



Red Hat Application Migration Toolkit 4.2

CLI Guide

Learn how to use the Red Hat Application Migration Toolkit to migrate your applications.

Red Hat Application Migration Toolkit 4.2 CLI Guide

Learn how to use the Red Hat Application Migration Toolkit to migrate your applications.

Legal Notice

Copyright © 2019 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide describes how to use the Red Hat Application Migration Toolkit CLI to simplify migration of Java applications.

Table of Contents

CHAPTER 1. INTRODUCTION	4
1.1. ABOUT THE CLI GUIDE	4
1.1.1. Use of RHAMT_HOME in This Guide	4
1.2. ABOUT RED HAT APPLICATION MIGRATION TOOLKIT	4
What is Red Hat Application Migration Toolkit?	4
How Does Red Hat Application Migration Toolkit Simplify Migration?	4
How Do I Learn More?	4
1.3. ABOUT THE CLI	4
CHAPTER 2. GETTING STARTED	5
2.1. PREREQUISITES	5
2.2. INSTALL THE CLI	5
2.3. RUN THE CLI	5
RHAMT Command Examples	6
Running RHAMT on an Application Archive	6
Running RHAMT on Source Code	6
Running Cloud-readiness Rules	6
Override RHAMT Properties	6
RHAMT CLI Bash Completion	6
Enable Bash Completion	6
Enable Persistent Bash Completion	6
RHAMT Help	7
2.4. ACCESS THE REPORT	7
CHAPTER 3. REVIEW THE REPORTS	8
3.1. APPLICATION REPORT	9
3.1.1. Dashboard	9
3.1.2. Issues Report	11
3.1.3. Application Details Report	12
3.1.4. Application Technology Report	14
3.1.5. Application Dependencies Graph Report	14
3.1.6. Source Report	16
3.2. TECHNOLOGY REPORT	16
3.3. DEPENDENCIES GRAPH REPORT	17
3.4. ARCHIVES SHARED BY MULTIPLE APPLICATIONS	18
3.5. RULE PROVIDER EXECUTION OVERVIEW	19
3.6. USED FREEMARKER FUNCTIONS AND DIRECTIVES	19
3.7. SEND FEEDBACK FORM	20
CHAPTER 4. EXPORT THE REPORT IN CSV FORMAT	22
4.1. EXPORT THE REPORT	22
Access the Report from the Application Report	22
4.2. IMPORT THE CSV FILE INTO A SPREADSHEET PROGRAM	22
4.3. OVERVIEW OF THE CSV DATA STRUCTURE	22
CHAPTER 5. MAVENIZE YOUR APPLICATION	24
5.1. GENERATE THE MAVEN PROJECT STRUCTURE	24
5.2. REVIEW THE MAVEN PROJECT STRUCTURE	24
Root POM File	25
BOM File	25
Application POM Files	26

CHAPTER 6. OPTIMIZE RHAMT PERFORMANCE	27
6.1. TIPS TO OPTIMIZE PERFORMANCE	27
6.1.1. Application and Command-line Suggestions	27
6.1.2. Hardware Upgrade Suggestions	27
6.2. CONFIGURE RHAMT TO EXCLUDE FILES AND PACKAGES	27
6.2.1. Exclude Packages	28
6.2.2. Exclude Files	28
6.2.3. Locations to Search for Exclusion	28
APPENDIX A. REFERENCE MATERIAL	29
A.1. RHAMT COMMAND-LINE ARGUMENTS	29
A.1.1. Specify the Input	32
A.1.2. Specify the Output Directory	33
A.1.3. Set the Source Technology	33
A.1.4. Set the Target Technology	34
A.1.5. Select Packages	34
A.2. RULE STORY POINTS	35
A.2.1. What are Story Points?	35
A.2.2. How Story Points are Estimated in Rules	35
A.2.3. Task Category	36
A.3. ADDITIONAL RESOURCES	36
A.3.1. Get Involved	36
A.3.2. Important Links	37
A.3.3. Known RHAMT Issues	37
A.3.4. Report Issues with RHAMT	37
A.3.4.1. Create a JIRA Issue	38

CHAPTER 1. INTRODUCTION

1.1. ABOUT THE CLI GUIDE

This guide is for engineers, consultants, and others who want to use Red Hat Application Migration Toolkit (RHAMT) to migrate Java applications or other components. It describes how to install and run the CLI, review the generated reports, and take advantage of additional features.

1.1.1. Use of `RHAMT_HOME` in This Guide

This guide uses the `RHAMT_HOME` replaceable variable to denote the path to your RHAMT installation. The installation directory is the `rhamt-cli-4.2.1.Final` directory where you extracted the RHAMT ZIP distribution.

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

1.2. ABOUT RED HAT APPLICATION MIGRATION TOOLKIT

What is Red Hat Application Migration Toolkit?

Red Hat Application Migration Toolkit (RHAMT) is an extensible and customizable rule-based tool that helps simplify migration of Java applications.

