

Windup Core Development Guide

Overview

This guide is for developers who plan to contribute source code updates or core rule add-ons to the Windup 2.0 project.



What is Windup?

JBoss Windup is a rule-based tool to simplify application migrations.

Running from a [Forge](#) environment, the tool examines EAR, WAR, and JAR deployments (or a project source directory) and produces an HTML report detailing the inner workings of the Java application to simplify migration efforts. It seeks to make migrating from other containers to JBoss EAP a piece of cake.

Windup 2.0 vs. Windup 0.7.x

Windup 2.0 aims to deliver the same functionality as legacy Windup, however, the internal architecture and rule structure is very different and allows for the creation of much more complex rules.

How Does Windup Simplify Migration?

Windup looks for common resources and highlight technologies and known “trouble spots” in migrating applications. The goal of Windup is to provide a high level view into relevant technologies in use within the application, and provide a consumable report for organizations to estimate, document, and migrate enterprise applications to Java EE and JBoss EAP.

These are some of the of the areas targetted by current core Windup rulesets:

Ruleset	Description
Java code	Reads compiled Java files, determines if blacklisted classes are imported, and if so, continues to profile the resource.
JSP code	Reads JSP files, extracts the JSP imports and taglibs, and continues to profile the resource
XML configuration	Reads XML files into a DOM objects

Follow Windup on Twitter!

Follow Windup on Twitter [@JBossWindup](#) for updates and more!

Features of Windup 2.0

Shared Data Model

Windup 2.0 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 2.0 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 2.0 provides more powerful and complex rules.

- XML-based rules are simple to write 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 2.0 has the ability to automate some of the migration processes.

- Windup is integrated with Forge 2.0, 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 2.0 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.

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-2.0.0.VERSION` folder extracted from the downloaded ZIP file.
- If you build Windup from GitHub source, `WINDUP_HOME` refers to the `windup-distribution-2.0.0-VERSION` folder extracted from the Windup source `dist/target/windup-distribution-2.0.0-VERSION.zip` file.

Get Started

Before you begin to contribute to Windup, take a quick tour to see how to use it.

Install and Configure Maven

If you plan to contribute to the core code base or create Java-based rule add-ons, you must first install and configure Maven to build Windup from source.

Download and Install Maven

If you plan to use Eclipse Luna (4.4) to build Windup, you can skip this step and go directly to the section entitled [Configure Maven to Build Windup](#). This version of Eclipse embeds Maven 3.2.1 so you do not need to install it

separately.

If you plan to run Maven using the command line or plan to use Red Hat JBoss Developer Studio 7.1.1 or an older version of Eclipse, you must install Maven 3.1.1. or later.

1. Go to [Apache Maven Project - Download Maven](#) and download the latest distribution for your operating system.
2. See the Maven documentation for information on how to download and install Apache Maven for your operating system.

Configure the Maven Installation in Your IDE

JBoss Developer Studio 7.1.1 is built upon Eclipse Kepler (4.3), which embeds Maven 3.0.4. If you plan to use JBoss Developer Studio 7.1.1 or an Eclipse version earlier than Eclipse Luna (4.4), you must replace the embedded 3.0.4 version of Maven with this newer version.

1. From the menu, choose **Window** -> **Preferences**.
2. Expand **Maven** and click on **Installations**.
3. Uncheck **Embedded (3.0.4/1.4.0.20130531-2315)**
4. Click **Add** and navigate to your Maven install directory. Select it and click **OK**.
5. Make sure the new external Maven installation is checked and click **OK** to return to JBoss Developer Studio.

Note: If you use another IDE, refer to the product documentation to update the Maven installation.

Configure Maven to Build Windup

Windup uses artifacts located in the [JBoss Nexus](#) repository. You must configure Maven to use this repository before you build Windup.

1. Open your `${user.home}/.m2/settings.xml` file for editing.
2. Copy the following `jboss-nexus-repository` profile XML prior to the ending `</profiles>` element.

```
<profile>
  <id>jboss-nexus-repository</id>
  <repositories>
    <repository>
      <id>jboss-nexus-repository</id>
      <url>http://repository.jboss.org/nexus/content/groups/public/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>jboss-nexus-plugin-repository</id>
      <url>http://repository.jboss.org/nexus/content/groups/public/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

```
</snapshots>
</pluginRepository>
</pluginRepositories>
</profile>
```

3. Copy the following XML prior to the ending `</activeprofiles>` element to make the profile active.

```
<activeProfile>jboss-nexus-repository</activeProfile>
```

You are now configured to build Windup.

Get the Windup Source Code

To contribute to the Windup 2.0 project source code, you must fork the Windup repository to your own Git, clone your fork, commit your work on topic branches, and make pull requests back to the Windup repository.

Install the Git Client

If you don't have the Git client (`git`), get it from: <http://git-scm.com/>

- Be sure to generate an SSH key and add the public key to your GitHub account: <https://help.github.com/articles/generating-ssh-keys>. You can verify the key using the following command:

```
ssh -T git@github.com
```

You should see a message indicating your ID has successfully authenticated.

- You should also configure your name and email identity using the following commands:

```
git config --global user_name "FIRST_NAME LAST_NAME"
git config --global user_email "YOUR_EMAIL_ADDRESS"
```

Fork and Clone the Windup Repository

1. [Fork](#) the Windup project. This creates the `windup` project in your own Git with the default remote name 'origin'.
2. Clone your fork. This creates and populates a directory in your local file system.

```
git clone https://github.com/<your-username>/windup.git
```

3. Change to the `windup` directory.
4. Add the remote `upstream` repository so you can fetch any changes to the original forked repository.

```
git remote add upstream git@github.com:windup/windup.git
```

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

```
git fetch upstream
```

Build Windup from Source

This information is provided for new developers who plan to contribute code to the Windup open source project. This section describes how to build Windup from source and how to extract the Windup distribution that is created during the build process.

If you use Linux and are an experienced Windup developer looking for a quick refresher, jump to: [Quick Build Review for Experienced Windup Developers](#).

System Requirements to Build Windup

1. Java 1.7.

You can choose from the following:

```
OpenJDK
Oracle Java SE
```

2. Maven 3.1.1 or newer

If you have not yet installed or configured Maven, see [Install and Configure Maven](#) for details.

If you have installed Maven, you can check the version by typing the following in a command prompt:

```
mvn --version
```

3. IDE requirements

If you prefer, you can work within an IDE. The following IDEs are recommended.

- [Red Hat JBoss Developer Studio 7.1.1](#) or newer
- [Eclipse 4.3 \(Kepler\)](#) or newer

You must also make sure the IDE embeds Maven 3.1.1 or later. See [Install and Configure Maven](#) for details.

Build Windup Using Maven Command Line

1. Make sure you have configured Maven as described here: [Install and Configure Maven](#).
2. Check out Windup source code: `git clone git@github.com:windup/windup.git` Windup. See [Get the Windup Source Code](#) for more info.
3. Open a command terminal and navigate to the root of the Windup project directory.

```
cd Windup/
```

4. To create a branch and build Windup using the latest version of the source code:

- Fetch the latest source code from the upstream repository.

```
git fetch upstream
```

- Checkout a local copy of the branch.

```
git checkout -b BRANCH_NAME upstream/master
```

5. If, at any point, you want to toss your changes and reset your local branch to the contents of the upstream repository, issue the following 2 commands. Do NOT do this if you have made changes that you don't want to lose!

```
git fetch upstream
```

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

6. Build the project.

```
mvn clean install
```

You can also build the project without the tests.

```
mvn clean install -DskipTests
```

Build Windup Using Red Hat JBoss Developer Studio or Eclipse

If you prefer, you can use an IDE to build Windup.

1. Make sure you have configured the Maven installation in your IDE as described here: [Install and Configure Maven](#).
2. Start JBoss Developer Studio or Eclipse.
3. From the menu, select **File** → **Import**.
4. In the selection list, choose **Maven** → **Existing Maven Projects**, then click **Next**.
5. Click **Browse** and navigate to the root of the Windup project directory, then click **OK**.
6. After all projects are listed, click **Next**. Ignore any Maven build or dependency errors and click **Finish**. If you get a dialog titled *Incomplete Maven Goal Execution*, ignore it and click **OK** to continue.
7. In the Project Explorer tab, find the **windup_parent** project in the list, right-click, and choose **RunAs** → **Maven install**.

Extract the Windup Distribution Source File

The build process creates a **windup-distribution-2.0.0-SNAPSHOT-offline.zip** file in the Windup source **dist/target/** directory.

Unzip the **dist/target/windup-distribution-2.0.0-SNAPSHOT-offline.zip** file into a directory of your choice.

Quick Build Review for Experienced Windup Developers

```
git clone git@github.com:windup/windup.git windup
## Or if you haven't set up github account:
#git clone https://github.com/windup/windup.git
cd windup
mvn clean install -DskipTests
rm -rf /tmp/windup-zip/ /tmp/windup/
unzip -q dist/target/windup-distribution-*-offline.zip -d /tmp/windup-zip
mv /tmp/windup-zip/windup-distribution-* /tmp/windup
```

Execute Windup (Built from Windup Source)

These instructions are for Windup core developers who plan to build Windup from source to test code updates.

- If you are new to the project and not familiar with the procedures to execute Windup, see [Execute Windup](#). It contains complete step-by-step instructions to execute Windup and also provides command line examples.
- Experienced users who need a refresher can follow the steps below.

```
mkdir tmp
cd tmp
unzip -q ./dist/target/windup-*.zip
cd windup-*
rm -rf ~/.forge/addons/
bin/windup.sh

## Execute Windup
$ windup-migrate-app --input /home/username/work/Migration/TestApps/_apps/BradApp --output
Report --packages org com net

## Exit Windup
$ exit

## View the Report
firefox Report/index.html
```

Execute Windup

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.

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

3. You are presented with the following prompt.

Using Windup at WINDUP_HOME

```
  _ _ _ _ _
 | | / ( ) _ _ _ _ _
 | | / / / _ V _ / / / _ \
 | | / / / / / _ / / / / /
 | _ / _ / / / _ \ _ _ / _ /
 | _ /
```

JBoss Windup, version [2.0.0-VERSION] - JBoss, by Red Hat, Inc. [<http://windup.jboss.org>]

[windup-distribution-2.0.0-VERSION]\$

Run Windup

1. The command to run Windup is `windup-migrate-app`.
2. The following is a list of the most commonly used command line arguments.

--input INPUT_ARCHIVE_OR_FOLDER

This is the fully qualified application archive or source path.

--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`.

--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`.

--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.

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](#).

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"
```

Windup 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
```

Windup Command Examples

The following Windup 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 [2.0.0.Alpha1 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.

Review the Report

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
```

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:



Overview / Profiled by Windup

Name	Technology	Effort	Issues
JEE Example App (org.windup.example:jee-example-app:1.0.0)			

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

Report Sections

Application Report Page

The first section of the application report page summarizes the migration effort. It provides the total *Story Points* and a graphically displays the effort by technology. 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.

- 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.

The following is an example of a Windup Application Report.



Application Report / JEE Example App

(org.windup.example:jee-example-app:1.0.0)



42

Story Points



java:example-app-1.0.0 10
org.windup.example:1.0.0 8
org.windup.example:1.0.0 4
org.windup.example:1.0.0 4
org.windup.example:1.0.0 2

jee-example-app-1.0.0.ear			
5			
Story Points			
Name	Technology	Issues	Estimated Story Points
org.windup.example:1.0.0.ear	Java		1
org.windup.example:1.0.0.ear	Java		0
org.windup.example:1.0.0.ear	Java		1
org.windup.example:1.0.0.ear	Java		3
org.windup.example:1.0.0.ear	Java		0

jee-example-app-1.0.0.ear/jee-example-services.jar			
14			
Story Points			
Name	Technology	Issues	Estimated Story Points
org.windup.example:1.0.0.ear	Java		0
org.windup.example:1.0.0.ear	Java		0
org.windup.example:1.0.0.ear	Java		3
org.windup.example:1.0.0.ear	Java		0
org.windup.example:1.0.0.ear/jee-example-services.jar	Java		0
org.windup.example:1.0.0.ear/jee-example-services.jar	Java	Warnings: 3 items • Ensure that the classLoaderName property is not used to change the class loader. • Ensure that the classLoaderName is not used. • The class is prepared to be loaded by the class loader. • The class is prepared to be loaded by the class loader.	7
org.windup.example:1.0.0.ear/jee-example-services.jar	Java	Warnings: 10 items • Ensure that the classLoaderName is not used. • Replace with the classLoaderName property. • Replace with the classLoaderName property. • Replace with the classLoaderName property.	2

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

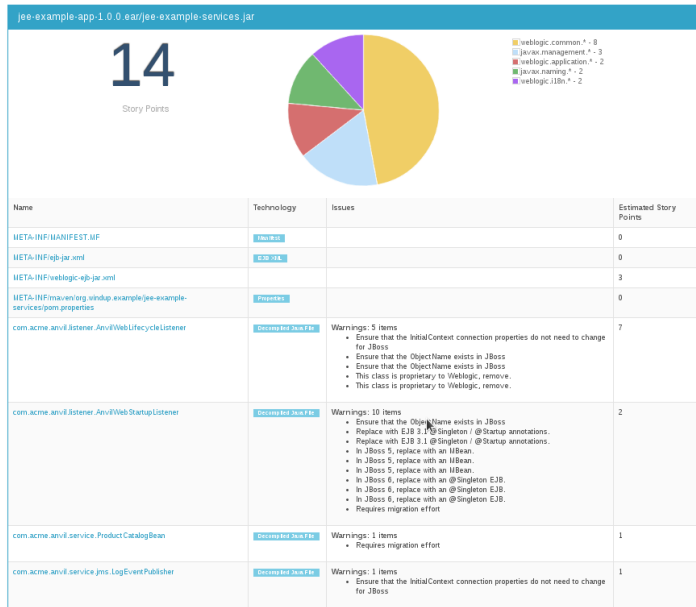
Issues

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.



