

Windup Core Development Guide

Introduction

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

What is Windup?



Overview

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

Running from a [Forge](#) environment, Windup examines application artifacts, including project source directories and applications archives, then produces an HTML report highlighting areas that need changes. Windup can be used to migrate Java applications from previous versions of *Red Hat JBoss Enterprise Application Platform* or from other containers, such as *Oracle WebLogic Server* or *IBM® WebSphere® Application Server*.

How Does Windup Simplify Migration?

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

Features of Windup

Shared Data Model

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

- It enables complex rule interaction, allowing rules to pass findings to other rules.
- It enables 3rd-party plug-ins to interact with other plug-ins, rules and reports.
- The findings in data graph model can be searched and queried during rule execution and used for reporting purposes.

Extensibility

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

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

Better Rules

Windup provides more powerful and complex rules.

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

Automation

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

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

Work Estimation

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

Level of effort is represented as *story points* in the Windup reports.

Better Reporting

Windup reports are now targeted for specific audiences.

- IT Management - Applications are ranked by cost of migration.
- Project Management - Reports detail the type of work and estimation of effort to complete the tasks.
- Developers - An Eclipse plug-in provides hints and suggested code changes within the IDE.

About Windup Rules

Windup is a rule-based migration tool that analyzes the APIs, technologies, and architectures used by the applications you plan to migrate. In fact, the Windup tool executes its own core set of rules through all phases of the migration process. It uses rules to extract files from archives, decompile files, scan and classify file types, analyze XML and other file content, analyze the application code, and build the reports.

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

Windup rules use the following familiar rule pattern:

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

Windup provides comprehensive set of standard migration rules out-of-the-box. Because applications may contain custom libraries or components, Windup allows you to write your own rules to identify use of components or software that may not be covered by the existing ruleset. If you plan to write your own custom rules, see the *Windup Rule Development Guide* for detailed instructions.

System Requirements

Software

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

Hardware

The following memory and disk space requirements are the minimum needed to run Windup. If your application is very large or you need to evaluate multiple applications, you may want to increase these values to improve performance. For tips on how to optimize performance, see [Optimize Windup Performance](#).

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

About the WINDUP_HOME Variable

This documentation uses the **WINDUP_HOME** *replaceable* value to denote the path to the Windup distribution. When

you encounter this value in the documentation, be sure to replace it with the actual path to your Windup installation.

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

Get Started

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

Install Windup

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

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

2. Download the latest [Windup ZIP distribution](#).
3. Extract the ZIP file in to a directory of your choice.

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 you do not specify a folder, Windup creates one for you at the same level as the application.
 - If you specify a folder and it does not exist, it is created by Windup.
 - If you specify a folder and it does exist, you see the following:
 - In batch mode, you must specify `--overwrite` or you see the following error.

```
***ERROR*** Files exist in /home/username/OUTPUT_REPORT_DIRECTORY, but --overwrite not specified. Aborting!
```

- In interactive mode, you see the following prompt and can choose whether to overwrite the directory.

```
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.

WARNING | Be careful not to specify a report output directory that contains important information!

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

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

Run Windup in Batch Mode

This is the quickest way to run Windup.

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

```
For Linux:    $ WINDUP_HOME/bin/windup --input INPUT_ARCHIVE --output OUTPUT_REPORT --packages PACKAGE_1 PACKAGE_2 PAC
For Windows:  > WINDUP_HOME\bin\windup.bat --input INPUT_ARCHIVE --output OUTPUT_REPORT --packages PACKAGE_1 PACKAGE_2
```

3. This command can take arbitrary options processed by different add-ons. The list of options in the core Windup distribution can be found in the [Javadoc](#). Most commonly used command line arguments are:

--input INPUT_ARCHIVE_OR_FOLDER

This is the fully qualified application archive or source path.

--output OUTPUT_REPORT_DIRECTORY (optional)

The fully qualified path to the folder that will contain the the report information produced by Windup. If omitted, the report will be generated in a **INPUT_ARCHIVE_OR_FOLDER.report** folder. If the output directory exists, you will see the error described above and must specify the **--overwrite** argument. This forces Windup to delete and recreate the folder, so be careful not to specify an output directory that contains important information!

--source (optional)

One or more source technologies, platforms, or frameworks to migrate from.

--target (optional)

One or more source technologies, platforms, or frameworks to migrate to.

--overwrite (optional)

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

--userRulesDirectory (optional)

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

--packages PACKAGE_1, PACKAGE_2, PACKAGE_N (optional)

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

--excludePackages PACKAGE_1, PACKAGE_2, PACKAGE_N (optional)

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

--source-mode true (optional)

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

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

```
WINDUP_HOME/bin/windup --input INPUT_ARCHIVE_OR_FOLDER --output OUTPUT_REPORT_DIRECTORY --packages PACKAGE_1 PACKAGE_2
```

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

```
WINDUP_HOME/bin/windup --sourceMode true --source SOURCE_TECHNOLOGY --target TARGET_TECHNOLOGY --input INPUT_ARCHIVE_O
```

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

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

```
**SUCCESS** Windup report created: PATH_TO_REPORTS/index.html
Access it at this URL: file:///home/username/PATH_TO_REPORTS/index.html
```

WARNING

Depending on the size of the application and the hardware Windup is running on, this command can take a very long time. For tips on how to improve performance, see [Optimize Windup Performance](#).

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](#).

Windup Help for Batch Mode

To see the complete list of available arguments for the `windup` command, execute the following command in the terminal:

```
WINDUP_HOME/bin/windup --help
```

Start Windup in Interactive Mode

Windup can also be run in interactive mode. This mode offers tab completion, which is useful if you need assistance with valid argument values, for example, valid `--source` or `--target` technologies.

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
```

