and any other pertinent information.
   - **Description**: Provide a detailed description of the issue. Be sure to include logs and exceptions traces.
4. Click the `Create` button to create the JIRA issue.
5. If the application or archive causing the issue does not contain sensitive information and you are comfortable sharing it with the Windup development team, attach it to the issue by choosing `More → Attach Files`. You are provided with an option to restrict visibility to JBoss employees.

---

6. Appendix

6.1. Glossary of Terms Used in Windup

6.1.1. Rules Terms

Rule

A piece of code that performs a single unit of work during the migration process. Depending on the complexity of the rule, it may or may not include configuration data. Extensive configuration information may be externalized into external configuration, for example, a custom XML file. The following is an example of a Java-based rule added to the `JDKConfig` RuleProvider class.

```
.addRule()

.when(JavaClass.references("java.lang.ClassLoader$").at(TypeReferenceLocation.TYPE))
    .perform(Classification.as("Java Classloader, must be migrated.")
    .with(Link.to("Red Hat Customer Portal: How to get resources via the ClassLoader
in a JavaEE application in JBoss EAP",
"https://access.redhat.com/knowledge/solutions/239033"))
     .withEffort(1))
```

RuleProvider

An implementation of OCPSoft ConfigurationProvider class specifically for Windup. It provides Rule instances and the relevant RuleProviderMetadata for those Java-based and XML-based Rule instances.

Ruleset

A ruleset is a group of one or more RuleProviders that targets a specific area of migration, for example, `Spring → Java EE 6` or `WebLogic → JBoss EAP`. A ruleset is packaged as a JAR and contains additional information needed for the migration, such as operations, conditions, report templates, static files, metadata, and relationships to other rulesets. The following Windup projects are rulesets.

- rules-java-ee
- rules-xml

Rules Metadata

Information about whether a particular ruleset applies to a given situation. The metadata can include the source and target platform and frameworks.

Rules Pipeline

A collection of rules that feed information into the knowledge graph.

6.1.2. Reporting Terms

Level of effort

The level of effort required for the migration task. The following values are used in the reports.

- *Lift and Shift*: The code or file is standards-based and can be ported to the new environment with no changes.
- *Known Solution*: There is a standard mapping algorithm to port the code or file to the new environment.
- *Custom*: The code or file must be rewritten or modified to work in the new environment.

Story Point

A term commonly used in Scrum Agile software development methodology to estimate the level of effort needed to implement a feature or change. It does not necessarily translate to man-hours, but the value should be consistent across tasks.

6.2. Rule Story Points

6.2.1. What are Story Points?

Story points are an abstract metric commonly used in Scrum Agile software development methodology to estimate the level of effort needed to implement a feature or change. They are based on a modified Fibonacci sequence.

In a similar manner, Windup uses story points to express the level of effort needed to migrate particular application constructs, and in a sum, the application as a whole. It does not necessarily translate to man-hours, but the value should be consistent across tasks.

6.2.2. How Story Points are Estimated in Rules

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

Level of Effort	Story Points	Description
Lift and Shift	0	The code or file is standards-based and requires no effort.
Mapped	1- 2 per file	There is a standard mapping algorithm to port the code or file. The number of story points required is small, but is dependent on the number of files to port.

Level of Effort	Story Points	Description
Custom	5 - 20 per change or component	The number of story points required to modify and rewrite code depends on the complexity of the change, the number of unknown imports, the size of the files, and the number of components. The following are examples of how to estimate story points. • Port MyBatis to JPA: '20' story points per query. • Port a web page from one web framework to another depends on the complexity and the number of components involved in the migration. You could estimate '20' story points per component.