RHAMT examines application artifacts, including project source directories and application archives, then produces an HTML report that highlights areas needing changes. RHAMT 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 Red Hat Application Migration Toolkit Simplify Migration?

Red Hat Application Migration Toolkit 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 Red Hat JBoss Enterprise Application Platform.

How Do I Learn More?

See the [Getting Started Guide](#) to learn more about the features, supported configurations, system requirements, and available tools in the Red Hat Application Migration Toolkit.

1.3. ABOUT THE CLI

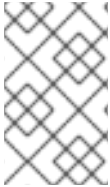
The CLI is a command-line tool in the Red Hat Application Migration Toolkit that allows users to assess and prioritize migration and modernization efforts for applications. It provides numerous reports that highlight the analysis without the overhead of the other tools. The CLI includes a wide array of customization options, and allows you to finely tune the RHAMT analysis options or integrate with external automation tools.

CHAPTER 2. GETTING STARTED

2.1. PREREQUISITES

Before installing the CLI, verify that you meet the following prerequisites.

- Java Platform, JRE version 8+
- A minimum of 4 GB RAM; 8 GB recommended



NOTE

If you are running macOS, it is recommended to set the maximum number of user processes, **maxproc**, to at least **2048**, and the maximum number of open files, **maxfiles**, to **100000**.

2.2. INSTALL THE CLI

1. Download the CLI from the [RHAMT Download page](#).
2. Extract the ZIP file to a directory of your choice.

The extracted directory is known as **RHAMT_HOME** in this guide.

2.3. RUN THE CLI

Use the following steps to run RHAMT against your application.

1. Open a terminal and navigate to the **RHAMT_HOME/bin/** directory.
2. Execute the **rhamt-cli** script, or **rhamt-cli.bat** for Windows, and specify the appropriate arguments.

```
$ ./rhamt-cli --input /path/to/jee-example-app-1.0.0.ear --output  
/path/to/output --source weblogic --target eap:6 --packages com.acme  
org.apache
```

- **--input**: The application to be evaluated. See the **--input** argument description.
- **--output**: The output directory for the generated reports. See the **--output** argument description.
- **--source**: The source technology for the application migration. See the **--source** argument description.
- **--target**: The target technology for the application migration. See the **--target** argument description.
- **--packages**: The packages to be evaluated. This argument is highly recommended to improve performance. See the **--packages** argument description.

See [RHAMT Command-line Arguments](#) for a detailed description of all available command-line arguments.

3. [Access the report.](#)

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

RHAMT Command Examples

Running RHAMT on an Application Archive

The following command analyzes the **com.acme** and **org.apache** packages of the [jee-example-app-1.0.0.ear](#) example EAR archive for migrating from JBoss EAP 5 to JBoss EAP 7.

```
$ RHAMT_HOME/bin/rhamt-cli --input /path/to/jee-example-app-1.0.0.ear --
output /path/to/report-output/ --source eap:5 --target eap:7 --packages
com.acme org.apache
```

Running RHAMT on Source Code

The following command analyzes the **org.jboss.seam** packages of the [seam-booking-5.2](#) example source code for migrating to JBoss EAP 6.

```
$ RHAMT_HOME/bin/rhamt-cli --sourceMode --input /path/to/seam-booking-5.2/
--output /path/to/report-output/ --target eap:6 --packages org.jboss.seam
```

Running Cloud-readiness Rules

The following command analyzes the **com.acme** and **org.apache** packages of the [jee-example-app-1.0.0.ear](#) example EAR archive for migrating to JBoss EAP 7. It also evaluates for cloud readiness.

```
$ RHAMT_HOME/bin/rhamt-cli --input /path/to/jee-example-app-1.0.0.ear --
output /path/to/report-output/ --target eap:7 --target cloud-readiness --
packages com.acme org.apache
```

Override RHAMT Properties

To override the default *Fernflower* decompiler, pass the **-Dwindup.decompiler** argument on the command line. For example, to use the *Procyon* decompiler, use the following syntax:

```
$ RHAMT_HOME/bin/rhamt-cli -Dwindup.decompiler=procyon --input
INPUT_ARCHIVE_OR_DIRECTORY --output OUTPUT_REPORT_DIRECTORY --target
TARGET_TECHNOLOGY --packages PACKAGE_1 PACKAGE_2
```

RHAMT CLI Bash Completion

The RHAMT CLI provides an option to enable bash completion for Linux systems, allowing the [RHAMT command-line arguments](#) to be auto completed by pressing Tab when entering the commands. For instance, when bash completion is [enabled](#), entering the following displays a list of available arguments.

```
$ RHAMT_HOME/bin/rhamt-cli [TAB]
```

Enable Bash Completion

To enable bash completion for the current shell, execute the following command. After the prompt returns, follow the steps in [Run the CLI](#).

```
$ source RHAMT_HOME/bash-completion/rhamt-cli
```

Enable Persistent Bash Completion

The following commands allow bash completion to persist across restarts; however, if bash completion is desired for the current shell then the steps in [Enable Bash Completion](#) must be followed.