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

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

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

4. The command to run Windup is `windup-migrate-app`. It uses the same arguments as batch mode. If you are unsure of valid argument values, hit the `tab` a few times to see what is available.
5. To evaluate an application archive, use the following syntax:

```
windup-migrate-app --input INPUT_ARCHIVE_OR_FOLDER --output OUTPUT_REPORT_DIRECTORY -source SOURCE_TECHNOLOGY --target
```



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 -source SOURCE_TE
```



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

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

WARNING

Depending on the size of the application and the hardware Windup is running on, this command can take a very long time. For tips on how to improve performance, see [Optimize Windup Performance](#).

7. To exit Windup, type:

```
exit
```

8. Review the report as described for batch mode.

Windup Help for Interactive Mode

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

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

Windup Command Examples

The following batch mode examples report against applications located in the Windup source `test-files` directory. The same arguments can be used to run the commands interactively in Windup using the `windup-migrate-app` command.

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_HOME/bin/windup --sourceMode true --input /home/username/windup-source/test-files/seam-booking-5.2/ --output /home/username/windup-reports/seam-booking-report --source eap4,eap5 --target eap6 --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_HOME/bin/windup --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 --source eap4,eap5 --target eap6 --packages com.acme org.apache
```

Windup Quickstart Examples

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

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

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_HOME/bin/windup --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:

Windup Report: Overview



Overview / Profiled by Windup

Name	Technology	Effort
JEE Example App (org.windup.example:jee-example-app-1.0.0)	Web Services JAX-WS 2.2 JAX-RS 2.1 Stereotypes JAX-WS 2.1 Stereotypes	34 Story Points

All Rules | Windup FreeMarker Methods

This page lists the applications that were processed along with the technologies that were encountered.

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

- The file name
- The file type
- A list of issues, if any, that were found in the file
- The estimated total *Story Points* to migrate the file. *Story Points* are covered in more detail in the *Windup Rules Development Guide*.

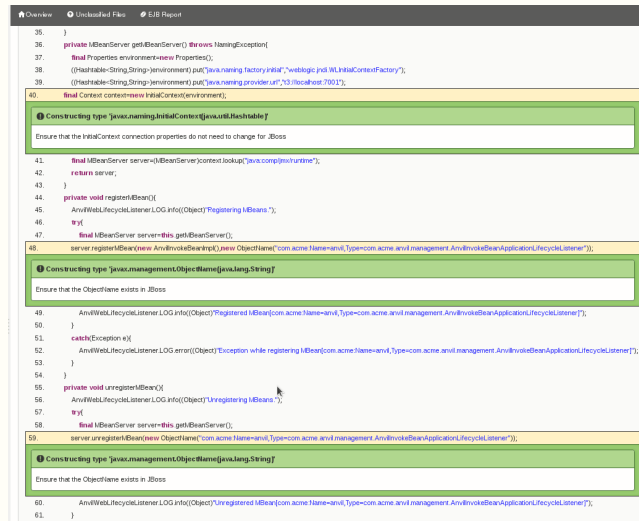
NOTE

The estimated Story Points change as new rules are added to Windup. The values here may not match what you see when you test this application.

In the following Windup Application Report example, the migration of the **JEE Example App** EAR is assigned a total of 34 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.

Windup Report: Application Report

Windup Report: Source Report - Part 2



Additional Reports

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

Rule Provider Execution Report

The `WINDUP_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.

Windup Report: Rule Provider Report

All Applications				
Rule Provider Executions				
Phase: DependentPhase				
Phase: InitializationPhase				
IgnoredArchivesConfigLoadingRuleProvider				
Phase: InitializationPhase				
Rule	Statistics	Executed?	Failed?	Failure Cause
addRule() perform(org.jboss.windup.rules.apps.java.archives.config.IgnoredArchivesConfigLoadingRuleProvider\$1@38a13868) with("generatedId_IgnoredArchivesConfigLoadingRuleProvider_1")	Vertices Created: 0 Edges Created: 0 Vertices Removed: 0 Edges Removed: 0	yes	no	
ArchiveIdentifierConfigLoadingRuleProvider				
Phase: InitializationPhase				
Rule	Statistics	Executed?	Failed?	Failure Cause
addRule() perform(org.jboss.windup.rules.apps.java.archives.config.ArchiveIdentifierConfigLoadingRuleProvider\$1@72d3d4) with("generatedId_ArchiveIdentifierConfigLoadingRuleProvider_1")	Vertices Created: 0 Edges Created: 0 Vertices Removed: 0 Edges Removed: 0	yes	no	
CopyJavaConfigToGraphRuleProvider				
Phase: InitializationPhase				
Rule	Statistics	Executed?	Failed?	Failure Cause
addRule() perform(org.jboss.windup.rules.apps.java.config.CopyJavaConfigToGraphRuleProvider\$1@3a9b7a95) with("generatedId_CopyJavaConfigToGraphRuleProvider_1")	Vertices Created: 11 Edges Created: 10 Vertices Removed: 0 Edges Removed: 0	yes	no	
GatherIgnoredFileNamesRuleProvider				
Phase: InitializationPhase				
Rule	Statistics	Executed?	Failed?	Failure Cause
addRule() when(Query.find(org.jboss.windup.graph.model.WindupConfigurationModel.asDefault)) perform(function over(?) perform(Gather all the information about ignored files)) with("generatedId_GatherIgnoredFileNamesRuleProvider_1")	Vertices Created: 0 Edges Created: 0 Vertices Removed: 0 Edges Removed: 0	yes	no	
Phase: DiscoveryPhase				
DiscoverFilesAndTypesRuleProvider				
Phase: DiscoveryPhase				

Rule Provider Execution Report

The `WINDUP_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 `WINDUP_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.

The screenshot shows a web browser displaying a directory index of report files. The interface includes a standard browser menu at the top, a search bar, and navigation links like 'Up to higher level directory'. The table below provides details for each file entry.