Source Report / jee-example-app-1.0.0.ear/jee-example-services.jar/com/acme

/anvil/listener/AnvilWebLifecycleListener.java

ALL APPLICATIONS / JEE-EXAMPLE-APP (JOS) WINDUP-EXAMPLE-JEE-EXAMPLE-APP (1.0.0) / ANVILWEBLIFECYCLELISTENER.JAVA

OverviewUnclassified Files

Information

- Estimated Story Points: 7
- [Download Java file](#)

```
01. package com.acme.anvil.listener;
02.
03. import java.lang.Class;
04. import java.lang.Exception;
05. import java.management.ObjectName;
06. import com.acme.anvil.management.AnvilInvokeBeanImpl;
07. import java.naming.NamingException;
08. import java.naming.Context;
09. import java.util.Hashtable;
10. import java.naming.InitialContext;
11. import java.util.Properties;
12. import java.management.MBeanServer;
13. import java.lang.Object;
14. import java.lang.StringBuilder;
15. import weblogic.application.ApplicationLifecycleEvent;
16.
17. import java.lang.String;
18. import org.apache.jagg.Logger;
19.
20. import weblogic.application.ApplicationLifecycleListener;
21.
22. This class is proprietary to WebLogic, remove.
23.
24. import java.lang.String;
25. import org.apache.jagg.Logger;
26.
27. import weblogic.application.ApplicationLifecycleListener;
28.
29. This class is proprietary to WebLogic, remove.
30.
31. public class AnvilWebLifecycleListener extends ApplicationLifecycleListener{
32.     private static Logger LOG;
33.     private static final String MBEAN_NAME = "com.acme:name=anvil,Type=com.acme.anvil.management.AnvilInvokeBeanApplicationLifecycleListener";
34.     public void preStart(final ApplicationLifecycleEvent evt){
35.         final String appName = evt.getApplicationContext().getApplicationName();
36.         AnvilWebLifecycleListener.LOG.info("Before Start Application[" + appName + "]");
37.     }
38.     public void postStart(final ApplicationLifecycleEvent evt){
```

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

```
41. private MBeanServer getMBeanServer() throws NamingException{
42.     final Properties environment = new Properties();
43.     ((Hashtable<String,String>)environment).put("java.naming.factory.initial","weblogic.jndi.WLInitialContextFactory");
44.     ((Hashtable<String,String>)environment).put("java.naming.provider.url","G://localhost:7002");
45.     final Context context = new InitialContext(environment);
46.
47.     Ensure that the InitialContext connection properties do not need to change for JBoss
48.
49.     final MBeanServer server = (MBeanServer)context.lookup("java:comp/jmx/runtime");
50.     return server;
51. }
52. private void registerMBean(){
53.     AnvilWebLifecycleListener.LOG.info("Registering MBeans.");
54.     try{
55.         final MBeanServer server = this.getMBeanServer();
56.         server.registerMBean(new AnvilInvokeBeanImpl(), new ObjectName("com.acme:name=anvil,Type=com.acme.anvil.management.AnvilInvokeBeanApplicationLifecycleListener"));
57.
58.         Ensure that the ObjectName exists in JBoss
59.
60.         AnvilWebLifecycleListener.LOG.info("Registered MBean[" + com.acme:name=anvil,Type=com.acme.anvil.management.AnvilInvokeBeanApplicationLifecycleListener");
61.     }
62.     catch(Exception e){
63.         AnvilWebLifecycleListener.LOG.error("Exception while registering MBean[" + com.acme:name=anvil,Type=com.acme.anvil.management.AnvilInvokeBeanApplicationLifecycleListener");
64.     }
65. }
66. private void unregisterMBean(){
67.     AnvilWebLifecycleListener.LOG.info("Unregistering MBeans.");
68.     try{
69.         final MBeanServer server = this.getMBeanServer();
70.         server.unregisterMBean(new ObjectName("com.acme:name=anvil,Type=com.acme.anvil.management.AnvilInvokeBeanApplicationLifecycleListener"));
71.
72.         Ensure that the ObjectName exists in JBoss
73.
74.         AnvilWebLifecycleListener.LOG.info("Unregistered MBean[" + com.acme:name=anvil,Type=com.acme.anvil.management.AnvilInvokeBeanApplicationLifecycleListener");
75.     }
76.     catch(Exception e){
77.         AnvilWebLifecycleListener.LOG.error("Exception while unregistering MBean[" + com.acme:name=anvil,Type=com.acme.anvil.management.AnvilInvokeBeanApplicationLifecycleListener");
78.     }
79. }
```

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](#)

Rule	Statistics	Executed?	Failed?	Failure Cause
<pre>addRule(perform(org.jboss.windup.rules.apps.rules.java.CopyJavaConfigToGraphRuleProvider Phase : Discovery) withId("Generated_EI_org.jboss.windup.rules.apps.rules.java.CopyJavaConfigToGraphRuleProvider_1")</pre>	Vertices Created: 3 Edges Created: 2 Vertices Removed: 0 Edges Removed: 0	yes	no	
org.jboss.windup.rules.apps.rules.java.DiscoverFileTypeRuleProvider Phase : Discovery				
Rule	Statistics	Executed?	Failed?	Failure Cause
<pre>addRule(where(Query.find(org.jboss.windup.graph.model.Resource.RxModelModel).generate() has(LazyDirectory.EQUALS true) and (default)) perform(iteration over ?1) perform(Resource.Director.findAndAddFiles)) withId("Generated_EI_org.jboss.windup.rules.apps.rules.java.DiscoverFileTypeRuleProvider_1")</pre>	Vertices Created: 9 Edges Created: 9 Vertices Removed: 0 Edges Removed: 0	yes	no	
org.jboss.windup.rules.apps.rules.java.DiscoverFileTypesRuleProvider_1				
Rule	Statistics	Executed?	Failed?	Failure Cause
<pre>addRule(where(Query.find(org.jboss.windup.graph.model.Resource.RxModelModel).generate() has(LazyDirectory.EQUALS false) has(LazyPath.REGEX ".+\.jar\$") and (default)) perform(iteration over ?1) perform(AddArchivesReferenceInformation)) withId("Generated_EI_org.jboss.windup.rules.apps.rules.java.DiscoverFileTypesRuleProvider_2")</pre>	Vertices 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
<pre>addRule(where(Query.find(org.jboss.windup.graph.model.ArchiveModel) and (default)) perform(iteration over ?1) perform(ConfigurateArchiveTypes)) withId("Generated_EI_org.jboss.windup.rules.apps.rules.java.DiscoverArchiveTypesRuleProvider_1")</pre>	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
<pre>addRule(where(Query.find(org.jboss.windup.graph.model.ArchiveModel) and (default)) perform(iteration over ?1) perform(UnzipArchivesToOutputFolder and(org.jboss.windup config.operation.iteration.Progress) add(RxNode))) withId("Generated_EI_org.jboss.windup.rules.apps.rules.java.UnzipArchivesToOutputRuleProvider_1")</pre>	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.

Index of file:///home/username/windup-jee-example-app-report/reports/

[Up to higher level directory](#)

Name	Size	Last Modified
Agent_java.1.html	7 KB	10/31/2014 08:41:14 AM
Agent_java.2.html	7 KB	10/31/2014 08:41:15 AM
AnvilWebLifecycleListener_java.1.html	11 KB	10/31/2014 08:41:14 AM
AnvilWebStartupListener_java.1.html	11 KB	10/31/2014 08:41:14 AM
AppenderDynamicMBean_java.1.html	16 KB	10/31/2014 08:41:14 AM
AppenderDynamicMBean_java.2.html	16 KB	10/31/2014 08:41:14 AM
AuthenticatorFilter_java.1.html	7 KB	10/31/2014 08:41:14 AM
HierarchyDynamicMBean_java.1.html	15 KB	10/31/2014 08:41:14 AM
HierarchyDynamicMBean_java.2.html	15 KB	10/31/2014 08:41:14 AM
JDBCAppender_java.1.html	10 KB	10/31/2014 08:41:14 AM
JDBCAppender_java.2.html	10 KB	10/31/2014 08:41:14 AM
JEE_Example_App...org_windup_example_je...example_app_1_0_0_1.html	26 KB	10/31/2014 08:41:16 AM
JMSAppender_java.1.html	14 KB	10/31/2014 08:41:15 AM
JMSAppender_java.2.html	14 KB	10/31/2014 08:41:14 AM
LogEventPublisher_java.1.html	8 KB	10/31/2014 08:41:15 AM
LogEventTopic_jms.xml.1.html	5 KB	10/31/2014 08:41:15 AM
LoggerDynamicMBean_java.1.html	14 KB	10/31/2014 08:41:15 AM
LoggerDynamicMBean_java.2.html	14 KB	10/31/2014 08:41:14 AM
LoginFilter_java.1.html	6 KB	10/31/2014 08:41:15 AM
MANIFEST_MF.1.html	5 KB	10/31/2014 08:41:16 AM
MANIFEST_MF.2.html	5 KB	10/31/2014 08:41:15 AM
MANIFEST_MF.3.html	5 KB	10/31/2014 08:41:15 AM

Developer Contributing Information

Development Guidelines and Conventions

- [Project Source Code Formatting](#)
- [Source Code Naming Conventions](#)
- [Maven Project Naming Conventions](#)

Project Source Code Formatting

All project source code contributed to Windup should be formatted using the settings defined in the `Eclipse_Code_Format_Profile.xml` file located in the root directory of the Windup project.

Eclipse IDE

1. In Eclipse, choose Windows → Preferences.
2. Expand Java → Code Style → Formatter
3. Click Import
4. Browse to the `Eclipse_Code_Format_Profile.xml` located in the root of the Windup project directory, then click 'OK'.

Netbeans IDE

Use this file to format Windup source code:

<http://plugins.netbeans.org/plugin/50877/eclipse-code-formatter-for-java>

IntelliJ IDEA

Use this file to format Windup source code:

<http://plugins.jetbrains.com/plugin/?id=6546>

Source Code Naming Conventions

Class Interface and Implementation Names

- Do not name interfaces using the prefix 'I' for interface names. Instead, use a descriptive term for the interface, like `Module.java`.
- The implementation class name should begin with a descriptive name, followed by the interface name, for example, for example `JavaHandlerModule.java`

Standard Prefixes and Suffixes

- Append all `RuleProvider` class names with `RuleProvider`.
- Append all XML rule file names with `.windup.xml`
- Append all `Model` class names with `Model`.
- All test names should be prefixed with 'Test'.
- Property constants: TBD

Maven Project Naming Conventions

Maven Module Names

The following are not really accurate at this time. This is still TBD!

- Lowercase with dashes. Start with `windup-`.
- Ruleset add-ons start with `windup-rules-`.

Submit Code Updates to the Windup Project

To get the Windup Source Code, see [Get the Windup Source Code](#) for instructions.

1. Open a command terminal and navigate to the root of the Windup project directory.
2. Create a new topic branch to contain your features, changes, or fixes using the `git checkout` command:

```
git checkout -b <topic-branch-name> upstream/master
```

If you are fixing a JIRA, it is a good practice to use the number in the branch name. For example:

```
git checkout -b WINDUP-225 upstream/master
```

3. Make changes or updates to the source files. Be sure to test the changes thoroughly!
4. When you are sure your updates are ready and working, use the `git add` command to add new or changed file contents to the staging area.

```
git add <folder-name>/  
git add <file-name>
```

5. Use the `git status` command to view the status of the files in the directory and in the staging area and ensure that all modified files are properly staged:

```
git status
```

6. Commit your changes to your local topic branch. For example:

```
git commit -m 'WINDUP-225: Description of change...'
```

7. Push your local topic branch to your GitHub forked repository. This will create a branch on your GitHub fork repository with the same name as your local topic branch name.

```
git push origin HEAD
```

Note: The above command assumes your remote repository is named 'origin'. You can verify your forked remote repository name using the command ``git remote -v``.

8. Browse to the newly created branch on your forked GitHub repository.

```
https://github.com/<your-username>/windup/tree/<topic-branch-name>
```

9. Open a Pull Request. For details, see [Using Pull Requests](#).
 - Give the pull request a clear title and description.
 - Review the modifications that are to be submitted in the pull to be sure it contains only the changes you expect.

10. The pull request will be reviewed and merged by a Windup project administrator.

Understand the Windup Architecture and Structure

Windup Architectural Components

The following open source software, tools, and APIs are used within Windup to analyze and provide migration information. If you plan to contribute source code to the core Windup 2.0 project, you should be familiar with them.

Forge

Forge is an open source, extendable, rapid application development tool for creating Java EE applications using Maven. For more information about Forge 2, see: [JBoss Forge](#).

Forge Furnace

Forge Furnace is a modular runtime container behind Forge that provides the ability to run Forge add-ons in an embedded application. For more information about Forge Furnace, see: [Run Forge Embedded](#).

TinkerPop

TinkerPop is an open source graph computing framework. For more information, see: [TinkerPop](#).

Titan

Titan is a scalable graph database optimized for storing and querying graphs. For more information, see: [Titan Distributed Graph Database](#) and [Titan Beginner's Guide](#).

Frames

Frames represents graph data in the form of interrelated Java Objects or a collection of annotated Java Interfaces. For more information, see: [TinkerPop Frames](#).

Windup includes several Frames extensions, which are documented here: [Frames Extensions](#).

Gremlin

Gremlin is a graph traversal language that allows you to query, analyze, and manipulate property graphs that implement the Blueprints property graph data model. For more information, see: [TinkerPop Gremlin Wiki](#).