- To enable bash completion for a specific user across system restarts, include the following line in that user's `~/.bashrc` file.

```
source RHAMT_HOME/bash-completion/rhamt-cli
```

- To enable bash completion for all users across system restarts, copy the Red Hat Application Migration Toolkit CLI bash completion file to the `/etc/bash_completion.d/` directory. By default, this directory is only writable by the root user.

```
# cp RHAMT_HOME/bash-completion/rhamt-cli /etc/bash_completion.d/
```

RHAMT Help

To see the complete list of available arguments for the `rhamt-cli` command, open a terminal, navigate to the `RHAMT_HOME` directory, and execute the following command:

```
$ RHAMT_HOME/bin/rhamt-cli --help
```

2.4. ACCESS THE REPORT

When you execute Red Hat Application Migration Toolkit, the report is generated in the `OUTPUT_REPORT_DIRECTORY` that you specify using the `--output` argument in the command line. Upon completion of execution, you will see the following message in the terminal with the location of the report.

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

The output directory contains the following files and subdirectories:

```
OUTPUT_REPORT_DIRECTORY/
├─ index.html           // Landing page for the report
├─ EXPORT_FILE.csv     // Optional export of data in CSV format
├─ archives/          // Archives extracted from the application
├─ mavenized/         // Optional Maven project structure
├─ reports/           // Generated HTML reports
├─ stats/             // Performance statistics
```

Use a browser to open the `index.html` file located in the report output directory. See [Review the Reports](#) for information on navigating the RHAMT reports.

CHAPTER 3. REVIEW THE REPORTS

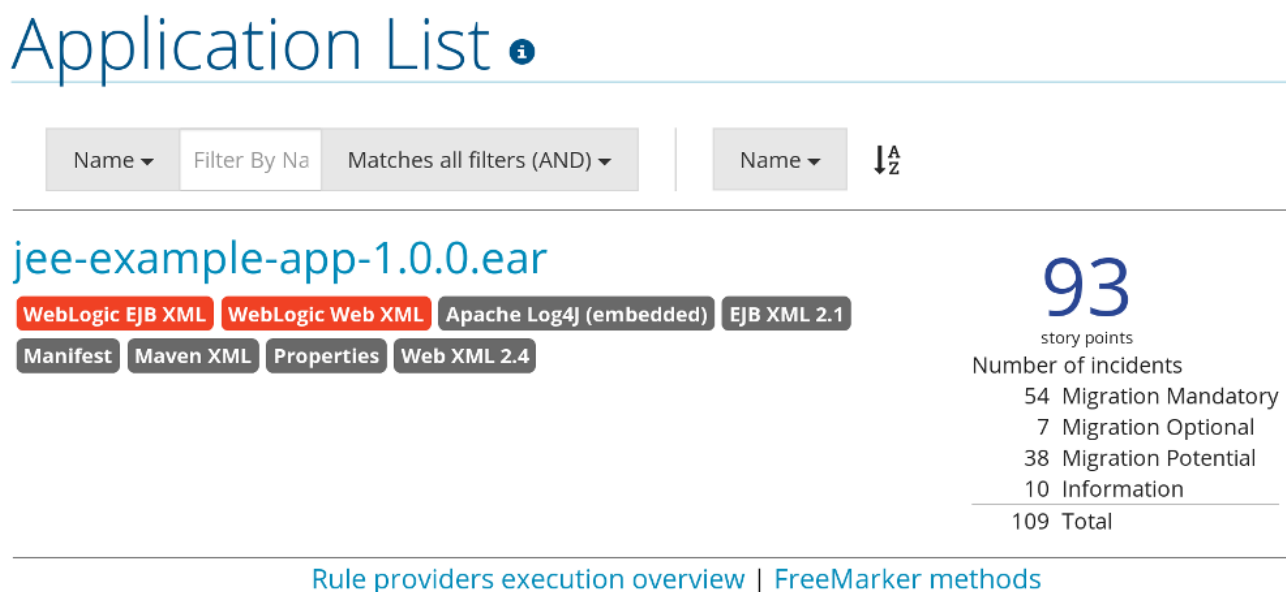
The report examples shown in the following sections are a result of analyzing the `com.acme` and `org.apache` packages in the `jee-example-app-1.0.0.ear` example application, which is located in the RHAMT GitHub source repository.

The report was generated using the following command.

```
$ RHAMT_HOME/bin/rhamt-cli --input /home/username/rhamt-cli-source/test-files/jee-example-app-1.0.0.ear/ --output /home/username/rhamt-cli-reports/jee-example-app-1.0.0.ear-report --target eap:6 --packages com.acme org.apache
```

Use a browser to open the `index.html` file located in the report output directory. This opens a landing page that lists the applications that were processed. Each row contains a high-level overview of the story points, number of incidents, and technologies encountered in that application.