Name	Size	Last Modified
[[AnvilWebClientListener.java.html	13 KB	05/20/2015 07:38:02 AM
[[AnvilWebServer.java.html	7 KB	05/20/2015 07:38:01 AM
[[AnvilWebServletListener.java.html	12 KB	05/20/2015 07:38:02 AM
[[AuthenticationFilter.java.html	7 KB	05/20/2015 07:38:01 AM
[[ItemCatalogItem.java.html	5 KB	05/20/2015 07:38:01 AM
[[ItemLookupItem.java.html	5 KB	05/20/2015 07:38:01 AM
[[ItemLookupLocalHome.java.html	5 KB	05/20/2015 07:38:01 AM
[[ItemLookupLocal.java.html	5 KB	05/20/2015 07:38:01 AM
[[ItemLookup.java.html	5 KB	05/20/2015 07:38:01 AM
[[JEE_Example_App...org_windup_example_jee_example_app_1.0.0_test	26 KB	05/20/2015 07:38:01 AM
[[LogEventPublisher.java.html	8 KB	05/20/2015 07:38:02 AM
[[LogEventListener.java.html	8 KB	05/20/2015 07:38:01 AM
[[LogEventTopic_jms_vml.html	5 KB	05/20/2015 07:38:01 AM
[[LogFilter.java.html	6 KB	05/20/2015 07:38:02 AM
[[MANIFEST_MF.1.html	5 KB	05/20/2015 07:38:01 AM
[[MANIFEST_MF.2.html	5 KB	05/20/2015 07:38:01 AM
[[MANIFEST_MF.3.html	5 KB	05/20/2015 07:38:01 AM
[[MANIFEST_MF.4.html	5 KB	05/20/2015 07:38:02 AM
[[MANIFEST_MF.html	5 KB	05/20/2015 07:38:01 AM
[[ProductCatalogItem.java.html	6 KB	05/20/2015 07:38:01 AM
[[ProductCatalogItemHome.java.html	5 KB	05/20/2015 07:38:02 AM
[[ProductCatalogLocalHome.java.html	5 KB	05/20/2015 07:38:01 AM
[[ProductCatalogLocal.java.html	5 KB	05/20/2015 07:38:01 AM
[[ProductCatalog.java.html	5 KB	05/20/2015 07:38:01 AM
[[application_vml.html	5 KB	05/20/2015 07:38:01 AM
[[sph_iv_vml.html	8 KB	05/20/2015 07:38:02 AM
[[spring_JEE_Example_App...org_windup_example_jee_example_app_1.0.0_test	4 KB	05/20/2015 07:38:01 AM
[[faces.config_vml.html	5 KB	05/20/2015 07:38:01 AM
[[netclassformatter_JEE_Example_App...org_windup_example_jee_example_app_1.0...	8 KB	05/20/2015 07:38:01 AM
[[pom_properties.1.html	5 KB	05/20/2015 07:38:02 AM
[[pom_properties.2.html	5 KB	05/20/2015 07:38:01 AM
[[pom_properties.3.html	5 KB	05/20/2015 07:38:02 AM
[[pom_properties.4.html	5 KB	05/20/2015 07:38:01 AM
[[pom_properties.html	5 KB	05/20/2015 07:38:01 AM
[[resources		05/20/2015 07:38:02 AM
[[twofoldcontent		05/20/2015 07:17:59 AM
[[web_vml.html	6 KB	05/20/2015 07:38:01 AM
[[weblogic_application_vml.html	5 KB	05/20/2015 07:38:01 AM
[[weblogic_sph_iv_vml.html	7 KB	05/20/2015 07:38:01 AM
[[weblogic_vml.html	5 KB	05/20/2015 07:38:01 AM
[[windup_freemarkerFunctions.html	8 KB	05/20/2015 07:38:02 AM
[[windup_categoryProvider.html	1007 KB	05/20/2015 07:38:02 AM

If you plan to contribute to the core code base or create Java-based rule add-ons, you must first configure Maven to build Windup from source. The procedure you follow depends on whether you plan to build Windup using an IDE or Maven Command line.

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 Using Your IDE](#). 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.

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. Be 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 Using Command Line

Follow this procedure to configure the Maven settings if you plan to build Windup using the command line.

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.

Get the Windup Source Code

The source code for the Windup project is located in 2 repositories:

- Windup core code base: <https://github.com/windup/windup>
- Windup XML-based rulesets: <https://github.com/windup/windup-rulesets>
- Windup distribution: <https://github.com/windup/windup-distribution>

To contribute to the Windup project source code, you must fork the Windup, Windup ruleset, and Windup distribution repositories to your own Git, clone your forks, commit your work on topic branches, and make pull requests back to the corresponding 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
```

Fork and Clone the Windup Rulesets Repository

The Windup XML rulesets now live in a separate repository. Follow these instructions to fork and clone them.

1. [Fork](#) the Windup Rulesets project. This creates the `windup-rulesets` 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-rulesets.git
```

3. Change to the `windup-rulesets` 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-rulesets.git
```

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

```
git fetch upstream
```

Fork and Clone the Windup Distribution Repository

1. [Fork](#) the Windup distribution project. This creates the `windup-distribution` 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-distribution.git
```

3. Change to the `windup-distribution` 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-distribution.git
```

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