Blueprints

Blueprints is an industry standard API used to access graph databases. For more information about Blueprints, see: [TinkerPop Blueprints Wiki](#).

Pipes

Pipes is a dataflow framework used to process graph data. It for the transformation of data from input to output. For more information, see: [Tinkerpop Pipes Wiki](#).

Rexster

Rexster is a graph server that exposes any Blueprints graph through HTTP/REST and a binary protocol called RexPro. Rexster makes extensive use of Blueprints, Pipes, and Gremlin. For more information, see: [TinkerPop Rexster Wiki](#).

OCPsoft Rewrite

OCPsoft Rewrite is an open source routing and URL rewriting solution for Servlets, Java Web Frameworks, and Java EE. For more information about Ocpsoft Rewrite, see: [OCPsoft Rewrite](#).

Windup Project Structure

The Windup 2.0 project consists of the following subprojects.

config/

This project is for the engine that runs the rules and abstracts the graph operations.

decompiler/

This subproject contains an API that wraps calls to the decompiler. Windup currently uses only one decompiler: *Procyon*

exec/

This subproject contains the bootstrap code to run the Windup application.

ext/

This subproject is for code extensions. It currently only contains the Groovy rules syntax. Eventually it will contain any code that is not related to the rules or the core code base.

graph/

This subproject contains the datastore and Frames extensions.

logging/

This is the logging subproject. This subproject may be removed, depending on the outcome of this JIRA: [WINDUP-49](#).

reporting/

This subproject contains code that does reporting.

rules/

This subproject contains all the rules.

test-files/

This subproject contains the demo applications that are used for test input.

tests/

This subproject contains the integration test suite.

tinkerpop/

This subproject contains a code fix for Titan NPE issues.

ui/

This subproject contains experimental Forge UI code.

utils/

This subproject contains all utility code.

Rules and Rulesets

Windup Processing Overview

Windup is a rule-based migration tool that allows you to write customized rules to analyze the APIs, technologies, and architectures used by the applications you plan to migrate. The Windup tool also executes its own core rules through all phases of the migration process.

The following is a high level conceptual overview of what happens within Windup when you execute the tool against your application or archive.

Discovery Phase

When you run the `windup-migrate-app` command, Windup executes its own core rules to extract files from archives, decompile classes, and analyze the application. In this phase Windup builds a data model 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.

For more information about the phases of rule execution and how to control rule dependencies, see [Rule Execution Lifecycle](#).

For more information about the graph database components, see [Windup Architectural Components](#).

Application Migration

The next step in the process is the execution of the migration rules. In this phase, the rules typically do not execute against the application input files. Instead, they execute against the graph database model. Windup rules are independent and decoupled and they communicate with each other using the graph database model. Rules query the graph database to obtain information needed to test the rule condition. They also update the data model with information based on the result of the rule execution. This allows rules to easily interact with other rules and enables the creation of very complex rules.

The Windup distribution contains a large number of migration rules, but in some cases, you may need to create additional custom rules for your specific implementation. Windup's architecture allows you to create Java-based rule addons or XML rules and easily add them to Windup. Custom rule creation is covered in the *Windup Rules Development Guide*.

Generate Findings Based on the Rule Execution Results

The final step in the process is to pull data from the graph database model

to generate of reports and optionally generate scripts. Again, Windup uses rules to generate the final output.

By default, Windup generates the following reports at the end of the application migration process. The reports are located in the `reports/` subdirectory of the output report path specified when you execute Windup:

- **Application Report:** This report provides a summary of the total estimated effort, or [story points](#), that are required for the migration. It also provides a detailed list of issues and suggested changes, broken down by archive or folder.
- **RuleProvider report:** This is a detailed listing of the rule providers that fired when running Windup and whether any errors occurred.
- **Additional reports** are generated that provide detailed line-by-line migration tips for individual files.

Windup can also generate scripts to automate migration processes based on the findings. For example, some configuration files are easily mapped and can be automatically generated as part of the migration process.

Rule Execution Lifecycle

Rule Lifecycle

Windup executes each rule in a sequential order. The following sections describe the phases of rule execution and how rules may specify that one or more other rules must be executed before or after they are run.

For a graphical overview of rule processing, see [this diagram](#).

Rule Phases

```
INFO: Loaded [67] org.jboss.windup.config.WindupRuleProvider [
org.jboss.windup.config.phase.Implicit
org.jboss.windup.config.phase.Initialization
org.jboss.windup.config.phase.Discovery
org.jboss.windup.config.phase.ArchiveExtraction
org.jboss.windup.config.phase.ArchiveMetadataExtraction
org.jboss.windup.config.phase.ClassifyFileTypes
org.jboss.windup.config.phase.DiscoverProjectStructure
org.jboss.windup.config.phase.Decompilation
org.jboss.windup.config.phase.InitialAnalysis
org.jboss.windup.config.phase.MigrationRules
org.jboss.windup.config.phase.PostMigrationRules
org.jboss.windup.config.phase.PreReportGeneration
org.jboss.windup.config.phase.ReportGeneration
org.jboss.windup.config.phase.PostReportGeneration
org.jboss.windup.config.phase.ReportRendering
org.jboss.windup.config.phase.PostReportRendering
org.jboss.windup.config.phase.Finalize
org.jboss.windup.config.phase.PostFinalize
]
```

By default, a rule runs during the [MIGRATION_RULES](#) phase. However, a rule may require certain processing or actions to occur before it executes, such as the extraction of archives and scanning of java or XML files.

Rule phases provide a way for rule authors to specify and control in which phase of the rule lifecycle the rule should execute. This is done by overriding the `WindupRuleProvider.getPhase()` method. The following

example specifies this rule should execute during the [INITIAL_ANALYSIS](#) rule phase.

```
@Override
public RulePhase getPhase() {
    return InitialAnalysis.class;
}
```

The following is the list of rule phases as defined in the `RulePhase` enum class. Note that there are also `PRE_` processing and `POST_` processing phases for each of the following rule phases.

DISCOVERY

This phase is called during resource discovery. Static files are scanned by their basic properties, for example, the name, extension, location, and fully qualified Java class name. Archives are unzipped in this phase. Typically, any rule that only puts data into the graph is executed during this phase.

INITIAL_ANALYSIS

This phase is called to perform a basic analysis of the files content. It extracts all method names from class files, extracts metadata, such as the XML namespace and root element, from XML files.

COMPOSITION

This phase is called to perform high-level composition operations on the graph. For example, it may link items found in XML files to the related Java classes or references to server resources in Java classes.

MIGRATION_RULES

This phase is the default phase for all rules unless overridden. During this phase, migration rules attach data to the graph associated with migration. This could include: - Hints to migrators for manual migration - Automated migration of schemas or source segments - Blacklists to indicate vendor specific APIs.

REPORT_GENERATION

During this phase, reporting visitors produce report data in the graph that will later be used by the report rendering phase.

REPORT_RENDERING

This is the phase that renders the report.

FINALIZE

This phase is called to clean up resources and close streams.

For more information about rule phases, see the [RulePhase Javadoc](#).

Execute Before and Execute After

A rule may specify that one or more rules must be executed before it is run. In this case, all named rules will be fired in the order specified before executing the the current rule.

```
@Override
public List<Class<? extends WindupRuleProvider>> getExecuteBefore()
{
    return asClassList(RuleToFireBefore.class);
}
```

A rule may also specify that one or more rules must be executed after it is run. In this case, all named rules will be fired in the order specified after executing the the current rule.

```
@Override
public List<Class<? extends WindupRuleProvider>> getExecuteAfter()
{
    return asClassList(RuleToFireAfter.class);
}
```

Rule Story Points

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.

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.
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

		<p>the number of components. The following are examples of how to estimate story points.</p> <ul style="list-style-type: none"> • 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.
--	--	---

Difference Between XML-based and Java-based Rules

First of all, to prevent confusion:

1. Rules can be *written* using XML or the Java API. We call them XML-based and Java-based Rules.
2. Independent of the syntax, the rules may *inspect* (classify) XML or Java code.

See the examples below.

Which one to choose?

The XML rules provide a quick, simple way to create rules to analyze Java and XML files. Java rules provide the ability to create very complex rules.

- If you simply need to highlight a specific section of Java code or XML file content and provide hints for it, it is recommended that you create the rule using XML.
- If you need to create a new custom report, you need to create the reporting rules using Java.
- If you need to extend functionality beyond what the XML rules provide, you need to create the rule using Java.

Pros and cons - XML

Pros:

- XML rules are fairly easy to write and require less code.
- You do not need to configure Maven to build from source because there is no need for compilation of XML rules.

- Deployment is simple. You simply drop the XML 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.

Pros and Cons - Java

Pros:

- Java rule add-ons allow you to write custom rules and provide a lot of flexibility.
- 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 the Windup 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 writing a lot of Java code/
- Writing Java rule add-ons are complex and required knowledge of Windup internals.

Examples

Example of a rule written in XML classifying 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>
```

Example of a rule written in Java classifying Java code:

```
/**
 * Scans for classes with EJB related annotations, and adds EJB related metadata for these.
 */
public class DiscoverEjbAnnotationsRuleProvider extends WindupRuleProvider
{
    @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.ANNOTATION))
    .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.ANNOTATION).as(EN
TITY_ANNOTATIONS)

.or(JavaClass.references("javax.persistence.Table").at(TypeReferenceLocation.ANNOTATION).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");
}
...
}

```

Summary

The following is a draft... it may go.

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

Create and Test Java Rule Add-ons

Install and Configure Maven

If you plan to contribute to the core code base or create Java-based rule add-ons, you must first install and configure Maven to build Windup from source.

Download and Install Maven

If you plan to use Eclipse Luna (4.4) to build Windup, you can skip this step and go directly to the section entitled [Configure Maven to Build Windup](#). This version of Eclipse embeds Maven 3.2.1 so you do not need to install it separately.

If you plan to run Maven using the command line or plan to use Red Hat JBoss Developer Studio 7.1.1 or an older version of Eclipse, you must install Maven 3.1.1. or later.

1. Go to [Apache Maven Project - Download Maven](#) and download the latest distribution for your operating system.
2. See the Maven documentation for information on how to download and install Apache Maven for your operating system.

Configure the Maven Installation in Your IDE

JBoss Developer Studio 7.1.1 is built upon Eclipse Kepler (4.3), which embeds Maven 3.0.4. If you plan to use JBoss Developer Studio 7.1.1 or an Eclipse version earlier than Eclipse Luna (4.4), you must replace the embedded 3.0.4 version of Maven with this newer version.

1. From the menu, choose **Window** -> **Preferences**.
2. Expand **Maven** and click on **Installations**.
3. Uncheck **Embedded (3.0.4/1.4.0.20130531-2315)**
4. Click **Add** and navigate to your Maven install directory. Select it and click **OK**.
5. Make sure the new external Maven installation is checked and click **OK** to return to JBoss Developer Studio.

Note: If you use another IDE, refer to the product documentation to update the Maven installation.

Configure Maven to Build Windup

Windup uses artifacts located in the [JBoss Nexus](#) repository. You must configure Maven to use this repository before you build Windup.

1. Open your `${user.home}/.m2/settings.xml` file for editing.
2. Copy the following `jboss-nexus-repository` profile XML prior to the ending `</profiles>` element.

```
<profile>
  <id>jboss-nexus-repository</id>
  <repositories>
```

```

<repository>
  <id>jboss-nexus-repository</id>
  <url>http://repository.jboss.org/nexus/content/groups/public/</url>
  <releases>
    <enabled>true</enabled>
  </releases>
  <snapshots>
    <enabled>true</enabled>
  </snapshots>
</repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>jboss-nexus-plugin-repository</id>
    <url>http://repository.jboss.org/nexus/content/groups/public/</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </pluginRepository>
</pluginRepositories>
</profile>

```

3. Copy the following XML prior to the ending `</activeprofiles>` element to make the profile active.

```
<activeProfile>jboss-nexus-repository</activeProfile>
```

You are now configured to build Windup.

Java-based Rule Structure

TO_DO: Add a how-to for compound rules, nested rules, rules over multiple sources, negative queries (not matched by anything).

Windup Rule Provider

Windup rules are based on [OCPsoft Rewrite](#), an open source routing and URL rewriting solution for Servlets, Java Web Frameworks, and Java EE. The **rewrite** framework allows you to create rule providers and rules in an easy to read format.

Windup rule add-ons must extend the [WindupRuleProvider](#) class.

- If the rule should run in a phase other than the default [MIGRATION_PHASE](#), you must implement the [getPhase\(\)](#) method and specify in which Windup lifecycle phase the rule should be executed. For more information about rule phases, see [Rule Execution Lifecycle](#).
- Rules are added using the [getConfiguration\(GraphContext context\)](#) method. This method is inherited from the [OCPsoft Rewrite](#) interface `org.ocpssoft.rewrite.config.ConfigurationProvider`. Rules are discussed in more detail later.
- If other rules must execute after or before the rules in this provider, you must provide the list of `WindupRuleProvider` classes using the **getExecuteAfter()** or **getExecuteBefore()** methods.

The following is an example of a simple Java-based rule add-on.