Figure 3.1. Application List



NOTE

The incidents and estimated story points change as new rules are added to RHAMT. The values here may not match what you see when you test this application.

The following table lists all of the reports and pages that can be accessed from this main RHAMT landing page. Click on the name of the application, `jee-example-app-1.0.0.ear`, to view the [application report](#).

Page	How to Access
Application	Click on the name of the application.
Technology Report	Click on the Technologies link at the top of the page.
Dependencies Graph Report	Click on the Dependencies Graph link at the top of the page.

Page	How to Access
Archives shared by multiple applications	Click on the Archives shared by multiple applications link. Note that this link is only available when there are shared archives across multiple applications.
Rule Providers Execution Overview	Click on the Rule providers execution overview link at the bottom of the page.
Used FreeMarker Functions and Directives	Click on the FreeMarker methods link at the bottom of the page.
Send Feedback form	Click on the Send Feedback link in the top navigation bar to open a form that allows you to submit feedback to the RHAMT team.

Note that if an application shares archives with other analyzed applications, you will see a breakdown of how many story points are from shared archives and how many are unique to this application.

Figure 3.2. Shared Archives



Information about the archives that are shared among applications can be found in the [Archives Shared by Multiple Applications](#) reports.

3.1. APPLICATION REPORT

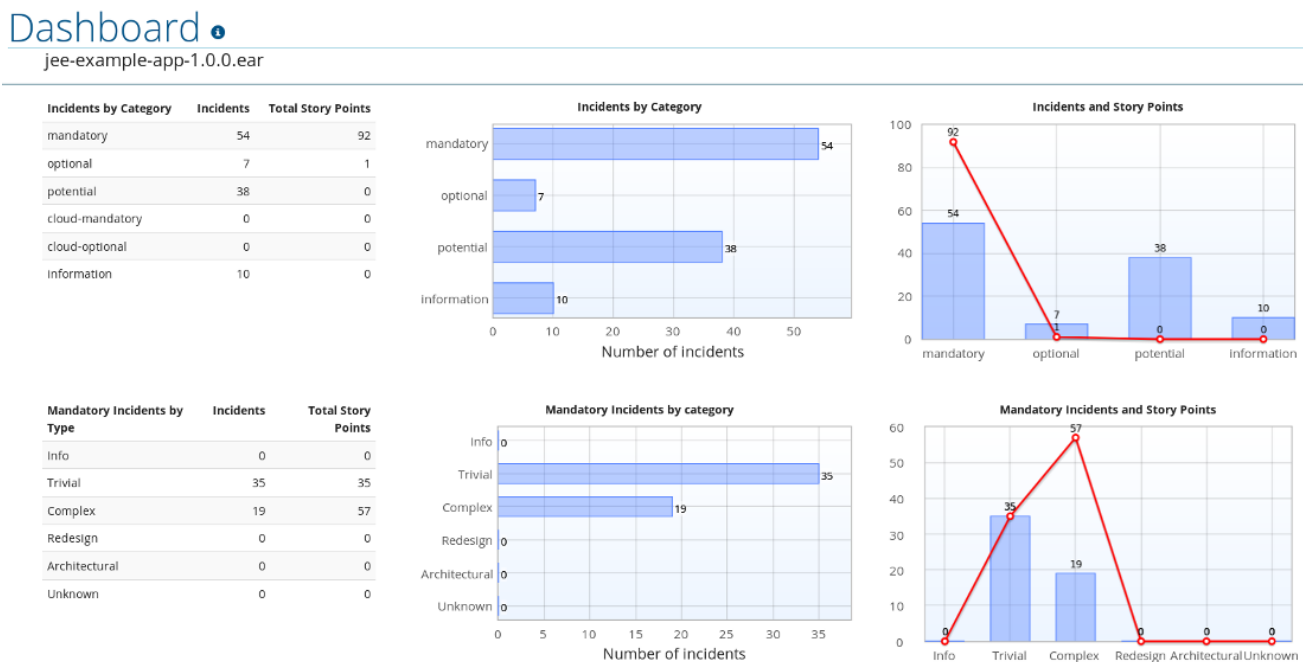
3.1.1. Dashboard

Access this report from the report landing page by clicking on the application name in the **Application List**.

The dashboard gives an overview of the entire application migration effort. It summarizes:

- The incidents and story points by category
- The incidents and story points by level of effort of the suggested changes
- The incidents by package

Figure 3.3. Dashboard



The top navigation bar lists the various reports that contain additional details about the migration of this application. Note that only those reports that are applicable to the current application will be available.