```
git fetch upstream
```

Build Windup from Source

Overview

This information is provided for new developers who plan to contribute code to the Windup open source project. It describes how to build Windup from source using an IDE or command line and 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 Developers](#).

System Requirements

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

The IDE must embed Maven 3.1.1 or later. See [Install and Configure Maven](#) for details.

Prerequisites

- Be sure you have configured Maven as described here: [Install and Configure Maven](#).
- Windup source code is located in 3 GitHub projects. Follow the instructions to [Get the Windup Source Code](#).

Build Using Red Hat JBoss Developer Studio or Eclipse

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

1. Configure the Maven installation in your IDE as described here: [Install and Configure Maven](#).
2. Start JBoss Developer Studio or Eclipse.
3. Import the `windup-ruleset` project.
 - From the menu, select `File` → `Import`.
 - In the selection list, choose `Maven` → `Existing Maven Projects`, then click `Next`.
 - Click `Browse` and navigate to the root of the `windup-ruleset` project directory, then click `OK`.
 - 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.
 - In the Project Explorer tab, right-click on the `windup-ruleset` project and choose `Run As` → `Maven install`.
4. Import the `windup` project.
 - From the menu, select `File` → `Import`.
 - In the selection list, choose `Maven` → `Existing Maven Projects`, then click `Next`.
 - Click `Browse` and navigate to the root of the `windup` project directory, then click `OK`.

- 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.
- In the Project Explorer tab, find the `windup_parent` project in the list, right-click, and choose `Run As -> Maven install` .

5. Import the windup-distribution project.

- From the menu, select `File -> Import` .
- In the selection list, choose `Maven -> Existing Maven Projects` , then click `Next`.
- Click `Browse` and navigate to the root of the `windup-distribution` project directory, then click `OK` .
- 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.
- In the Project Explorer tab, right-click on the `windup-distribution` project and choose `Run As -> Maven install` .

Build Using Maven Command Line

Windup source code consists of 3 projects:

- [windup-ruleset](#)
- [windup](#)
- [windup-distribution](#)

The `windup-distribution` project has dependencies on the other 2 projects, so you must build the `windup-ruleset` and `windup` projects first. Use the following steps to build the Windup distribution.

1. Build the [windup-ruleset](#) project.

- Open a command terminal and navigate to the root of the Windup Ruleset project directory.

```
cd windup-ruleset/
```

- Build the project.

```
mvn clean install
```

2. Build the [windup](#) project.

- Open a command terminal and navigate to the root of the windup project directory.

```
cd windup/
```

- Build the project.

```
mvn clean install
```

- You can also build the project without the tests.

```
mvn clean install -DskipTests
```

3. Build the [windup-distribution](#) project.

- Open a command terminal and navigate to the root of the Windup distribution project directory.

```
cd windup-distribution/
```

- Build the project.

```
mvn clean install
```

- This creates a `windup-distribution-<VERSION>-offline.zip` file in the `windup-distribution/target/`

directory.

Build Using Red Hat JBoss Developer Studio or Eclipse

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

1. Configure the Maven installation in your IDE as described here: [Install and Configure Maven](#).
2. Start JBoss Developer Studio or Eclipse.
3. Import the `windup-ruleset` project.
 - From the menu, select `File` → `Import`.
 - In the selection list, choose `Maven` → `Existing Maven Projects`, then click `Next`.
 - Click `Browse` and navigate to the root of the `windup-ruleset` project directory, then click `OK`.
 - 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.
 - In the Project Explorer tab, right-click on the `windup-ruleset` project and choose `Run As` → `Maven install`.
4. Import the `windup` project.
 - From the menu, select `File` → `Import`.
 - In the selection list, choose `Maven` → `Existing Maven Projects`, then click `Next`.
 - Click `Browse` and navigate to the root of the `windup` project directory, then click `OK`.
 - 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.
 - In the Project Explorer tab, find the `windup_parent` project in the list, right-click, and choose `Run As` → `Maven install`.
5. Import the `windup-distribution` project.
 - From the menu, select `File` → `Import`.
 - In the selection list, choose `Maven` → `Existing Maven Projects`, then click `Next`.
 - Click `Browse` and navigate to the root of the `windup-distribution` project directory, then click `OK`.
 - 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.
 - In the Project Explorer tab, right-click on the `windup-distribution` project and choose `Run As` → `Maven install`.

Extract the Distribution Source File

The build process creates a `windup-distribution-<VERSION>-offline.zip` file in the `windup-distribution/target/` directory.

Unzip the file into a directory of your choice.

Quick Build Review for Experienced Developers

```
git clone git@github.com:windup/windup-ruleset.git windup
cd windup-ruleset
mvn clean install
git clone git@github.com:windup/windup.git windup
cd windup
mvn clean install -DskipTests
git clone git@github.com:windup/windup-distribution.git windup
cd windup-distribution
mvn clean install
unzip target/windup-distribution-<VERSION>-offline.zip -d <WINDUP-DIRECTORY>
```

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/

## Execute using Batch Mode
bin/windup.sh --input /home/username/work/Migration/TestApps/_apps/BradApp --output Report --packages org com net

## Execute using Interactive Mode
bin/windup.sh
$ windup-migrate-app --input /home/username/work/Migration/TestApps/_apps/BradApp --output Report --packages org com net

## Exit
$ exit

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

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 `ide-config` 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 `ide-config` directory of the Windup project, 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 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 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 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](#).

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 Core Project Structure

Windup source code consists of 3 projects:

- [windup-ruleset](#)
- [windup core](#)
- [windup-distribution](#)

The Windup core project contains the executable source code and 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.

Run Windup Against Your Application

When you run the `WINDUP_HOME/bin/windup` command against your application, Windup executes its own core rules to process the following types of application input artifacts:

- Archive files such as EARS, WARs, JARs

- Java classes
- JSP files
- Manifest files
- XML files

Windup extracts files from archives, decompiles classes, and analyzes the application code. In this phase, it 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, see [Rule Phases](#).

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 is architected to allow you to create Java-based rule addons or XML rules and add 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 phases provide a way for rule authors to control the rule lifecycle by specifying the phase in which the rule should execute. Windup executes rules sequentially within rule phases, however, you can also provide more fine-grained control over the order of rule execution within a phase. A rule may specify that one or more rules must be executed before it this rule is run. All named rules will be fired in the order specified before executing the the current rule. 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.

The rule phase and execution order is stored in the associated rule's [RuleMetadata](#).

For a detailed description of Windup rule phases and the order of their execution, see: [Rule Phases](#)

Set the Rule Execution Phase

You can set the phase in which the rule executes in one of the following ways.

- Add the `@RuleMetadata(phase = RulePhase)` annotation to the rule.
- Code the `setPhase(RulePhase)` method in the constructor of the rule.

Control the Execution Order Within the Rule Phase

You can also provide more fine-grained control over the order of rule execution within a phase. A rule may specify that one or more rules must be executed before it this rule is run. All named rules will be fired in the order specified before executing the the current rule. 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.

This is done in one of the following ways.

- Add the `@RuleMetadata(after = PreviousRuleProvider, before = NextRuleProvider)` annotation to the rule.
- Use the `addExecuteAfter(NextRuleProvider)` or `addExecuteBefore(PreviousRuleProvider)` methods in the constructor, specifying the rules that should precede or follow this rule.

Code Example Using the Annotation Method

The following is an example of a rule that overrides the rule phase and sets the ordering using the `@RuleMetadata` annotation.