```

public class ExampleRuleProvider extends WindupRuleProvider
{

```

```

@Override public RulePhase getPhase(){
    return RulePhase.DISCOVERY;
}

// @formatter:off
@Override
public Configuration getConfiguration(GraphContext context)
{
    return ConfigurationBuilder.begin()
        .addRule()
        .when(
            // Some implementation of GraphCondition.
            Query.find(...)....
        )
        .perform(
            ...
        );
}
// @formatter:on
// (@formatter:off/on prevents Eclipse from formatting the rules.)
}

```

Add Rule Code Structure

Like most rule-based frameworks, Windup rules consist of the following:

- Condition: This is the **when(condition)** that determines if the rule should execute.
- Operation: This defines what to **perform()** if the condition is met.

Rules consist of the *when* part, the *perform*, the *otherwise* part, and some others, all of which are optional.

when()

```

.when(Query.fromType(XmlMetaFacetModel.class))

```

In the `.when()` part, the rule typically queries the graph, using the `Query` API. Results of the query are put on variables stack (`Variables`), many times indirectly through the querying API.

Other way to fill a when part is subclassing the `GraphCondition`. Actually, `Query` also implements it, and is only a convenient way to create a condition.

Last noticable but important feature is the ability to use [Gremlin](#) queries. See [Gremlin docs](#) for reference manual.

perform()

```

.perform(
    new AbstractIterationOperation<XmlMetaFacetModel>(XmlMetaFacetModel.class, "xml")
    {
        public void perform(GraphRewrite event, EvaluationContext context, XmlMetaFacetModel model)
        {
            // for each model, do something
        }
    }
)

```

In the `.perform()` part, the rule typically iterates over the items of interest (Java files, XML files, querying services, ...), processes them, and writes the findings into the graph.

For that, various operations are available. These are subclasses of `GraphOperation`. You can also implement your own. See [Rules-Operations\[Rules Operations\]](#) for more info. Again, there are several convenient implementations for constructs like iteration (`Iteration`).

Iteration

```
.perform(  
    Iteration.over(XmlMetaFacetModel.class, "xmlModels").as("xml")  
    .when(...)  
    .perform(  
        new AbstractIterationOperation<XmlMetaFacetModel>(XmlMetaFacetModel.class, "xml"){  
            public void perform(GraphRewrite event, EvaluationContext context, XmlMetaFacetModel  
xmlFacetModel)  
            {  
            }  
        })  
    .otherwise(  
        new AbstractIterationOperation<XmlMetaFacetModel>(XmlMetaFacetModel.class, "xml"){  
            public void perform(GraphRewrite event, EvaluationContext context, XmlMetaFacetModel  
payload)  
            { ... }  
        })  
    .endIteration()
```

Nested iterations

TO_DO: Provide information about nested iterations

otherwise

Windup rules inherit the rule constructs from OCP Rewrite. For example, `.otherwise()` Gives you a chance to perform something in case the conditions in `.when()` return false (e.g. they do not match anything). For more information, see [OCP Rewrite web](#).

```
.otherwise(  
    new AbstractIterationOperation<XmlMetaFacetModel>(XmlMetaFacetModel.class, "xml")  
    {  
        public void perform(GraphRewrite event, EvaluationContext context, XmlMetaFacetModel model)  
        {  
            // for each model, do something alternate  
        }  
    }  
)
```

Where

TO_DO: Provide information about the where clause

Metadata

Rules can specify metadata. Currently, the only appearing in some rules, and not actually used, is `RuleMetadata.CATEGORY`.

```
.withMetadata(RuleMetadata.CATEGORY, "Basic")
```

`.withMetadata()` is basically putting key/value to a `Map<Object, Object>`.

Available utilities

For a list of what key services and constructs can be used in the rule, see [Available Rules Utilities](#).

Basic Rule Execution Flow Patterns

TO_DO: Need some links to github to the respective example files.

operation(); - single operation

When no iteration or condition is needed.

Example: [CopyJavaConfigToGraphRuleProvider](rules-
java/src/main/java/org/jboss/windup/rules/apps/java/config/CopyJavaConfigToGraphRuleProvider.java)

```
return ConfigurationBuilder.begin()
    .addRule()
    .perform(new GraphOperation(){
        @Override
        public void perform(GraphRewrite event, EvaluationContext context){
            ...
        }
    });
```

if(...){ operation(); } - single conditional operation

```
return ConfigurationBuilder.begin()
    .addRule()
    .when( ... )
    .perform(new GraphOperation(){
        @Override
        public void perform(GraphRewrite event, EvaluationContext context){
            ...
        }
    });
```

for(FooModel.class){ ... } - Single iteration

For simple iterations, `IteratingRuleProvider` can be used, which makes the `perform` a bit less verbose:

```
public class ComputeArchivesSHA512 extends IteratingRuleProvider<ArchiveModel>
{
    public ConditionBuilder when() {
        return Query.find(ArchiveModel.class);
    }

    // @formatter:off
    public void perform( GraphRewrite event, EvaluationContext context, ArchiveModel archive ){
        try( InputStream is = archive.asInputStream() ){
            String hash = DigestUtils.sha512Hex(is);
            archive.asVertex().setProperty(KEY_SHA512, hash);
        }
        catch( IOException e ){
            throw new WindupException("Failed to read archive: " + archive.getFilePath() +
                "\n Due to: " + e.getMessage(), e);
        }
    }
    // @formatter:on

    @Override public String toStringPerform() { return this.getClass().getSimpleName(); }
}
```

if(...){ for(...) } - conditional iteration

```
return ConfigurationBuilder.begin()
    .addRule()
    .when(
        new GraphCondition(){ ... }
    ).perform(
        Iteration.over(ArchiveModel.class)
        .perform( ... )
    )
```

for(...){ if(...){ ... } } - iteration with a condition


```

return ConfigurationBuilder.begin()
.addRule()
.when(
    // Stores all ArchiveModel's into variables stack, under that type.
    Query.find(ArchiveModel.class)
).perform(
    Iteration.over(ArchiveModel.class) // Picks up ArchiveModel's from the varstack.
    .when(new AbstractIterationFilter<ArchiveModel>(){
        @Override
        public boolean evaluate(GraphRewrite event, EvaluationContext context, ArchiveModel
payload)
        {
            return payload.getProjectModel() == null;
        }
        @Override
        public String toString()
        {
            return "ProjectModel == null";
        }
    })
    .perform( ... )

```

`for(...){ for(...) { ... } }` - nested iterations

Create a Basic Java-based Rule Add-on

You can create a rule using Java or XML. This topic describes how to create a rule add-on using Java.

Prerequisites

- You must [Install Windup](#).
- Be sure you [Install and Configure Maven](#).
- Before you begin, may want 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://ocpssoft.org/rewrite/>
 - The JavaDoc for the Windup API is located here: <http://windup.github.io/windup/docs/javadoc/latest/>

NOTE

Working examples of Java-based rules can be found in the Windup quickstarts GitHub repository located here: <https://github.com/windup/windup-quickstarts>

Create a Rule Add-on

Create a Maven Project

Create a new Maven Java Project. These instructions will refer to the project folder location with the **replaceable** variable 'RULE_PROJECT_HOME'. Modify the project `pom.xml` file as follows

1. Add the following properties:

```

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
  <version.windup>2.0.0.VERSION</version.windup>
  <version.forge>2.12.1.Final</version.forge>

```

```
</properties>
```

2. Add a dependency management section for the Windup BOM.

```
<dependencyManagement>
  <dependencies>
    <!-- Windup BOM -->
    <dependency>
      <groupId>org.jboss.windup</groupId>
      <artifactId>windup-bom</artifactId>
      <version>${version.windup}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

3. Add a `<dependencies>` section to include Windup, rulesets, and test dependencies required by your rule add-on. Windup is a Forge/Furnace based application and has a modular design, so the dependencies will vary depending on the Windup APIs used by the rule. For more information on Windup dependencies, see [Windup Dependencies](#).

The following are examples of some dependencies you may need for your rule add-on.

```
<!-- Windup API dependencies -->
<dependency>
  <groupId>org.jboss.windup.graph</groupId>
  <artifactId>windup-graph</artifactId>
  <classifier>forge-addon</classifier>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.jboss.windup.config</groupId>
  <artifactId>windup-config</artifactId>
  <classifier>forge-addon</classifier>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.jboss.windup.ext</groupId>
  <artifactId>windup-config-groovy</artifactId>
  <classifier>forge-addon</classifier>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.jboss.windup.utils</groupId>
  <artifactId>utils</artifactId>
  <classifier>forge-addon</classifier>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.jboss.windup.reporting</groupId>
  <artifactId>windup-reporting</artifactId>
  <classifier>forge-addon</classifier>
  <scope>provided</scope>
</dependency>

<!-- Dependencies on other rulesets -->
<dependency>
  <groupId>org.jboss.windup.rules.apps</groupId>
  <artifactId>rules-java</artifactId>
  <classifier>forge-addon</classifier>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.jboss.forge.furnace.container</groupId>
  <artifactId>cdi-api</artifactId>
  <scope>provided</scope>
</dependency>
<dependency>
```

```

<groupId>org.jboss.forge.furnace.container</groupId>
<artifactId>cdi</artifactId>
<classifier>forge-addon</classifier>
<scope>provided</scope>
</dependency>

<!-- Test dependencies -->
<dependency>
  <groupId>org.jboss.forge.furnace.test</groupId>
  <artifactId>furnace-test-harness</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.jboss.forge.furnace.test</groupId>
  <artifactId>arquillian-furnace-classpath</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <type>jar</type>
</dependency>

<dependency>
  <groupId>org.jboss.windup.exec</groupId>
  <artifactId>windup-exec</artifactId>
  <classifier>forge-addon</classifier>
  <scope>test</scope>
</dependency>

```

4. Add the `<plugins>` section to make it a Forge add-on.

```

<plugins>
<!-- The following plugins make this artifact a Forge add-on. -->
<plugin>
  <groupId>org.jboss.forge.furnace</groupId>
  <artifactId>furnace-maven-plugin</artifactId>
  <version>${version.forge}</version>
  <executions>
    <execution>
      <id>generate-dot</id>
      <phase>prepare-package</phase>
      <goals> <goal>generate-dot</goal> </goals>
      <configuration> <attach>true</attach> </configuration>
    </execution>
  </executions>
</plugin>
<plugin>
  <artifactId>maven-jar-plugin</artifactId>
  <executions>
    <execution>
      <id>create-forge-addon</id>
      <phase>package</phase>
      <goals> <goal>jar</goal> </goals>
      <configuration>
        <classifier>forge-addon</classifier>
      </configuration>
    </execution>
  </executions>
</plugin>
</plugins>

```

Create the Java RuleProvider

1. Add `beans.xml` file to `META-INF/` dir in resources - typically, `$project-dir/src/main/resources/META-INF/beans.xml` . This file can be empty. It tells CDI to scan your add-on for CDI beans.
2. Within your Maven project, create a Java class that extends the `WindupRuleProvider` class. It is suggested that you end the name of the class with `RuleProvider` . For example:

```
public class MyCustomRuleProvider extends WindupRuleProvider
{
}
```

3. If the rule provider must run in a phase other than the default `MIGRATION_RULES` phase, override the phase using the `getPhase()` method. For example, you may want the rule provider to perform a task during the initial **DISCOVERY** phase, for example, to ignore files with a specific name or extension.

```
@Override
public RulePhase getPhase() {
    return RulePhase.DISCOVERY;
}
```

For more information about rule phases, see [Rules Execution Lifecycles](#).

4. To control the order in which the rule is executed, implement the `getExecuteBefore()` or `getExecuteAfter()` method.

```
List<Class<? extends WindupRuleProvider>> executeBeforeList = new ArrayList<>();

@Override
public List<Class<? extends WindupRuleProvider>> getExecuteBefore()
{
    return executeBeforeList.add(RuleToFireBefore.class);
}
```

```
List<Class<? extends WindupRuleProvider>> executeAfterList = new ArrayList<>();

@Override
public List<Class<? extends WindupRuleProvider>> getExecuteAfter()
{
    return executeAfterList.add(RuleToFireAfter.class);
}
```

5. Finally, add the rule or rules to the rule provider.

- High-level Conditions and Operations

The following is a specific high-level rule which uses high-level conditions (`JavaClass`) and operations (`Classification`). See the documentation of those conditions and operations for the details.

```
@Override
public Configuration getConfiguration(GraphContext context)
{
    return ConfigurationBuilder.begin()
        .addRule()
        .when(

            JavaClass.references("weblogic.servlet.annotation.WLServlet").at(TypeReferenceLocation.ANNOTATION)
        )
        .perform(
            Classification.as("WebLogic @WLServlet")
                .with(Link.to("Java EE 6 @WebServlet", "https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/index.html"))
                .withEffort(0)
                .and(Hint.withText("Migrate to Java EE 6 @WebServlet.").withEffort(8))
        );
}
```

For more examples of rule providers, see the [BaseConfig.java](#) rule.

- Low-level Conditions and Operations

As you can see, the conditions and operations above are Java-specific. They come with the `Java Basic` ruleset. The list of existing rulesets will be part of the project documentation. Each ruleset will be accompanied with a documentation for its ``Condition`s` and ``Operation`s` (and also ``Model`s`).

These high-level elements provided by rulesets may cover majority of cases, but not all. Then, you will need to dive into the mid-level Windup building elements.

- Mid-level Conditions and Operations

Install the Java-based Rule Add-on

The easiest and fastest way to build the rule add-on, install it into the local Maven repository, and install it into Windup as a rule add-on is to use the Windup `addon-build-and-install` command.

- If you have not started Windup, follow the instructions to [Execute Windup](#).
- At the Windup console prompt, enter the `addon-build-and-install` command:

```
addon-build-and-install --projectRoot RULE_PROJECT_HOME
```

- You should see the following result.

```
***SUCCESS*** Addon MyCustomRuleProvider:::2.0.0.VERSION was installed successfully.
```

Test the Java-based Rule Add-on

Test the Java-based rule add-on 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: QUICKSTART_HOME/windup-reports-java/index.html
```

For more information and examples of how to run Windup, see: [Execute Windup](#)

Review the Output Report

1. Open `OUTPUT_REPORT_DIRECTORY/index.html` file in a browser.
2. You are presented with an Overview page containing the application profiles.
3. Click on the application link to review the detail page. Check to be sure the warning messages, links, and story points match what you expect to see.

TBD.

- Models
 - [Frames](#)
 - [Windup Models](#)
 - Rules
 - [OCPSoft Rewrite](#)
 - Conditions, Operations
 - Variables
 - Inter-rule action
 - [Inter-rule dependency](#)
 - Short IDs - [WINDUP-216](#)
-

Create and Test XML Rules

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.

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://ocpssoft.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>.

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!

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>

```

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](#)

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](#).