Report	Description
Issues	Provides a concise summary of all issues that require attention.
Application Details	Provides a detailed overview of all resources found within the application that may need attention during the migration.
Technologies	Displays all embedded libraries grouped by functionality, allowing you to quickly view the technologies used in each application.
Dependencies Graph	Displays a graph of all Java-packaged dependencies found within the analyzed applications. This graph also demonstrates the relations of each dependency, allowing you to view nested and multiple dependencies.
Dependencies	Displays all Java-packaged dependencies found within the application.
Unparsable	Shows all files that RHAMT could not parse in the expected format. For instance, a file with a <code>.xml</code> or <code>.wsdl</code> suffix is assumed to be an XML file. If the XML parser fails, the issue is reported here and also where the individual file is listed.
Remote Services	Displays all remote services references that were found within the application.
EJBs	Contains a list of EJBs found within the application.

Report	Description
JBPM	Contains all of the JBPM-related resources that were discovered during analysis.
JPA	Contains details on all JPA-related resources that were found in the application.
Hibernate	Contains details on all Hibernate-related resources that were found in the application.
Server Resources	Displays all server resources (for example, JNDI resources) in the input application.
Spring Beans	Contains a list of Spring beans found during the analysis.
Hard-coded IP Addresses	Provides a list of all hard-coded IP addresses that were found in the application.
Ignored Files	Lists the files found in the application that, based on certain rules and RHAMT configuration, were not processed. See the <code>--userIgnorePath</code> option for more information.
About	Describes the current version of RHAMT and provides helpful links for further assistance.

3.1.2. Issues Report

Access this report from the dashboard by clicking the **Issues** link.

This report includes details about every issue that was raised by the selected migration paths. The following information is provided for each issue encountered.

- A title to summarize the issue.
- The total number of incidents, or times the issue was encountered.
- The [rule story points](#) to resolve a single instance of the issue.
- The estimated level of effort to resolve the issue.
- The total story points to resolve every instance encountered. This is calculated by multiplying the number of incidents found by the story points per incident.

Figure 3.4. Issues Report

Issues ⁱ

jee-example-app-1.0.0.ear

Migration Mandatory				
Issue by Category	Incidents Found	Story Points per Incident	Level of Effort	Total Story Points
WebLogic proprietary logger (NonCatalogLogger)	19	1	Trivial change or 1-1 library swap	19
Call of JNDI lookup	5	1	Trivial change or 1-1 library swap	5
WebLogic proprietary T3 JNDI binding	3	3	Complex change with documented solution	9

Each reported issue may be expanded, by clicking on the title, to obtain additional details. The following information is provided.

- A list of files where the incidents occurred, along with the number of incidents within each file. If the file is a Java source file, then clicking the filename will direct you to the corresponding [Source Report](#).
- A detailed description of the issue. This description outlines the problem, provides any known solutions, and references supporting documentation regarding either the issue or resolution.
- A direct link, entitled **Show Rule**, to the rule that generated the issue.

Figure 3.5. Expanded Issue

Issue by Category	Incidents Found	Story Points per Incident	Level of Effort	Total Story Points
WebLogic proprietary logger (NonCatalogLogger)	19	1	Trivial change or 1-1 library swap	19
File	Incidents Found	Hint		
com.acme.anvil.service.ProductCatalogBean	5	Issue Detail: WebLogic proprietary logger (NonCatalogLogger) Show Rule		
com.acme.anvil.listener.AnvilWebStartupListener	6	The WebLogic <code>NonCatalogLogger</code> is not supported on JBoss EAP, and should be migrated to a supported logging framework, such as the JDK Logger or JBoss Logging: <pre>import java.util.logging.Logger; Logger LOG = Logger.getLogger("MyLogger");</pre>		
com.acme.anvil.LoginFilter	3			
com.acme.anvil.AuthenticateFilter	5	<ul style="list-style-type: none"> • JBoss Logging Quickstart • JDK Logging Documentation 		

Issues are sorted into four categories by default. Information on these categories is available at [Task Category](#).