```
@RuleMetadata(id = "MyCustomRuleProvider",
    phase = DependentPhase.class,
    after = { MyFirstRuleProvider.class, MySecondRuleProvider.class },
    before = { MyFinalRuleProvider1.class })
public class MyCustomRuleProvider extends AbstractRuleProvider
{
}
```

Code Example Using the Constructor Method

The following example provides the same results using constructor methods.

```
public MyCustomRuleProvider extends AbstractRuleProvider
{
    super(MetadataBuilder.forProvider(MyCustomRuleProvider.class)
        .setPhase(DependencyPhase.class)
        .addExecuteAfter({ MyFirstRuleProvider.class, MySecondRuleProvider.class }),
        .addExecuteBefore({ MyLastRuleProvider.class}));
}
```

Additional Information

For more information about what can be specified in the `@RuleMetadata` annotation, see the [RuleMetadata](#) JavaDoc.

For more information about RuleProvider constructor MetadataBuilder methods, see the [MetadataBuilder](#) JavaDoc.

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

Rule Phases

Rule phases provide a way for rule authors to control the rule lifecycle by specifying the phase in which the rule should execute. Windup executes rules sequentially within rule phases, however, the order of rule execution within a phase can be controlled by specifying other rules that should be run before or after the rule.

By default, rules run in the [MigrationRulesPhase](#). 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, so a rule can specify that it is run during another phase in the process.

The rule phases below are listed in the order in which they are executed by Windup. The exception is the last phase, which can occur during any phase of the execution lifecycle.

[InitializationPhase](#)

This is the first phase of Windup Execution. Initialization related tasks, such as copying configuration data to the graph, should occur during this phase.

[DiscoveryPhase](#)

This resource discovery phase immediately follows the [InitializationPhase](#). During this phase, input files are identified by the name, extension, location, and fully qualified Java class names. Typically, any rule that only puts data into the graph is executed during this phase.

[**ArchiveExtractionPhase**](#)

This phase immediately follows the [DiscoveryPhase](#). During this phase, input files such as EARs, WARs, JARs, and other zipped files are unzipped during this phase.

[**ArchiveMetadataExtractionPhase**](#)

This phase occurs immediately after [ArchiveExtractionPhase](#). It calculates checksums for archives and determines whether the archive is an EAR, WAR, JAR, or some other type of compressed file.

[**ClassifyFileTypesPhase**](#)

This phase follows the [ArchiveMetadataExtractionPhase](#). During this phase, files are scanned and metadata is created for them. For example, this phase may find all of the Java files in an application and mark them as Java, or it may find all of the bash scripts in an input and identify them appropriately.

[**DiscoverProjectStructurePhase**](#)

This phase, which follows [ClassifyFileTypesPhase](#), identifies the project structure of the input application. This includes identification of project files, any subprojects, and the type of project, for example Maven or Ant.

[**DecompilationPhase**](#)

This phase follows the [DiscoverProjectStructurePhase](#). This phase is responsible for identifying and decompiling classes included in the input application.

[**InitialAnalysisPhase**](#)

This phase follows the [DecompilationPhase](#) and is called to perform a basic analysis of file content. It extracts all method names from class files and extracts metadata, such as the XML namespace and root element, from XML files.

[**MigrationRulesPhase**](#)

This phase, which follows the [InitialAnalysisPhase](#), is the default phase for all rules unless it is specifically overridden. During this phase, migration rules attach data to the graph associated with migration. This can include hints to migrators for manual migration, automated migration of schemas or source segments, blacklists to indicate vendor specific APIs.

[**PostMigrationRulesPhase**](#)

This phase occurs immediately after [MigrationRulesPhase](#). This phase can be used to execute a rule that must follow all other migration rules. The primary use case at the moment involves unit tests.

[**PreReportGenerationPhase**](#)

This phase occurs after the [PostMigrationRulesPhase](#) and immediately before the [ReportGenerationPhase](#). It can be used for initialization tasks that will be needed by all reports during that phase.

[**ReportGenerationPhase**](#)

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

[**PostReportGenerationPhase**](#)

This phase occurs immediately after the main tasks of report generation. It can be used to generate reports that need data from all of the previously generated reports.

[**ReportRenderingPhase**](#)

This is the phase that renders the report.

[**PostReportRenderingPhase**](#)

This phase occurs immediately after reports have been rendered. It can be used to render any reports that need to execute last. One possible use is to render all the entire contents of the graph itself.

FinalizePhase

This phase is called to clean up resources and close streams. This phase occurs at the end of execution. Rules in this phase are responsible for any cleanup of resources and closing any streams that may have been opened.

PostFinalizePhase

This occurs immediately after the FinalizePhase. This is an ideal place to put Rules that would like to be the absolute last things to fire, for example reporting on the execution time of previous rules or reporting on all of the rules that have executed and which AbstractRuleProviders executed them.

DependentPhase

This phase can occur during any phase of the execution lifecycle. It's exact placement is determine by the code within the rule.

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);

```

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 the *level of effort* for the *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
Trivial	1	The migration is a trivial change or a simple library swap with no or minimal API changes.
Complex	3	The changes required for the migration task are complex, but have a documented solution.
Redesign	5	The migration task requires a redesign or a complete library change, with significant API changes.
Rearchitecture	7	The migration requires a complete rearchitecture of the component or subsystem.
Unknown	13	The migration solution is not known and may need a complete rewrite.