XML Rule - When Condition Syntax

Conditions allowed in the `when` portion of a rule must extend [GraphOperaton](#) 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](#)

javaclass Syntax

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>

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"/>
```

project Syntax

Summary

Use the `project` element to find information in project files. For a better understanding of the `project` condition, see the JavaDoc for the [Project](#) class.

The following is an example of a rule that tests for an `xmlfile` .

```
<rule>
  <when>
    <project>
      <artifact groupId="junit" artifactId="junit" from="MyEjbRule" to="???"/>
    </project>
  </when>
  <perform>
    <hint title="Title for Hint for Project">
      <message>Descriptive message</message>
    </hint>
  </perform>
</rule>
```

Construct a project Element

project Element Attributes

The `project` element does not have any attributes.

project Element Child Elements

artifact

The namespace to referenced in XML files. This element contains the following attributes:

- `groupId="PROJECT_GROUP_ID"`
Match on the project `<groupId>`
- `artifactId="PROJECT_ARTIFACT_ID"` Match on the project `<artifactId>`
- `from="VARIABLE_NAME"`
Begin the search query with the filtered result from a previous search identified by its as `VARIABLE_NAME`.
- `to = TO_DO`: complete the project artifact 'to' element syntax

filecontent Syntax

`TO_DO`: complete the filecontent element syntax

Use the `filecontent` element to find information in file content. For a better understanding of the `filecontent` condition, see the JavaDoc for the [FileContent](#) class.

Use the `filecontent` element to test for content within files. The following is

an example of a rule that tests for `filecontent`.

XML Rule - Perform Action Syntax

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. 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](#)

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>
```

Construct a classification Element

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>
```

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/context.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>
```

Construct a link Element

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"
```

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>
```

Construct a hint Element

hint Element: Attributes

message="MESSAGE"

A message describing the migration hint

Example:

```
message=""
```

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"/>
```

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>
```

Construct an xslt Element

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"/>
```

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>
```

Construct a lineitem Element

lineitem Element: Attributes

message="MESSAGE"

A `lineitem` message

Example:

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

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>
```

Construct an iteration Element

iteration Element: Attributes

over="NUMBER"

??

Example:

```
over="2"
```

iteration Element: Child Elements

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

Test an XML Rule in Windup

Add the Rule to Windup

A Windup rule is installed simply by copying the rule to the appropriate Windup folder. Windup scans for rules in the following locations:

- The `WINDUP_HOME/rules/` folder.

This is the folder where you run the Windup executable. See [About the WINDUP_HOME Variable](#) for details.

- The `${user.home}/.windup/rules/` folder.

This folder is created by Windup the first time you execute Windup. It is used to track rules and add-ons and to contain the Windup log.

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


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
```

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 the [Windup source code](#) GitHub repository and the Windup quickstarts [GitHub repository](#) or [latest release ZIP download](#).

Core Developer topics

Windup Bootstrap

DRAFT

This section describes how Windup starts up, finds the resources, loads the rules, and executes them.

Boot

1. `@Inject WindupProcessor`

```
CDI instantiates `WindupProcessorImpl`
```

2. Create `WindupProcessorConfig`
3. `WindupProcessorImpl.execute(WindupProcessorConfig)`

```
WindupProcessorImpl instantiates `@Inject GraphContext graphContext`, which holds  
graph-related objects.
```

4. `GraphContextFactoryImpl` :

```
`create()`  
`@Produces GraphContext produceGraphContext()`
```

GraphContext

RuleSubset

Frames creation

In the `GraphContextImpl` constructor. For additional method Handlers, see

```
final Module addModules = new Module()
```

```

{
  @Override
  public Graph configure(Graph baseGraph, FramedGraphConfiguration config)
  {
    config.setFrameClassLoaderResolver(fclr);
    // Java Handlers
    config.addMethodHandler(new FrameMapHandler());
    config.addMethodHandler(new MapInPropertiesHandler());
    config.addMethodHandler(new MapInAdjacentPropertiesHandler());
    return baseGraph;
  }
};

```

Classloading Notes

DRAFT

Page contains temporary notes, to be edited.

This covers the specific situations when the Windup core developer needs to look up certain runtime classes. Does not relate to rules authoring.

Forge add-ons transitive dependencies

Let A depend on B depend on C. Depending on scope of C in B's POM:

```
<scope>provided</scope>
```

`provided` will NOT export C, so A depending on B won't see C's classes. Only `compile` scope is exported. `compile` will export C, so A depending on B will also see C's classes.

- `provided` == import
- `compile` == import & export
- `optional` == optional, not installed automatically
- `runtime` == export only

Finding classes

```

AddonRegistry.getServices(MyServiceInterface.class)

// To get the AddonRegistry, use Windup's FurnaceHolder:
FurnaceHolder.getFurnace().getAddonRegistry().getServices(MyServiceInterface.class);

```

or

```
@Inject Imported<MyServiceInterface.class> imported;
```

Instantiating

If you want something from the current add-on, you can simply `@Inject` the type you wish to use. Alternately you can `@Inject` an `Imported<T>` instance, or also `@Inject` the `AddonRegistry` itself.

Which ClassLoader is used? Always the one of the current code?

Depends, if you are in a Furnace Service, then yes, it will use the current code's add-on ClassLoader.

Reading annotations from classes

Since classes can be proxied by Forge/Furnace, it's safer to read annotations

using Annotations API :

```
Rules annotation = Annotations.getAnnotation(this.getClass(), Rules.class);
```

Supposedly, Furnace proxies should also be able to handle such call:

```
Rules annotation1 = ((FurnaceProxy) proxy).getOriginalClass().getAnnotation(Rules.class);
```

TO_DO: Find out how exactly this is done.

How do I know if I am in a Furnace service?

Every furnace service is wrapped in a proxy. If it came from another add-on (not your current add-on) and was not instantiated with new Blah(), you are in a furnace service. So if it was @Inject`ed or retrieved via `addonRegistry.getServices(...)

```
(16:32:40) jsightler: ozizka: I do like a Resource.open(...) type API -- that sounds like a good idea.
(16:32:57) LincolnBaxter: ozizka: unless you use the TCCL (which you shouldn't in a modular
environment), then using the class's own classloader is the best solution for that situation
(16:33:41) LincolnBaxter: ozizka: using a shared Resource.open() API will actually be sort of a no-op,
and just introduce another layer, since you generally need to be able to specify a classloader anyway
(16:33:53) jsightler: lincolnthree: Well...
(16:34:09) LincolnBaxter: ozizka: unless I'm misunderstanding the point of this
(16:34:10) jsightler: lincolnthree: In windup we frequently need resources from the caller's add-on
(16:34:18) LincolnBaxter: ozizka: ah wait
(16:34:27) LincolnBaxter: ozizka: the *caller*'s classloader
(16:34:35) LincolnBaxter: ozizka: not the CL of the current operation
(16:34:44) LincolnBaxter: ozizka: that's different. hmm
(16:35:11) ozizka-FN: Right
(16:35:18) ozizka-FN: Actually that's what I wanted to ask -
(16:35:21) jsightler: We deal with this in an ad-hoc way at the moment... it might be nice to have a
centralized service that does it.
(16:35:25) ozizka-FN: which classloader is used?
(16:35:33) ozizka-FN: Always the one of the current code?
(16:35:44) ozizka-FN: Implicitly
(16:35:54) LincolnBaxter: ozizka: depends, if you are in a Furnace Service, then yes, it will use the
current code's add-on's classloader
(16:36:00) ozizka-FN: Ah
(16:36:11) ozizka-FN: How do I know if I am in a Furnace service?
(16:36:14) LincolnBaxter: ozizka: because every furnace service is wrapped in a proxy
(16:36:31) LincolnBaxter: ozizka: basically if it came from another add-on (not your current add-on),
you are in a furnace service
(16:36:43) LincolnBaxter: ozizka: and if you didn't instantiate with new Blah()
(16:37:00) LincolnBaxter: ozizka: so if it was @Injected or retrieved via addonRegistry.getServices(...)
(16:37:43) LincolnBaxter: jsightler: ozizka: mbriskar: why do we need the classloader of the calling
add-on? just curious
(16:37:54) ozizka-FN: So here we would use TCCL right?
LincolnBaxter lincolnthree
(16:38:28) ozizka-FN: lincolnthree: To load it's resources
(16:38:31) ozizka-FN: from other add-on
(16:38:32) ozizka-FN: .
(16:38:35) ozizka-FN: Or is there other way?
(16:38:42) LincolnBaxter: ozizka: that's a vague statement ;)
(16:38:48) LincolnBaxter: ozizka: where is *here*?
(16:38:56) LincolnBaxter: ozizka: the best way is to pass the actual classloader you want to use
(16:39:01) ozizka-FN: For this use case. For the Resource.open()
LincolnBaxter lincolnthree
(16:39:21) ozizka-FN: lincolnthree, right, but well, putting classloading code into rules...
(16:39:25) ozizka-FN: Not user friendly
```

Resources loading

```
(16:48:30) ozizka-FN: lincolnthree: Just a note - the potential collisions caused by add-on's CL span is
not considered as a risk?
(16:49:25) ozizka-FN: I mean, it could load the file from it's deps.
(16:49:36) ozizka-FN: * file = resource
(16:49:50) LincolnBaxter: ozizka1: so the way it works is actually very controlled
(16:50:06) LincolnBaxter: ozizka1: classloader will always attempt to load its own resources before
```

loading from another add-on
 (16:50:23) LincolnBaxter: ozizka1: at which point the order is determined by the order of the add-on dependency in the POM

(16:53:26) ozizka-FN: Q: Unwrapping proxies caused the code in invoke() not actually being called?
 (16:53:32) ozizka-FN: Since the method calls were not intercepted?
 (16:54:09) LincolnBaxter: ozizka1: exactly :)

(16:54:10) ozizka-FN: lincolnthree, is that impl a real TCCL or just TCCL-like concept?
 (16:54:29) LincolnBaxter: ozizka1: what do you mean "real" ?
 (16:54:51) ozizka-FN: Not sure about how TCCL is implemented, I thought you'd need to call something like
 (16:54:52) ozizka-FN: Thread.currentThread().setContextClassLoader(classLoader);
 (16:54:56) LincolnBaxter: ozizka1: also if you are interested in where the resource loading / ordering is controlled —>
<https://github.com/forged/furnace/blob/master/container/src/main/java/org/jboss/forged/furnace/impl/modules/AddonModuleLoader.java#L194>
 (16:55:01) jsightler: lincolnthree: I wondered that too :)
 (16:55:30) LincolnBaxter: ozizka1: jsightler, actually it does:
<https://github.com/forged/furnace/blob/master/proxy/src/main/java/org/jboss/forged/furnace/proxy/ClassLoaderInterceptor.java#L103>
 (16:55:55) LincolnBaxter: ozizka1: jsightler: <https://github.com/forged/furnace/blob/master/container-api/src/main/java/org/jboss/forged/furnace/util/ClassLoaders.java#L27>

(16:56:57) ozizka-FN: Hm, ok so I guess if we dug deeper, it would eventually come to Thread.currentThread().setContextClassLoader(classLoader); ?
 (16:57:15) LincolnBaxter: ozizka1: yep: <https://github.com/forged/furnace/blob/master/container-api/src/main/java/org/jboss/forged/furnace/util/SecurityActions.java#L71>

Furnace Proxies:

<https://github.com/forged/furnace/blob/master/proxy/src/main/java/org/jboss/forged/furnace/proxy/Proxies.java#L15>

What classloader is used by Freemarker to load resources?

`org.jboss.windup.reporting.freemarker.FurnaceFreeMarkerTemplateLoader`

Connect to the Graph via Rexster

There is a specific add-on for running Rexster along your graph with groupId **org.jboss.windup.rexster** and artifactId **Rexster**.

Running Rexster Within Your Test

Deploy the Rexster add-on by adding the Rexster add-on to the dependencies.

```
@Deployment
@Dependencies({
    ....
    @AddonDependency(name = "org.jboss.windup.rexster:rexster", version = "2.0.0-SNAPSHOT"),
    ....
})
```

Then put some debug breakpoint in the specific phase of graph you are interested in and browse <http://localhost:8182/> to enjoy the graph!

Username and password: rexster

Run Rexster within Windup

Make sure the Rexster add-on is deployed in Furnace and is accessible on <http://localhost:8182/>.

```
username: rexster
password: rexster
```

Decompiling

The Decompiler API

DRAFT

- The 3 main use cases for the decompiler API are to process:
 1. Directories
 2. JARS or Archives
 3. Class files
- The `DecompilerConfig` should also contain `Filter<String>` (or maybe `Filter<TypeReference>`) to decide whether to decompile the class or not.
- `windup.ProcyonConfiguration` should extend abstract `windup.DecompilerConfig<T>`, which would contain the configurables common to all decompilers. In case of Procyon, `outputDir` would be delegated to `procyon.DecompilerSettings`.

Procyon

Config

- `windup.ProcyonConfiguration` contains `procyon.DecompilerSettings`.
- `procyon.DecompilerSettings` contains `outputDir` (the others are not much important). TO_DO: Make that a parameter to method call?

TypeReference

TO_DO: Check how we get info about Java files; perhaps cache the `TypeReference` results - if worth it?