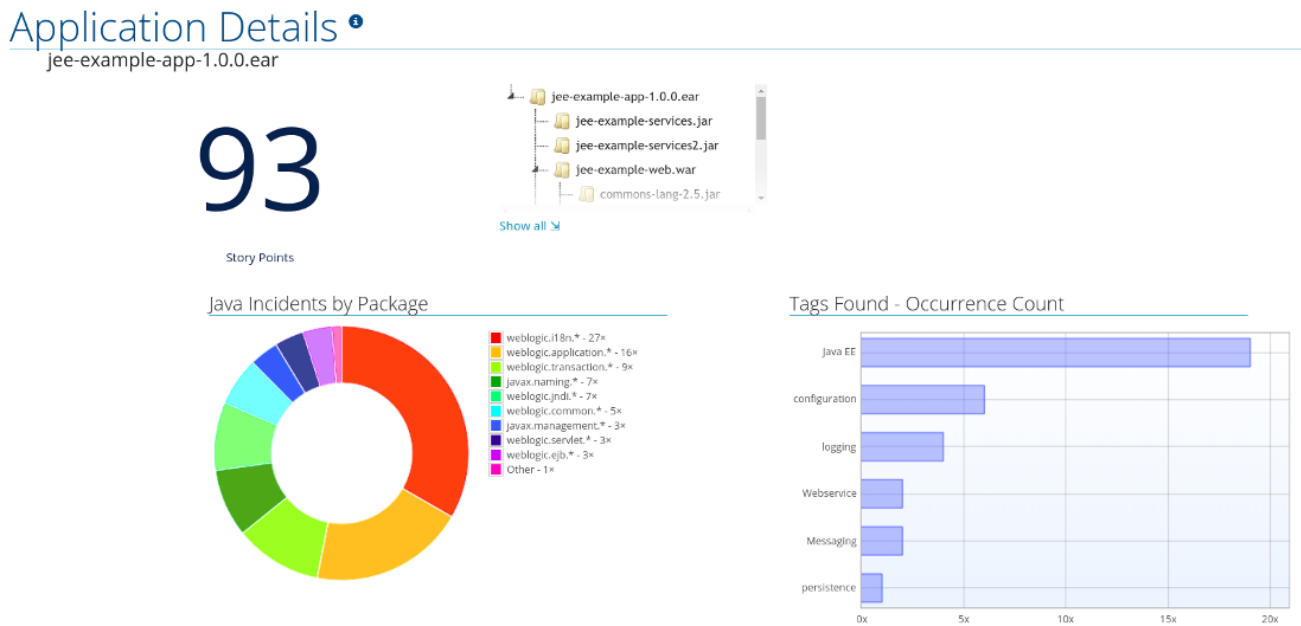
3.1.3. Application Details Report

Access this report from the dashboard by clicking the **Application Details** link.

The report lists the story points, the Java incidents by package, and a count of the occurrences of the technologies found in the application. Next is a display of application messages generated during the

migration process. Finally, there is a breakdown of this information for each archive analyzed during the process.

Figure 3.6. Application Details Report



Expand the `jee-example-app-1.0.0.ear/jee-example-services.jar` to review the story points, Java incidents by package, and a count of the occurrences of the technologies found in this archive. This summary begins with a total of the story points assigned to its migration, followed by a table detailing the changes required for each file in the archive. The report contains the following columns.

Column Name	Description
Name	The name of the file being analyzed.
Technology	The type of file being analyzed, for example, Decompiled Java File or Properties .
Issues	Warnings about areas of code that need review or changes.
Story Points	Level of effort required to migrate the file. See Rule Story Points for more details.

Note that if an archive is duplicated several times in an application, it will be listed just once in the report and will be tagged with **[Included Multiple Times]**.

Figure 3.7. Duplicate Archive in an Application

[Included Multiple Times] jee-example-services.jar (61 story points)

61
Story Points

Maven coordinates	org.windup.example:jee-example-services:1.0.0
Organization	Unknown
Name	JEE Example EJB Services
Version	1.0.0
Links	
Description	
Duplicates	test-app.ear/copy1/jee-example-services.jar test-app.ear/copy2/jee-example-services.jar

The story points for archives that are duplicated within an application will be counted only once in the total story point count for that application.

3.1.4. Application Technology Report

Access this report from the dashboard by clicking the **Technologies** link.

The report lists the occurrences of technologies, grouped by function, in the analyzed application. It is an overview of the technologies found in the application, and is designed to assist users in quickly understanding each application's purpose.

The below image shows the technologies used in the **jee-example-app**.

Figure 3.8. Technologies in an Application

Technologies
jee-example-app-1.0.0.ear

View **Connect** **Store** **Sustain** **Execute**

Java EE

- Web**: Web XML File: 1
- EJB**: Stateless EJB: 4, Message-Driven: 2
- Transactions**: JTA: 3

Embedded

- Logging**: Log4j: 2

3.1.5. Application Dependencies Graph Report

The analyzed applications' dependencies are shown in this report, accessible from the **Dependencies Graph** link from the dashboard.

It includes a list of all WARs and JARs, including third-party JARs, and graphs the relations between each of the included files. Each circle in the graph represents a unique dependency defined in the application.

The below image shows the dependencies used in the **jee-example-app**, with the selected application in the center of the graph.

Figure 3.9. Graph of Dependencies in an Application

Dependencies Graph i

jee-example-app-1.0.0.ear

Selected: jee-example-app-1.0.0.ear



Interacting with the Dependency Graph

The dependency graph may be adjusted by using any of the following.

- Clicking a dependency will display the name of the application in the upper-left corner. While selected the dependency will have a shaded circle identifying it, as seen on the center in the above image.
- Clicking and dragging a circle will reposition it. Releasing the mouse will fix the dependency to the cursor's location.
- Clicking on a fixed dependency will release it, returning the dependency to its default distance from the application.
- Double clicking anywhere will return the entire graph to the default state.

- Clicking on any item in the legend will enable or disable all items of the selected type. For instance, selecting the embedded WARs icon will disable all embedded WARs if these are enabled, and will enable these dependencies if they are disabled.