Task Severity

In addition to the *level of effort*, migration tasks can be assigned a *severity* that indicates whether the task must be completed or can be postponed.

Mandatory

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

Optional

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

Difference Between XML-based and Java-based Rules

Summary

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

Which one to choose?

XML-based rules provide a quick, simple way to create rules to analyze Java, XML, and properties files. If you simply need to highlight a specific section of Java code or XML file content and provide migration hints for it, creation of *XML-based* rules is the recommended approach. Creation of custom *XML-based* rules is covered in the *Windup Rules Development Guide*.

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

Pros and Cons of XML-based Rules

Pros:

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

Cons:

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

Pros and Cons of Java-based Rules

Pros:

- Java rule add-ons allow you to write custom conditions and operations and provide a lot of flexibility.
- Java rule add-ons allow you to access and manipulate the shared data model graph and to customize reports.
- You can set breakpoints and test Java rule add-ons using a debugger.
- IDEs provide code completion for the Windup API.

Cons:

- You must configure Maven to compile Java rule add-ons.

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

Examples of XML-based and Java Based Rules

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

```
<?xml version="1.0"?>
<ruleset id="EjbRules"
  xmlns="http://windup.jboss.org/schema/jboss-ruleset"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset http://windup.jboss.org/schema/jboss-ruleset/windup-jb
<rules>
  <rule id="EjbRules_1000">
    <when>
      <javaclass references="javax.persistence.Entity" as="default">
        <location>TYPE</location>
      </javaclass>
    </when>
    <perform>
      <iteration>
        <classification classification="JPA Entity" effort="0"/>
      </iteration>
    </perform>
  </rule>
</rules>
</ruleset>
```

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

```
/**
 * Scans for classes with EJB related annotations, and adds EJB related metadata for these.
 */
public class DiscoverEjbAnnotationsRuleProvider extends AbstractRuleProvider
{
    @Override
    public Configuration getConfiguration(GraphContext context) {
        return ConfigurationBuilder.begin()
            .addRule()
            .when(JavaClass.references("javax.ejb.{annotationType}").at(TypeReferenceLocation.ANNOTATION))
            .perform(new AbstractIterationOperation<JavaTypeReferenceModel>() {
                {
                    public void perform(GraphRewrite event, EvaluationContext context, JavaTypeReferenceModel payload) {
                        {
                            extractEJBMetadata(event, payload);
                        }
                    }
                }
            })
            .where("annotationType").matches("Stateless|Stateful")
            .withId(ruleIDPrefix + "_StatelessAndStatefulRule")
            .addRule()
            .when(JavaClass.references("javax.ejb.MessageDriven").at(TypeReferenceLocation.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(ENTITY_ANNOTATIONS)
                .or(JavaClass.references("javax.persistence.Table").at(TypeReferenceLocation.ANNOTATION).as(TABLE_ANNOTATIONS))
            .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");
    }
    ...
}
```

Quick Comparison Summary

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

Create and Test Java Rule Add-ons

Java-based Rule Structure

RuleProvider

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 can implement the [RuleProvider](#) interface or they can extend the [AbstractRuleProvider](#) class, the [SingleRuleProvider](#) class, or the [IteratingRuleProvider](#) 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 in the [getConfiguration\(GraphContext context\)](#) method. This method is inherited from the [OCPSoft Rewrite](#) interface `org.ocpsoft.rewrite.config.ConfigurationProvider`. Rules are discussed in more detail below under [Add Rule Code Structure](#)
- If other rules must execute after or before the rules in this provider, you must provide the list of RuleProvider classes using the [getExecuteBefore\(\)](#) or <http://windup.github.io/windup/docs/javadoc/latest/org/jboss/windup/config/RuleProvider.html#getExecuteAfter%28%29getExecuteBefore%28%29> methods.

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

```
public class ExampleRuleProvider extends AbstractRuleProvider
{
    @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

As mentioned above, individual rules are added to a ruleset in the [getConfiguration\(GraphContext context\)](#) method using the [OCPsoft Rewrite](#) ConfigurationBuilder class.

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.
- Otherwise: The **when(condition)** is not met
- Operation: This defines what to **perform()** if the condition is not met.

Rules must define the condition, or *when*, and an operation, or *perform*. However, the otherwise and remainder are optional.

when()

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

The `.when()` clause of a 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.

The `.when()` clause can also subclass [GraphCondition](#). The [Query](#) class extends [GraphCondition](#) and is a convenient way to create a condition. You can also use multiple conditions within one `when()` call using `and()`.

Example:

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

One last but important feature is the ability to use [Gremlin](#) queries. See the [Gremlin Documentation](#) reference manual for more information.

perform()

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

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

For that, various operations are available, which are subclasses of [GraphOperation](#). You can also implement your own operations.

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

An iteration is also an operation, so anywhere an operation is expected, you can use the Iteration. If the Iteration is placed within the perform method of another Iteration, it is called nested iteration.

The following is an example of a nested iteration.

```

.addRule()
    .when(...)
    .perform(
        Iteration.over("list_variable").as("single_var")
        .perform(
            Iteration.over("single_var") //second iteration
            .perform(...).endIteration()
        )
    .endIteration()
);

```

otherwise

As previously mentioned, Windup rules are based on [OCPsoft Rewrite](#). The `.otherwise()` clause allows you to perform something if the condition specified in `.when()` clause is not matched. For more information, see [OCP Rewrite web](#).

The following is an example of an otherwise operation.