Loader

- `JarTypeLoader` - iterates a `JarFile`, expects a `.jar` File as input.
- `ClasspathTypeLoader` - loads classes from a list of classes.
- `ClasspathTypeLoader("some/path:other/path")`
- `ClasspathTypeLoader()` → `sysprop("java.class.path")` : `sysprop("sun.boot.class.path")`
- `InputTypeLoader`
- `CompoundTypeLoader` - 2 typeloader, queried in succession

TO_DO: Create a simple `FileSystemTypeLoader`.

Dependencies (Forge add-ons)

DRAFT

Based on [this](#) discussion.

Maven Add-on Structure

The following is the typical structure of Maven `add-on` modules.

```
my-add-on-parent
+--- my-add-on - jar, classifier: forge-addon
+--- my-add-on-api - jar
+--- my-add-on-impl - jar
+--- my-add-on-tests - jar
```

The `add-on` POM file can declare dependencies on other forge-addons. To

prevent exporting the add-on to dependent modules, use `<optional>true</optional>` .

```
<dependency>
  <groupId>org.jboss.forge.furnace.container</groupId>
  <artifactId>cdi</artifactId>
  <classifier>forge-addon</classifier>
  <optional>true</optional>
</dependency>
```

The add-on POM files may also declare a dependency on typical Maven artifacts. To prevent exporting the add-on to dependent modules, use `<optional>true</optional>` .

```
<dependency>
  <groupId>com.thinkaurelius.titan</groupId>
  <artifactId>titan-lucene</artifactId>
  <version>${version.titagraph.lucene}</version>
</dependency>
```

One more question regarding dependencies. Is it advisable to depend on the `forge-addon` artifact from add-on's subparts?

Add-ons should reference the `<classifier>forge-addon</classifier>` artifact from other add-ons.

Add-on sub-parts

Add-on sub-parts declare dependency preferably on forge add-on APIs, not on the add-ons themselves, wherever possible (wherever an add-on has an explicit API). These dependencies must be marked as `provided` scope.

```
<dependency>
  <groupId>org.jboss.forge.furnace.container</groupId>
  <artifactId>cdi-api</artifactId>
  <classifier>forge-addon</classifier>
  <scope>provided</scope>
</dependency>
```

They may instead depend on the primary add-on dependency, e.g. `forge-addon` . The latter makes the POM files simpler, but can be confusing, because the add-on dependency still needs to be declared in the depending add-on's POM.

```
<dependency>
  <groupId>org.jboss.forge.furnace.container</groupId>
  <artifactId>cdi-api</artifactId>
</dependency>
```

Declaring a `forge-addon` dependency anywhere else than the `forge-addon` pom doesn't actually cause Furnace to establish an add-on dependency. I.e. classes from the dep won't be available unless you declare it in the `forge-addon`'s pom.

Add-on's sub-parts may also declare dependencies on other normal maven dependencies, and these are treated as normal.

Add-on depends on API , scope `compile` > Add-on depends on

NOTE

Impl , scope compile with `<optional>true</optional>` - prevents exporting this dependency to consumers of the add-on.

Test dependencies

For test dependencies on add-ons: Any add-on/sub-part requiring an add-on for testing purposes should use `<scope>test</scope>` .

```
<dependency>
  <groupId>org.jboss.windup.graph</groupId>
  <artifactId>windup-graph</artifactId>
  <version>${project.version}</version>
  <classifier>forge-addon</classifier>
  <scope>test</scope>
</dependency>
```

Dependencies between sub-parts within the same add-on

Subpart may declare dependency on other subpart. E.g. impl typically depends on api . In that case, the scope should be set appropriately for the given situation - typically provided or compile scope.

```
<dependency>
  <groupId>org.jboss.windup.graph</groupId>
  <artifactId>windup-graph-api</artifactId>
  <scope>compile</scope>
</dependency>
```

Frames Extensions

Windup has several Frames extensions to satisfy its needs.

Multiple Types in a Vertex

Frames stores only one type in the *type* property. To specify multiple types, use the '|' separator. All sub-parts are indexed.

Map Handler

TBD

To store a map in a single vertexes properties.

Map

To store a map in as adjacent vertices.

```
@TypeValue("MapModelMain")
public interface MapMainModel extends WindupVertexFrame
{
    @AdjacentMap(label = "map") void setMap(Map<String, MapValueModel> map);
    @AdjacentMap(label = "map") Map<String, MapValueModel> getMap();
}
```

See the `FrameMapHandlerTest` .

List handler.

TBD.

To store a list of strings in a single vertexes properties. The keys will be 0, 1, 2, 3, ...

List

`WindupVertexListModel` offers a generic model for lists of other models, which are stored as adjacent vertices.

Internal API Features

Find the `RuleProvider` that provided a `Rule`

A `Rule` implements `Context`, which stores metadata. The following example gets the `RuleProvider` from the `Rule` context.

```
WindupRuleProvider ruleProvider =  
    (WindupRuleProvider) context.get(RuleMetadata.RULE_PROVIDER);
```

For more information about the metadata stored in the context, see [Rules Metadata](#)

In-memory Frames

```
GraphService<FooModel> fooModelService = context.getService(FooModel.class);  
FooModel inMemoryModel = fooModelService.create();
```

Performance measurement

```
ExecutionStatistics.get().begin("Process-01");  
// ... your code to be timed goes here  
ExecutionStatistics.get().end("Process-01");
```

The results goes to `stats/detailed_stats.csv`.

Logging

DRAFT

This page describes how to log from the Java-based rules.

Textual logging

Windup uses `java.util.logging`. Convenient way to get the logger is:

```
private static final Logger LOG = Logging.get(MyClass.class);
```

Displaying Progress

IterationProgress

```
return ConfigurationBuilder.begin().addRule()  
    .when(Query.find(ArchiveModel.class))  
    .perform(UnzipArchiveToOutputFolder.unzip())  
    .and(IterationProgress.monitoring("Unzipped archive: ", 1))  
    .and(Commit.every(1))  
    );
```

Variables Stack

DRAFT

The `Variables` class stores the variables which are available in rules, EL expressions, and FreeMarker and XSLT templates.

Accessing variables

From a Java-based rule

TO_DO: Describe how to access variables from a Java-based rule.

From a Groovy-based rule

TO_DO: Describe how to access variables from a Groovy-based rule.

From an EL expression

TO_DO: Describe how to access variables from an EL expression.

From a FreeMarker

TO_DO: Describe how to access variables from a FreeMarker.

From XSLT template

TO_DO: Describe how to access variables from an XSLT template.

Accessing variables as a flat Map

TO_DO: Describe how to access variables from a Map.

Git Rebasing

This section describes a few ways you can pull in the code to rebase to the latest code base.

Approach Suggested on GitHub

This is the approach suggested on GitHub.

1. Checkout the source:

```
git checkout -b lincolnthree-WINDUP-133 master
git pull https://github.com/lincolnthree/windup.git WINDUP-133
```

2. Run the tests to make sure it builds.

```
mvn clean install
```

3. If they pass, merge and push them to your repository.

```
git checkout master
git merge --no-ff lincolnthree-WINDUP-133
git push origin master
```

Script Provided by the Windup Development Team

Ondra Zizka wrote the following script to bring down a series of pull requests into a single branch. It then pushes that branch to upstream master-ignore.

1. Bring down the pulls into a single branch.

```
pull() {
  cmd="git fetch upstream master "
  for PR in "$@"
  do
    cmd="$cmd pull/$PR/head:pullRequest$PR"
  done

  $cmd

  git branch -D pulls
  git checkout master
  git rebase upstream/master
  git checkout -b pulls
```

```

for PR in "$@"
do
  git checkout pullRequest$PR
  git rebase pulls
  git checkout pulls
  git merge pullRequest$PR
  git branch -D pullRequest$PR
done
}

```

2. Fetch using the one pull request.

```
git fetch upstream master pull/$PR/head:pullRequest$PR
```

3. The branch is now updated with the latest source.

Rules topics

Available Rule Utilities

Programmatically Access the Graph

Note: Needs update. This is out of date!

(Lower Level API, to cover cases not provided by high level API)

This topic describes how to to programmatically access or update graph data when you create a Java-based rule add-on.

Query the graph

There are several ways - including Query API, Gremlin support, or GraphService methods.

Query the Graph Within the .when() method

Building a rule contains the method when(), which is used to create a **condition**. Vertices that fulfill the condition, are passed to the perform() method.

For the queries in the when() method, class Query is used. There are several methods which you can use to specify the condition. For example: * **find()** specifies the Model type of the vertex * **as()** method specifies the name of the final list, that is passed to the perform() method * **from(String name)** starts the query not on the all vertices, but only on the vertices already stored in the the given **name** (used to begin query on the result of the other one) * **withProperty()** specify the property value of the given vertex

The following are examples of simple queries.

Return a list of archives

```
Query.find(ArchiveModel.class)
```

```
Query.find(ApplicationReportModel.class).as(VAR_APPLICATION_REPORTS)
```

Iteration

```

ConfigurationBuilder.begin().addRule()
    .when(
        GraphSearchConditionBuilderGremlin.create("javaFiles", new ArrayList())
        .V().framedType( JavaFileModel.class ).has("analyze")
    )
    .perform(
        // For all java files...
        Iteration.over("javaFiles").var("javaFile").perform(

```

Nested Iteration

```

code.java
// For all java files...
Iteration.over("javaFiles").var("javaFile").perform(
    // A nested rule.
    RuleSubset.evaluate(
        ConfigurationBuilder.begin().addRule()
        .when(...)
        .perform(
            Iteration.over("regexes").var(RegexModel.class, "regex").perform(
                new AbstractIterationOperator<RegexModel> ( RegexModel.class, "regex" ) {
                    public void perform( GraphRewrite event, EvaluationContext context, RegexModel regex
                ) {
                    //...
                }
            )
        )
    ).endIteration()
) // perform()
)

```

Modify Graph Data

For more custom operations dealing with Graph data that are not covered by the `Query` mechanism, use the `GraphService`.

```

GraphService<FooModel> fooService = new GraphService<FooModel>(graph,FooModel.class);

List<FooModel> = fooService.findAll();
FooModel = fooService.create();

// etc ...

```

`GraphService<>` can also be used to query the graph for models of the specified type:

```

FooModel foo = new GraphService<>(graphContext, FooModel.class).getUnique();

```

```

FooModel foo = new GraphService<>(graphContext, FooModel.class).getUniqueByProperty("size", 1);

```

Concepts & Philosophy

TO_DO: - OZIZKA: Can this topic be marked obsolete and be replaced by this one: [Windup Processing Overview ?](#)

Windup is a rule-based tool that allows users to write customized rules based on the needs, constructs, and custom APIs used in their applications.

Individual rules should be decoupled, only expressing what rules should precede and follow, and "communicate" through the graph. Each rule will query the graph database, use the results for locating the candidates of it's interests, process them, and then write the results to the graph database.

Graph database and Models (Frames)

As a graph db implementation, we use [TinkerPop](#) backed by [Titan](#) backed by BerkeleyDB. For explanation of graph database concepts, see [Titan](#).

Like there's ORM (like JPA) for JDBC, TinkerPop's BluePrints has [Frames](#). This allows you to access the graph data in form of your custom Java models (implemented as Java Proxies to the querying).

We use this concept heavily. Each ruleset will likely have it's own models. (But you can opt to use Blueprints API if you like).

See also the list of [Windup Models](#).

Examples of breaking non-trivial workflows into rules

1. Finding all `@Entity`'s which use `org.hibernate` extensions.

TO_DO

2. Finding *MyBatis* DAOs and classes using them.

TO_DO

Creating Rule Operations

An operation is a task which can be invoked from within a rule.

It is created by extending `GraphOperation`.

```
public class FooOperation extends GraphOperation
{
    @Override
    public void perform(GraphRewrite event, EvaluationContext context)
    {
        ...
    }
}
```

Existing operations:

```
org.jboss.windup.config.operation.GraphOperation
org.jboss.windup.config.MavenExampleRuleProvider.1
org.jboss.windup.config.XmlExampleRuleProvider2.1
org.jboss.windup.config.XmlExampleRuleProvider3.1
▼ org.jboss.windup.config.operation.ruleelement.AbstractIterationOperation
    org.jboss.windup.config.JavaExampleRuleProvider.2
    org.jboss.windup.config.WindupConfigurationExampleRuleProvider.2
    org.jboss.windup.config.XmlExampleRuleProvider1.1
    org.jboss.windup.config.XmlExampleRuleProvider1.2
    org.jboss.windup.config.graph.TypeOperation
    org.jboss.windup.reporting.freemarker.FreeMarkerIterationOperation
    org.jboss.windup.reporting.rules.ApplicationReportRenderingRuleProvider.1
    org.jboss.windup.reporting.rules.CreateApplicationReportRuleProvider.1
    org.jboss.windup.reporting.rules.CreateSourceReportRuleProvider.2
    org.jboss.windup.reporting.rules.CssJsResourceRenderingRuleProvider.1
    org.jboss.windup.reporting.rules.RenderSourceReportRuleProvider.1
    org.jboss.windup.rules.apps.java.scan.operation.AddArchiveReferenceInformation
    org.jboss.windup.rules.apps.java.scan.operation.AddClassFileMetadata
    org.jboss.windup.rules.apps.java.scan.operation.ConfigureArchiveTypes
    org.jboss.windup.rules.apps.java.scan.operation.RecurseDirectoryAndAddFiles
    org.jboss.windup.rules.apps.java.scan.operation.UnzipArchiveToTemporaryFolder
    org.jboss.windup.rules.apps.java.scan.provider.AnalyzeJavaFilesRuleProvider.FireASTTypeNameEventsIterationOperator
    org.jboss.windup.rules.apps.java.scan.provider.DiscoverJavaFilesRuleProvider.IndexJavaFileIterationOperator
    org.jboss.windup.rules.apps.java.scan.provider.DiscoverMavenHierarchyRuleProvider.1
    org.jboss.windup.rules.apps.java.scan.provider.DiscoverMavenProjectsRuleProvider.1
    org.jboss.windup.rules.apps.java.scan.provider.DiscoverXmlFilesRuleProvider.1
    org.jboss.windup.reporting.ApplicationReport
    org.jboss.windup.reporting.freemarker.FreeMarkerOperation
    org.jboss.windup.reporting.renderer.RenderGraphRuleProvider.1
    org.jboss.windup.rules.apps.java.blacklist.ASTEventEvaluatorsBufferOperation
```

Rulesets

DRAFT

Example page for design decisions. Could be generated automatically in the future.