3.1.6. Source Report

The analysis of the `jee-example-services.jar` lists the files in the JAR and the warnings and story points assigned to each one. Notice the

`com.acme.anvil.listener.AnvilWebLifecycleListener` file, at the time of this test, has 22 warnings and is assigned 16 story points. Click on the file link to see the detail.


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

In this example, warnings appear at various import statements, declarations, and method calls. Each warning describes the issue and the action that should be taken.

Figure 3.10. Source Report

Source Report

jee-example-app-1.0.0.ear/jee-example-services.jar/com/acme/anvil/listener/AnvilWebLifecycleListener.java

 This report displays what Red Hat Application Migration Toolkit found in individual files. Each item is shown below the line it was found on, and next to it, you may find a link to the rule which it was found by.

Information

16

Story Points

Technologies

Decompiled Java File

```
01. package com.acme.anvil.listener;
02.
03. import com.acme.anvil.management.AnvilInvokeBeanImpl;
04. import java.util.Properties;
05. import javax.management.MBeanServer;
06. import javax.management.ObjectName;
07. import javax.naming.InitialContext;
08. import javax.naming.NamingException;
09. import org.apache.log4j.Logger;
10. import weblogic.application.ApplicationLifecycleEvent;
```

WebLogic ApplicationLifecycleEvent

WebLogic ApplicationLifecycleEvent must be replaced with standard Java EE ServletContextEvent. Otherwise, a custom solution using CDI's ApplicationScoped beans or EJB's @Startup beans is required in order to propagate a custom event object because ServletContextEvent types are not extendible in the standard Java EE programming model.

Use a `javax.servlet.ServletContextListener` with `@javax.annotation.servlet.WebListener`, or an EJB 3.1 `@javax.ejb.Startup` `@javax.ejb.Singleton` service bean.

- Migrate [WebLogic ApplicationLifecycleEvent to standard EJB with JBoss EAP](#)
- [Java EE ServletContextEvent JavaDoc](#)
- [WebLogic custom ApplicationLifecycleEvent Documentation](#)

3.2. TECHNOLOGY REPORT

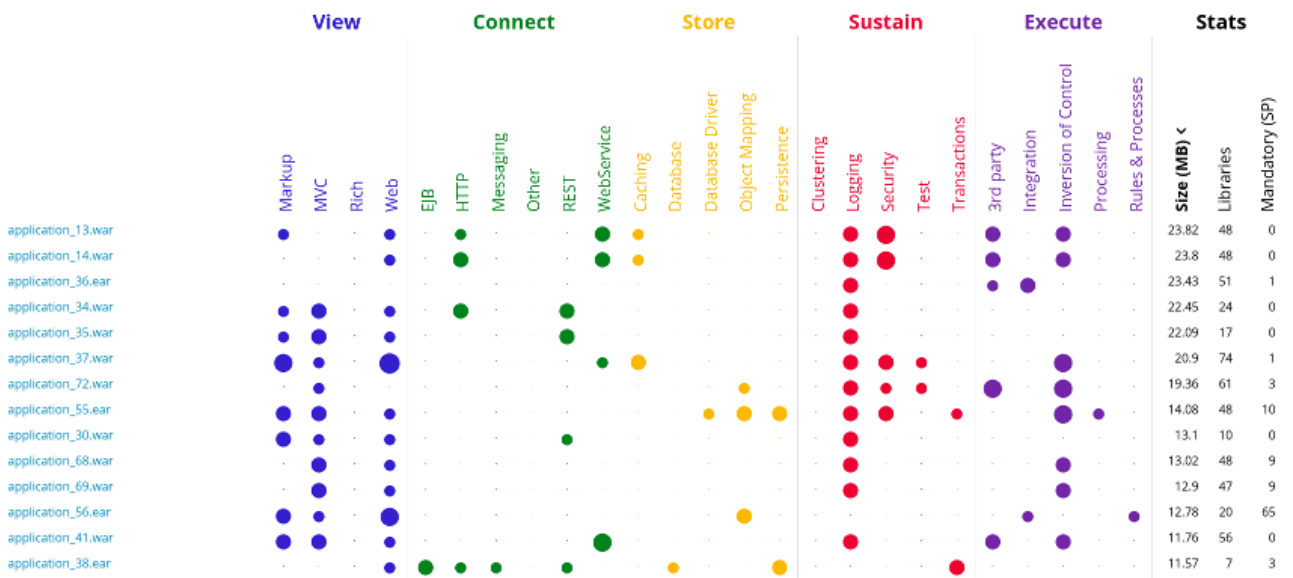
Access this report from the report landing page by clicking the **Technologies** link.

This report provides an aggregate listing of the technologies used, grouped by function, for the analyzed applications. It shows how the technologies are distributed, and is typically reviewed after analyzing a large number of applications to group the applications and identify patterns. It also shows the size, number of libraries, and story point totals of each application.

Clicking any of the headers, such as **Markup**, sorts the results in descending order. Selecting the same header again will resort the results in ascending order. The currently selected header is identified in bold, next to a directional arrow, indicating the direction of the sort.