```

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

```

Where

The `where()` clause is used to provide information about used parameters within the rule. So for example if you have used a parameter in some condition like for example `JavaClass.references("{myMatch}")`, you may use the where clause to specify what the `myMatch` is like `.where("myMatch").matches("java.lang.String.toString\(.*\)")`.

The following is an example

```

.when(JavaClass.references("{myMatch}").at(TypeReferenceLocation.METHOD))
.perform(...)
.where("myMatch").matches("java.lang.String.toString\(.*\)")

```

+ Please note that within the where clause the regex is used in contrast to `JavaClass.references()` where a windup specific syntax is expected.

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](#).

Variable stack

The communication between the conditions and operations is done using the variable stack that is filled with the output of the condition/s and then given to the Iteration to be processed. Within conditions, you can specify the name of the result iterable that is saved in the stack using `as()` method, the iteration can specify the iterable to iterate over using the `over()` method and even specify the name of for each processed single model of the result being processed. Example:

```
.addRule()
    .when(Query...as("result_list"))
    .perform(
        Iteration.over("result_list").as("single_var")
        ...
    )
);
```

The varstack may be accessed even from the second condition in order to narrow the result of the previous one. After that the iteration may choose which result it wants to iterate over (it is even possible to have multiple iterations listed in the perform, each of which may access different result saved in the variable stack).

```
.addRule()
    .when(Query...as("result_list").and(Query.from("result_list")...as("second_result_list")))
    .perform(
        Iteration.over("second_result_list")
        ...
    )
);
```

Basic Rule Execution Flow Patterns

Single Operation - `operation()`;

No condition or iteration is needed. The following is an example of a single operation.

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

Windup source code example: <https://github.com/windup/windup/blob/master/rules-java/impl/src/main/java/org/jboss/windup/rules/apps/java/config/CopyJavaConfigToGraphRuleProvider.java#L99-L101>

The `copyConfigToGraph` `GraphOperation` used in the `perform()` above is defined before the rule.

Single Conditional Operation - `if(...){ operation() }`

A single condition must be met. The following is an example of a single conditional operation.

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

Windup source code example:

<https://github.com/windup/windup/blob/master/reporting/api/src/main/java/org/jboss/windup/reporting/rules/AttachApplicationRepL41>

Single Iteration - `for(FooModel.class){ ... }`

For simple iterations, you can extend the `IteratingRuleProvider` class to simplify the `perform` code.

```

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(); }
}

```

Windup source code example: <https://github.com/windup/windup/blob/master/rules-java/api/src/main/java/org/jboss/windup/rules/apps/java/scan/provider/DiscoverArchiveManifestFilesRuleProvider.java>

Conditional Iteration - if(...){ for(...) }

```

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

```

Windup source code example: <https://github.com/windup/windup/blob/master/rules-java-ee/addon/src/main/java/org/jboss/windup/rules/apps/javaee/rules/DiscoverEjbAnnotationsRuleProvider.java#L82-L93>

Iteration With a Condition - for(...){ if(...){ ... } }

```

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( ... )
    )

```

Windup source code example: <https://github.com/windup/windup/blob/master/reporting/impl/src/main/java/org/jboss/windup/reporting/rules/rendering/RenderReL66>

Nested Iterations - for(...){ for(...){ ... } }


```

.addRule()
    .when(...)
    .perform(Iteration //first iteration
    .over("list_variable").as("single_var")
    .perform(
        Iteration.over("single_var") //second iteration
        .perform(...).endIteration()
    )
    .endIteration()
);

```

Windup source code example:

<https://github.com/windup/windup/blob/master/config/tests/src/test/java/org/jboss/windup/config/iteration/payload/IterationPayload202>

Create a Basic Java-based Rule Add-on

You can create a Windup 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://ocpsoft.org/rewrite/>
 - The JavaDoc for the Windup API is located here: <http://windup.github.io/windup/docs/javadoc/latest/>

Working examples of Java-based rules can be found in the [Windup quickstarts](#) and [Windup source code](#) repositories.

Create the 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. Be sure to replace `WINDUP_VERSION` with the current version of Windup, for example: `2.2.0.Final`.

```

<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>WINDUP_VERSION</version.windup>
</properties>

```

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

```

<dependencyManagement>
  <dependencies>
    <!-- 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.

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

```

<!-- 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.config</groupId>
        <artifactId>windup-config-xml</artifactId>
        <classifier>forge-addon</classifier>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.jboss.windup.config</groupId>
        <artifactId>windup-config-groovy</artifactId>
        <classifier>forge-addon</classifier>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.jboss.windup.utils</groupId>
        <artifactId>windup-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>windup-rules-java</artifactId>
        <classifier>forge-addon</classifier>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.jboss.windup.rules.apps</groupId>
        <artifactId>windup-rules-java-ee</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.

```
<build>
  <plugins>
    <!-- This plugin makes this artifact a Forge add-on. -->
    <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>
</build>
```

Create the Java RuleProvider Class

1. Within your Maven project, create the Java RuleProvider class.
 - This class can extend [AbstractRuleProvider](#) or one of the following helper classes: [SingleRuleProvider](#) or [IteratingRuleProvider](#).
 - If you prefer not to extend one of these classes, you can implement the [RuleProvider](#) interface.
 - It is recommended that you end the name of the class with RuleProvider . For example:

```
public class MyCustomRuleProvider extends AbstractRuleProvider
{
}
```

2. Provide a constructor for your rule class.
 - In the constructor, you can create a new [RuleProviderMetadata](#) builder instance for this RuleProvider type, using the provided parameters and [RulesetMetadata](#).
 - By default, rules run in the [MigrationRulesPhase](#). If your rule should run earlier during the initial [DiscoveryPhase](#), this can be overridden in the constructor using the `setPhase()` method.
 - Use the `addExecuteAfter()` or `addExecuteBefore()` method to control the order in which the rule is executed,

```
public MyCustomRuleProvider()
{
    super(MetadataBuilder.forProvider(MyCustomRuleProvider.class)
        .setPhase(DiscoveryPhase.class)
        .addExecuteBefore(MyOtherRuleProvider.class));
}
```

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

3. Finally, add rules to the rule provider. Rules are added in the `getConfiguration()` method using the `ConfigurationBuilder.begin().addRule()` code construct.

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

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

- Conditions are specified using `.when()` .
- Actions are performed using `.perform()` .

- 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))
        );
}

```

Working examples of Java-based rules can be found in the [Windup quickstarts](#) and [Windup source code](#) repositories.