What is a Ruleset?

A ruleset is a Windup "plug-in" targetting a specific area of migration (e.g. Spring to Java EE 6 migration). Underhood, it is a set of rules, and everything they might need: Operations and Conditions, Report templates (if needed), and static files (e.g. images, XML or CSV files, etc.). A ruleset may also declare metadata, like ruleset ID, dependencies on other rulesets, etc.

xref:Ruleset-Java-Basic-Ruleset Forge add-on (i.e. a `.jar` file).

Groovy-based ruleset is a directory with `.windup.groovy` script(s) and its static files, possibly in a `.zip` file.

See ? to read about how to create your own Ruleset.

Basic tags

- `app` - denotes the rules which help migrating applications.
- `server` - denotes the rules which help migrating server configuration.
- `java` - denotes the rules treating Java source code.
- `java-ee` - denotes the rules treating Java EE applications or Java EE containers.

Besides that, you may use any custom tag.

Rulesets

Core rulesets

Rulesets distributed with Windup and maintained by the Windup team.

- [Java Basic Ruleset](#)
- Tags: `java`, `app`
- Java EE Applications
- `java-ee`, `app`
- Java EE Servers
- `java-ee`, `server`
- Reporting
 - `CreateApplicationReportIndexRuleProvider`
 - `CreateJavaApplicationOverviewReportRuleProvider`
 - `RenderSourceReportRuleProvider`

Community rulesets

Rulesets contributed by Windup users.

Java Basic Ruleset

DRAFT

This is an example page. Content serves for design decision.

Purpose of this ruleset.

Metadata

ID: myRuleset **Graph prefix:** myRuleset

Rule providers

A definition list (dl/dt/dd) of RuleProviders of given Ruleset.

- **JDKconfigRuleProvider** - Targetted towards the classes distributed with JDK, like `java.lang.`, `java.security.`, `org.xml.*`, etc.
- **FooRuleProvider** - blah
- **BarRuleProvider** - blah
- **Maven Project analysis**
- **Seam 2.0 to CDI**

Models

- JavaClassModel
- AmbiguousJavaClassModel
- AmbiguousReferenceModel<REFERENCETYPE>
- ClassificationModel
- EjbBeanBaseModel
- EjbEntityBeanModel
- EjbMessageDrivenModel
- EjbSessionBeanModel
- HibernateConfigurationFileModel
- HibernateEntityModel
- HibernateMappingFileModel
- HibernateSessionFactoryModel
- InlineHintModel
- JarArchiveModel
- JarManifestModel
- JavaClassFileModel
- JavaClassModel
- JavaMethodModel
- JavaParameterModel
- JavaSourceFileModel
- JavaTypeReferenceModel
- LinkModel

- MavenProjectModel
- PackageModel
- ProjectDependencyModel
- ProjectModel
- PropertiesModel
- SourceFileModel
- SpringBeanModel
- SpringConfigurationFileModel
- WarArchiveModel
- WebXmlModel

Configuration

- **--packages** - Which packages to scan.

Reports

Ruleset Java Classifications and Inline Hints

DRAFT

Some of the most important rules are **classifications** and **inline hints**.

Classifications

Classifications are important because they tell developers very quickly **what** the resource is, so that they can quickly identify resources of importance. For example, say you are responsible for migrating an application from Weblogic to JBoss... wouldn't it be nice to know that project is a Spring project? Wouldn't it be nice to have the Weblogic descriptors identified and called out to us? This is what classifiers do. They identify resources, and add a classification to the resource.

Classifications are displayed both at the 10,000 foot view, main report and also at the resource report level.

- Java Classifications TO_DO: link
- XML Classifications TO_DO: link

Inline Hints

Windup takes the approach of profiling resources and providing syntactically highlighted resource reports. The resource report will show the resource with potentially highlighted lines that may contain inline hints to help the developers migrate a piece of code.

Adding inline hints for Java and XML are as simple as adding Regular Expressions or XPath Expressions to the Windup rules.

Ruleset Java EE Apps

DRAFT

TBD?

DRAFT

TBD ?

Ruleset Reporting

DRAFT

TBD?

XML Ruleset

DRAFT

- XPathMatch
- XsltTransformation

A core ruleset to provide support for XML files.

Metadata

ID: xml **Graph prefix:** xml

Rule providers

A definition list (dl/dt/dd) of RuleProviders of given Ruleset with brief descriptions.

TO_DO: Not sure if all the Xml* really belong here - maybe rather to Java or Java EE?

- **DiscoverXmlFilesRuleProvider** -
- **XmlBaseConfig** -
- **XmlGlassfishConfig** -
- **XmlJbossConfig** -
- **XmlJbossEsbConfig** -
- **XmlJonasConfig** -
- **XmlJppConfig** -
- **XmlJrunConfig** -
- **XmlKnowHowConfig** -
- **XmlOrionConfig** -
- **XmlPersistenceConfig** -
- **XmlResinConfig** -
- **XmlSoa5Config** -
- **XmlSonicEsbConfig** -
- **XmlSpringConfig** -
- **XmlWeblogicConfig** -
- **XmlWebserviceConfig** -
- **XmlWebsphereConfig** -

Models

_Subclasses of WindupVertexFrame .

- DoctypeMetaModel

- XmlFileModel
- XmlTypeReferenceModel
- XsltTransformationModel

Configuration

_Subclasses of TO_DO

Reports

Report files or report parts created by this ruleset.

Windup Models

Windup models are the classes extending WindupVertexFrame. They are used to model the data in the graph database to Java objects.

This is an overview of the most important models. The complete and up-to-date list of models is in [Javadoc](#).

- ▼ AmbiguousReferenceModel
 - AmbiguousJavaClassModel
- ▼ ApplicationModel
 - ApplicationArchiveModel
- ▼ ArchiveModel
 - EarArchiveModel
 - JarArchiveModel
 - WarArchiveModel
 - BlackListModel
 - ClassCandidateTypeModel
 - ClassificationModel
 - DoctypeMetaModel
 - Impl
- ▼ JavaClassModel
 - AmbiguousJavaClassModel
 - JavaMethodModel
 - JavaParameterModel
 - MailserverModel
 - MapMainModel
 - MapValueModel
 - NamespaceMetaModel
 - ProjectDependency
- ▼ ProjectModel
 - MavenProjectModel
- ▼ ReportModel
 - ApplicationReportModel
- ▼ ResourceModel
 - ▼ FileModel
 - Impl
 - JarManifestModel
 - JavaFileModel
 - ▼ PropertiesModel
 - Impl
 - SourceReportModel
 - ▼ XmlResourceModel
 - Impl
 - ▼ FooModel
 - FooSubModel
 - Impl
 - SomeModel
 - WhiteListModel
- ▼ WindupConfigurationModel
 - Impl
 - WindupConfigurationPackageModel
 - XmlMetaFacetModel

Meta Models

- User input
- Rules and Rule Providers metadata

Core Models

FileModel ArchiveModel

Reporting Models

Custom Models (coming from add-ons)

See respective ruleset's documentation.

Create Java Queries

Query for all frames of some Model

```
Query.find(FileModel.class).as("result")
```

Query by property

```
Query.find(FileModel.class)
  .withProperty(FileModel.PROPERTY_IS_DIRECTORY, true)
  .as("res")
```

Gremlin query

```
code.java
Query.find(FileModel.class).pipelined(new QueryGremlinCriterion() {
    public void query( GraphRewrite event, GremlinPipeline<Vertex, Vertex> pipeline ) {
        throw new UnsupportedOperationException( "Not supported yet." ); //To change body of
generated methods, choose Tools | Templates.
    }
})
.withProperty(FileModel.PROPERTY_IS_DIRECTORY, true)
.as("res")
```

Custom query

The `Query` API is just a convenience implementation. For a custom query, implement your own `GraphCondition` :

```
.when(
    new GraphCondition() {
        public boolean evaluate( GraphRewrite event, EvaluationContext context ) {
            event.getGraphContext(); //...
        }
    }
)
```

Rules Metadata

Overview

Draft

Rules metadata are used for ordering, filtering, debugging and performance reasons.

Metadata used by Windup

To control ordering, override `getExecuteAfter()` and `getExecuteBefore()` .

```
public class MyRules extends WindupRuleProvider
{
```

```
public List<Class<? extends WindupRuleProvider>> getExecuteAfter() {
    return asClassList(UpdateVictimsDbRules.class, ComputeArchivesSHA512.class);
}
```

To specify the rule ID, use `.withId("...")`.

```
.addRule()
.perform(...)
.withId("CheckArchivesWithVictims")
```

For other metadata, either use `.withMetadata()` or override `enhanceMetadata()`. To provide categorization for filtering, use `CATEGORY`.

For its metadata, Windup uses the keys defined in `RuleMetadata`:

- [CATEGORY](#) (String): Define which category the rule belongs to, which can be then used to filter the rules to execute (plus all those they depend on). Not implemented yet.
- [ORIGIN](#) (String): The rule origin. Typically a class name.
- [AUTO_COMMIT](#) (boolean): Whether to call database `commit()` after the rule executes.
- [TREAT_EXCEPTIONS_AS_FATAL](#) (boolean): Whether all Exceptions from this Rule are to be treated as fatal. The default is non-fatal.
- [RULE_PROVIDER](#): The `WindupRuleProvider` that originated the rule. This is used internally, not to be specified by rule author.

You can find out more information about rule metadata here: [RuleMetadata](#) Javadoc.

How to define rule metadata

The `RuleMetadata` is defined using the `.withMetadata(Object key, Object value)`.

```
.addRule()
.when(...)
.perform(...)
.withId("CheckArchivesWithVictims")
.withMetadata(RuleMetadata.CATEGORY, ...)
```

In case you want to apply the same metadata to all rules in some `RuleProvider`, override its `enhanceMetadata()` method. The default implementations sets `CATEGORY` to "Uncategorized", `ORIGIN` to its class's name, and `RULE_PROVIDER` to the `RuleProvider` object:

```
public class MyRules extends WindupRuleProvider
{
    public void enhanceMetadata(Context context) {
        if ( ! context.containsKey(RuleMetadata.CATEGORY) )
            context.put(RuleMetadata.CATEGORY, "Uncategorized");
        if ( ! context.containsKey(RuleMetadata.ORIGIN) )
            context.put(RuleMetadata.ORIGIN, this.getClass().getName());
        if ( ! context.containsKey(RuleMetadata.RULE_PROVIDER) )
            context.put(RuleMetadata.RULE_PROVIDER, this);
    }
}
```

Why a Map?

For the rules definition, Windup uses the OCPsoft Rewrite library. Whenever

there is something Windup specific which can't be pulled to the upstream implementation, Windup resorts to putting data into a metadata map.

This map is typically not used by rule authors, unless they use a custom Windup extension (don't confuse that with custom Ruleset).

Ruleset metadata

Ruleset metadata cover the whole Ruleset, not individual rules. For ruleset metadata, unlike rules, Windup uses a type-safe interface `WindupRulesetMetadata` (see [javadoc](#)).

Currently, it only defines Ruleset ID.

Rules Operations

A list of high-level operations across rulesets.

- `JavaClass`
- `Classification`
 - `Hint`
 - `Link`

TBD.

For the implementation of the operation, see [here](#).

TBD

Rules Ops Reporting Hint

DRAFT

TBD

Ops Reporting TypeReference

TBD

Different for Java and XML. Contains Line, Column

DRAFT: TBD?

Debugging and Troubleshooting

Debugging and Profiling

Debug the Windup Distribution Runtime

You can debug the Windup distribution using one of the following approaches.

1. Pass the `--debug` argument on the command line when you execute Windup.

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

2. Configure the `FORGE_OPTS` for debugging.

```
export FORGE_OPTS="-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000"
```

Debug a Test Using NetBeans

Click the `Debug Test File` button, or choose `Menu → Debug → Debug Test File`.

Profile a Test Using NetBeans

1. Profiling requires a lot of memory, so be sure to increase the NetBeans memory settings.

- Open the `NETBEANS_HOME/etc/netbeans.conf` file
- Find the line with `netbeans_default_options=`
- Add the following option

```
-J-Xmx5200m -J-XX:MaxPermSize=1024m
```

- Restart NetBeans.
2. Click `Menu > Profile > Profile Test File`.
 3. In the resulting dialog, enter the following.

```
exec.args=-Djboss.modules.system.pkgs=org.netbeans
```

Profile Forge Runtime Using YourKit

1. Download and unzip [YourKit](#). Be sure to get the trial license.
2. Configure the `FORGE_OPTS` for Yourkit:

```
export YOURKIT_HOME=/home/ondra/sw/prog/YourKit-b14092
export FORGE_OPTS="-Djboss.modules.system.pkgs=com.yourkit -agentpath:$YOURKIT_HOME/bin/linux-x86-64/libyjpagent.so=sampling,onexit=snapshot,delay=0"
```

3. Run `forge`. For details, see [Profiling Forge](#), but skip the first 2 points that direct you to copy the YourKit module and JAR into the Forge installation. Forge 2 already contains the YourKit module and JAR>.
4. Run `windup-analyze-app`.

```
forge -e windup-migrate-app
```

Troubleshoot Windup Issues

Logging

Logging is currently broken and will not be fixed any time soon.