- 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

4. Create a `beans.xml` file in the project `META-INF/` directory, for example:

```
PROJECT_DIRECTORY/src/main/resources/META-INF/beans.xml
```

This file tells CDI to scan your add-on for CDI beans. The file can be empty, but it is a good practice to include the basic schema information.

```

<!-- Marker file indicating CDI 1.0 should be enabled -->
<beans xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
</beans>

```

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 [Start Windup in Interactive Mode](#).
- At the Windup console prompt, enter the `addon-build-and-install` command:

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

- You should see a result similar to the following.

```
***SUCCESS*** Addon MyCustomRuleProvider:::2.2.0.Final was installed successfully.
```

Test the Java-based Rule Add-on

Test the Java-based rule add-on against your application file by running Windup in a terminal.

The command uses this syntax:

```
WINDUP_HOME/bin/windup [--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.

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=sam"
```

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.

```
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.

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.

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

Download the Latest Quickstart ZIP

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

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

Fork and Clone the Quickstart GitHub Project

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

1. Click the [Fork](#) link on the [Windup quickstart](#) GitHub page to create the project in your own Git. The forked GitHub repository URL created by the fork should look like this: https://github.com/YOUR_USER_NAME/windup-quickstarts.git
2. Clone your Windup quickstart repository to your local file system:

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

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

```
cd windup-quickstarts/
```

4. If you want to be able to retrieve the 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. Many require only a few minutes of your time!

- Send an email to windup-users@lists.jboss.org and let us know what should Windup migration rules cover.
- Provide example applications to test migration rules.
- Identify application components and problem areas that may be difficult to migrate.
 - Write a short description of these problem migration areas.
 - Write a brief overview describing how to solve the problem migration areas.
- [Try Windup](#) on your application. Be sure to [report any issues](#) you encounter.
- You can contribute to the Windup rules repository.
 - Write a Windup rule to identify or automate a migration process.
 - Create a test for the new rule.
 - Fork the `windup-rulesets` Github repository and put the rule therein.
 - Details are provided in the [Windup Rules Development Guide](#).
- You can also contribute to the project source code.
 - Create a core rule.
 - Improve Windup performance or efficiency.
 - See the [Windup Core Development Guide](#) for information about how to configure your environment and set up the project.

Any level of involvement is greatly appreciated!

Important Links

- Windup wiki: <https://github.com/windup/windup/wiki>
- Windup documentation (generated from the Wiki documentation at the link above):
 - [Windup User Guide](#)
- Windup forums: <https://community.jboss.org/en/windup>
- Windup issue tracker: <https://issues.jboss.org/browse/WINDUP>
- Windup users mailing List: windup-users@lists.jboss.org
- Windup on Twitter: [@JBossWindup](#)
- Windup IRC channel: Server FreeNode (irc.freenode.net), channel #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 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 **Confirm address** 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

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

Ruleset

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

- `rules-java-ee`

- rules-xml

Rules Metadata

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

Rules Pipeline

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

Reporting Terms

Level of effort

The effort required to complete the migration task. *Level of effort* is represented as *story points* in the Windup reports.

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. Story points are covered in more detail in the *Windup Rules Development Guide*.

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.

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

IRC chat

Server: irc.freenode.net

Channel: [#windup](http://irc.freenode.net/#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.
```

Optimize Windup Performance

Overview

Windup performance depends on a number of factors, including hardware configuration, the number and types of files in the application, the size and number of applications to be evaluated, and whether the application contains source or compiled code. For example, a file that is larger than 10 MB may need a lot of time to process.

In general, Windup spends about 40% of the time decompiling classes, 40% of the time executing rules, and the remainder of the time processing other tasks and generating reports. This section describes what you can do to improve the performance of Windup.

Tips to Optimize Performance

Application and Command Line Suggestions

Try these suggestions first before upgrading hardware.

- If possible, execute Windup against the source code instead of the archives. This eliminates the need to decompile additional JARs and archives.
- Specify the `--target` platform on the on the `WINDUP_HOME/bin/windup` or `windup-migrate-app` command line to limit the execution of rules to only those that apply to this target platform.
- Be sure to specify a comma-delimited list of the packages to be evaluated by Windup using the `--packages` argument on the `WINDUP_HOME/bin/windup` or `windup-migrate-app` command line. If you omit this argument, Windup will decompile everything, which has a big impact on performance.
- Specify the `--excludePackages` and `--excludeTags` where possible to exclude them from processing.
- Add additional proprietary packages that should not be processed to the `ignore/proprietary.package-ignore.txt` file in the Windup distribution directory. Windup can still find the references to the packages in the application source code, but avoids the need to decompile and analyze the proprietary classes.

Hardware Upgrade Suggestions

If the steps above do not improve performance, you may need to upgrade your hardware.

- Very large applications that require decompilation have large memory requirements. 8 GB RAM is recommended. This allows 3 - 4 GB RAM for use by the JVM.
- An upgrade from a single or dual-core to a 4-core CPU processor provides better performance.
- Disk space and fragmentation can impact performance. A fast disk, especially a Solid State Drive (SSD), with greater than 4 GB of defragmented disk space should improve performance.