See [Known Issues](#) and [WINDUP-73](#) for the current status.

Debugging Exceptions

Exceptions in Surefire reports are broken due to the way Forge wraps exceptions and the way Surefire handles them. You need to debug or rewrap exceptions using `TestUtil.rewrap(ex)`.

See [Known Issues](#) and [WINDUP-197](#) for the current status..

Classloading Problems

Configuring dependencies in a Forge-based project can be a little tricky. See [Dependencies](#) for some hints.

```
Caused by: java.lang.IllegalArgumentException: Type value for model  
'org.jboss.windup.rules.files.model.FileReferenceModel' is already registered with model  
org.jboss.windup.rules.files.model.FileReferenceModel
```

This means that the model class is loaded twice. I.e. the module containing it is loaded twice. Or, in tests, you may be (accidentally) adding the class to the deployment. This may especially happen after Maven coordinates of some module are changed.

Wiki Information

About the Windup Wiki

Purpose

This wiki is a place to create Windup documentation in a collaborative manner, a bit wildly, and on the fly. It is intended to reflect the state of the Windup source code master branch, however pages may become obsolete as the master changes quickly.

Areas of Interest

The Wiki pages are divided into the following main areas of interest, which may often overlap. The page names use these prefixes:

- **Dev- ...**: These pages are of interest to Windup core developers.
- **Rules- ...**: These pages are of interest to rules developers.
- **User- ...**: These pages that are of interest to Windup users.
- **no prefix**: These pages that are also of interest to Windup users.

Markdown, AsciiDoc, or...?

This Wiki uses AsciiDoc because it is capable of handling the publishing requirements and structural elements needed to create technical books. If you would like to contribute to the documentation, feel free to use your favorite Wiki language, however, it will ultimately be converted to AsciiDoc.

Contributor Guidelines

Please see the [Contributing Guidelines](#) for details about how to name pages and use the correct AsciiDoc syntax to make the transition to the final documentation go more smoothly.

The following is a brief overview.

- Name wiki pages using only letters, numbers and dashes. Do not include spaces, colons, or other special characters in page names.
- Be sure to add a level 3 page heading (===) at the top of each page. When compiled in a book, the Guide name is the top level, chapters are level 2, and the page is level 3. This is the title that appears in the table of contents for the page.

```
=== Build Windup from Source
```

- At the beginning of each new Wiki page, add an anchor with the exact page name, including the dashes. This generates the correct anchors that are needed when the pages are combined into a book.

```
[[Page-Name]]  
=== Page Name
```

- Wiki generates anchors using dashes for spaces. AsciiDoctor generates a leading underscore and underscores for spaces. When referring to another section within the same page, do not use the [\[section-title-anchor\]](#) syntax. Instead, create an anchor and use the 'xref:' syntax to provide the link. Replace spaces in the section title with dashes.
 - Create the anchor at the top of the section using double brackets to force the generation of dashes instead of underscores.

```
[[section-header]]  
==== Section Header
```

- Then use the `xref` to create the link.

```
See xref:section-header[Section Header] for more information.
```

Add Images to the Windup Wiki

- Fork <https://github.com/windup/windup/wiki>
- Clone the Windup Wiki into a `windup.wiki/` directory using the SSH clone URL:

```
git clone git@github.com:USER_NAME/windup.git windup.wiki
```

- Navigate to the new `windup.wiki/` directory.
- Add the upstream repository.

```
git remote add -f upstream git@github.com:windup/windup.wiki.git
```

- Check out a branch to work in.

```
git checkout -b BRANCH_NAME upstream/master
```

- Copy the image into the `windup.wiki/images` directory

```
cp IMAGE_NAME.png images/
```

- Add the new image to Git

```
git add images/IMAGE_NAME.png
```

- Commit the change

```
git commit -m "Add new image"
```

- Push the change to your Windup Wiki repository.

```
git push origin HEAD:master
```

- Test the new image in a page on your own Wiki. Don't forget to add the `:imagesdir: images` directive to the beginning of the page to point to the images directory.

```
image:IMAGE_NAME.png[New Image]
```

For example:

==== My Image



- If it works, push the image to the upstream repository

```
git push upstream HEAD:master
```

Windup Product Release and Documentation Process

Windup Release Process

Windup consists of artifacts from multiple GitHub repositories. Use the following procedure to create a Windup release.

1. Release <http://github.com/windup/nexus-indexer>

```
mvn release:prepare release:perform -DreleaseVersion=X -Dtag=Y -DdevelopmentVersion=Z
```

2. Release <http://github.com/windup/windup> *primary reactor*
3. Release <http://github.com/windup/windup-rulesets>
4. Release <http://github.com/windup/windup-distribution>

Create Windup JavaDoc

The most recent Windup JavaDoc build is located here:

<http://windup.github.io/windup/docs/javadoc/latest/index.html>

If you want to build an updated version, use the following procedure to create JavaDoc for Windup.

1. In Windup project root directory, type the following Maven command:

```
mvn validate -PjavadocDist
```

2. The JavaDoc is created in the Windup `target/site/apidocs` subdirectory.

NOTE

To cross-link with 3rd-party libraries, see <https://issues.jboss.org/browse/WINDUP-422>[WINDUP-422} and <http://stackoverflow.com/questions/27392535/javadoc-how-to-make-it-link-the-dependencies-maven-project>.

Windup Documentation Process

Wiki Documentation

- Name wiki pages using only letters, numbers and dashes. Do not include spaces, colons, or other special characters in page names.
- Use AsciiDoc rather than Markdown for the syntax.

Product Documentation

Windup product documentation is stored here:

<https://github.com/windup/windup-documentation>

See the [CONTRIBUTING guide](#) for AsciiDoc syntax rules regarding headers and links to make this process easier.

Update the Windup Documentation with the Latest Wiki Changes

Summary

The Windup guides are located in the windup-documentation/docs directory. They are named the same as the guides in the Wiki but have a 'Windup-' prefix. The windup-documentation/docs guides should kept synchronized with the versions in the windup.wiki.

Wiki Guide	Windup Docs Guide
User-Guide.asciidoc	Windup-User-Guide.adoc
Rules-Development-Guide.asciidoc	Windup-Rules-Development-Guide.adoc
Core-Development-Guide.asciidoc	Windup-Core-Development-Guide.adoc
Migration-Planning-Guide	Windup-Migration-Planning-Guide.adoc

Documentation Update Scripts

The <https://github.com/windup/windup-documentation/tree/master/scripts/windupDocUpdate.sh> script copies the pages from the Windup wiki to the windup-documentation repository, makes minor modifications, and generates each guide in a single HTML page book format. The resulting guides are created in the `WINDUP_DOCUMENTATION_HOME/html/` directory.

The script performs the following steps.

1. Copies the Windup wiki pages to the `WINDUP_DOCUMENTATION_HOME/docs/` directory, renaming them with the `.adoc` extension.
2. Makes sure each guide includes all pages that are referenced by pages within it. Scripts are provided to search for links within each individual guide. They navigate to the `WINDUP_DOCUMENTATION_HOME/docs/` directory and search using the following command:

```
grep 'xref:[A-Z]' `find . -name '*.adoc'`.
```

3. Replaces each `xref:` to external pages with an `xref:` using the following command.

```
find . -name '*.adoc' -print | xargs sed -i 's/xref:/xref:/g'
```

4. Navigates to the `WINDUP_DOCUMENTATION_HOME/` directory and builds the books. The single page HTML files are created in the `WINDUP_DOCUMENTATION_HOME/html/` directory.

Windup Guide	Command
Windup User Guide	<code>asciidoctor -t -dbook -a toc -o html/WindupUserGuide.html docs/Windup-User-Guide.adoc</code>
Windup Rules Development Guide	<code>asciidoctor -t -dbook -a toc -o html/WindupRulesDevelopmentGuide.html docs/Windup-Rules-Development-Guide.adoc</code>
Windup Core Development Guide	<code>asciidoctor -t -dbook -a toc -o html/WindupCoreDevelopmentGuide.html docs/Windup-Core-Development-Guide.adoc</code>

Publish the HTML Docs for Access at GitHub IO

The <https://github.com/windup/windup-documentation/tree/master/scripts/windupPublish.sh> script copies the pages from the Windup wiki to the windup-documentation repository, makes minor modifications, and generates each guide in a single HTML page book format. The resulting guides are created in the `WINDUP_DOCUMENTATION_HOME/html/` directory.

The script performs the following steps.

- The first time, you must fork and clone the windup GitHub repository. After that, just fetch the latest upstream.

```
git clone https://github.com/windup/windup.git
git fetch upstream
```

- Navigate to the local `WINDUP_HOME/` directory.

```
cd windup
```

- Checkout the `gh-pages` from the `WINDUP_HOME` repository

```
git checkout -b gh-pages windup/gh-pages
```

- If the `WINDUP_HOME/docs/WINDUP_RELEASE/` directory does not yet exist, create a `docs/WINDUP_RELEASE/html/images/`

```
mkdir -p docs/WINDUP_RELEASE/html/images (if it doesn't exist!)
```

- Copy any the images, stylesheets, and html files from windup-

documentation:

```
cp WINDUP_DOCUMENTATION_HOME/html/*.html WINDUP_HOME/docs/WINDUP_RELEASE/html/  
cp WINDUP_DOCUMENTATION_HOME/*.css WINDUP_HOME/docs/WINDUP_RELEASE/html/  
cp WINDUP_DOCUMENTATION_HOME/images/*  
WINDUP_HOME/docs/WINDUP_RELEASE/html/images/
```

- Add the updated files to GitHub.

```
git add docs
```

- Commit the changes.

```
git commit -m "Update message..."
```

- Push the changes to GitHub.

```
git push upstream gh-pages
```

- View the documentation at http://windup.github.io/windup/docs/WINDUP_RELEASE/html/WindupUserGuide.html
- Update the symlink for `latest` to point to the new version.
 - Navigate to the `WINDUP_HOME/docs` directory.
 - Remove the existing symlink for "latest".

```
unlink latest
```

- Create a symlink to the new documentation.

```
Syntax: ln -s WINDUP_RELEASE latest  
Example: ln -s 2.0.0.Final latest
```

- Add the modified "latest" directory to Git.

```
git add latest
```

- Commit the change.

```
git commit -m 'Replace symlink for latest to point to 2.0.0.Final'
```

- Push the changes to your own git repository, verify and issue a pull.

```
git push origin HEAD
```

- Push the changes upstream

```
git push upstream gh-pages
```

- View the documentation at <http://windup.github.io/windup/docs/latest/html/WindupUserGuide.html>

Additional Resources

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.

Download the Latest 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.

Fork and Clone the 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 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. To get the latest files from the `upstream` repository.

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

Get Involved

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. Some items require only a few minutes of your time!

- [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 Windup rules.
 - Write a Windup rule add-on to automate a migration process.
 - Create a test for the new rule.
 - For details, see the *Windup Rules Development Guide*.
- You can also contribute to the project source code.
 - Create a core rule.
 - Improve 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!

Helpful links

- Windup Wiki: <https://github.com/windup/windup/wiki>
- Windup documentation (generated from the Wiki documentation at the link above):
 - <http://windup.github.io/windup/docs/latest/html/WindupUserGuide.html>
 - <http://windup.github.io/windup/docs/latest/html/WindupRulesDevelopmentGuide.html>
 - <http://windup.github.io/windup/docs/latest/html/WindupCoreDevelopmentGuide.html>
- Windup Forums: <https://community.jboss.org/en/windup>
- Windup Forums: <https://developer.jboss.org/en/windup> (inherited from Windup 0.x)
- 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](#)

You can also follow us on this IRC channel: `irc.freenode.net#windup`.

Known Windup Issues

Windup known issues are tracked here: [Open Windup issues](#)

Report Issues with Windup

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

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 `Confirmaddress` button.
4. Follow the instructions sent to your email address.

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.

Appendix

Glossary of Terms Used in Windup

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

A class that implements one or more rules using the `.addRule()` method. The following are examples of legacy Java RulesProviders that are defined in `rules-java-ee` ruleset.

- EjbConfig
- JDKConfig
- SeamToCDI

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.

Reporting Terms

Lift and Shift (Level of effort)

The code or file is standards-based and can be ported to the new environment with no changes.

Mapped (Level of effort)

There is a standard mapping algorithm to port the code or file to the new environment.

Custom (Level of effort)

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.

Windup Project Information

Github Repository

The Windup project github repository is located at <https://github.com/windup/windup/>.

See the *Core-Developer-Guide* for details on how to contribute to the Windup project source code.

Documentation

The Windup documentation is currently located here in the Windup project [Wiki](#).

For additional information, refer to the Windup [Javadoc](#).

Website

There is currently no website for Windup 2.0.

The windup.jboss.org website currently provides information primarily for legacy Windup 1.x (legacy).

IRC chat

Server: [irc.freenode.net](irc://irc.freenode.net)

Channel: [#windup](#)

Mailing lists

Subscribe to the JBoss mailing lists at <https://lists.jboss.org/mailman/listinfo/windup-dev>.

- Core development discussion: windup-dev@redhat.com
- Rules development discussion, usage: windup-users@redhat.com

Core development team (and IRC nicks)

Lead: Lincoln Baxter (lincolnthree)

Members: Jess Sightler (jsightler), Matej Briskar (mbriskar), Ondrej Zizka (ozizka)

IRC meeting bot commands (hint for the moderator)

```
#startmeeting
#chair lincolnthree, ozizka, jsightler, mbriskar
#addtopic Status Reports
#addtopic Next steps
#nexttopic
#info ...
#endmeeting
Useful Commands: #action #agreed #help #info #idea #link #topic.
```