Figure 3.11. Technologies Used Across Multiple Applications

Technologies ^o



3.3. DEPENDENCIES GRAPH REPORT

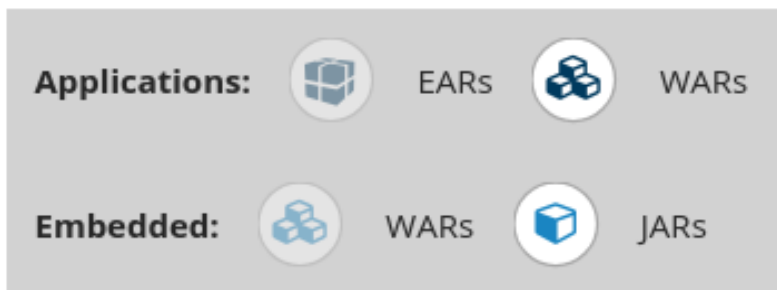
Access this report from the report landing page by clicking the **Dependencies Graph** link.

It includes a list of all WARs and JARs, and graphs the relations between each of the included files. Each circle in the graph represents a unique dependency defined in the application. If a file is included as a dependency in multiple applications, these are linked in the graph.

In the below image we can see two distinct groups. On the left half we see a single WAR that defines several JARs as dependencies. On the right half we see the same dependencies used by multiple WARs, one of which is the selected **overlord-commons-auth-2.0.11.Final.jar**.

Figure 3.12. Dependency Graph Across Multiple Applications

Selected: overlord-commons-auth-2.0.11.Final.jar



The dependency graph may be adjusted by using any of the following.


- Clicking a dependency will display the name of the application in the upper-left corner. While selected the dependency will have a shaded circle identifying it, as seen on the center in the above image.
- Clicking and dragging a circle will reposition it. Releasing the mouse will fix the dependency to the cursor's location.
- Clicking on a fixed dependency will release it, returning the dependency to its default distance from the application.
- Double clicking anywhere will return the entire graph to the default state.
- Clicking on any item in the legend will enable or disable all items of the selected type. For instance, selecting the embedded WARs icon will disable all embedded WARs if these are enabled, and will enable these dependencies if they are disabled.

3.4. ARCHIVES SHARED BY MULTIPLE APPLICATIONS

Access these reports from the report landing page by clicking the **Archives shared by multiple applications** link. Note that this link is only available if there are applicable shared archives.

Figure 3.13. Archives Shared by Multiple Applications

Shared Libraries

 This section groups all issues found in libraries included in multiple applications.

Archives shared by multiple applications

WebLogic EJB XML WebLogic Web XML EJB XML 2.1
Manifest Maven XML Properties Web XML 2.4

79
story points

Number of incidents
48 Migration Mandatory
13 Migration Optional
40 Migration Potential
101 Total

This allows you to view the detailed reports for all archives that are shared across multiple applications.


3.5. RULE PROVIDER EXECUTION OVERVIEW

Access this report from the report landing page by clicking the **Rule providers execution overview** link.

This report provides the list of rules that executed when running the RHAMT migration command against the application.

Figure 3.14. Rule Provider Execution Overview

Rule Providers Execution Overview

 This report lists "rule providers", or sets of Red Hat Application Migration Toolkit rules. They may originate from a `.windup.xml` or `.rhamt.xml` file or a Java class implementing `RuleProvider`.

Phase: DependentPhase

Phase: InitializationPhase

IgnoredArchivesConfigLoadingRuleProvider

Phase: InitializationPhase

Rule-ID	Rule	Statistics	Status?	Result?	Failure Cause
IgnoredArchivesConfigLoadingRuleProvider_1	<pre>addRule() .perform(org.jboss.windup.rules.apps.java.archives.config .IgnoredArchivesConfigLoadingRuleProvider\$1@29a5ccf3) withId("IgnoredArchivesConfigLoadingRuleProvider_1")</pre>	Vertices Created: 0 Edges Created: 0 Vertices Removed: 0 Edges Removed: 0	Condition met.	success	


3.6. USED FREEMARKER FUNCTIONS AND DIRECTIVES

Access this report from the report landing page by clicking the **Red Hat Application Migration Toolkit FreeMarker methods** link.

This report lists all the registered functions and directives that were used to build the report. It is useful for debugging purposes or if you plan to build your own custom report.

Figure 3.15. FreeMarker Functions and Directives

Used FreeMarker Functions and Directives

 This report shows the custom Freemarker extensions created for and used by Red Hat Application Migration Toolkit.

Functions	
Function Name	Description
iterableToList	Turns the given Iterable into a List. (Implemented by: org.jboss.windup.reporting.freemarker.IterableToListMethod)
getMigrationEffortPointsForFile	Takes a FileModel as a parameter and returns an int containing the effort estimate for this file. (Implemented by: org.jboss.windup.reporting.freemarker.GetEffortForFile)
getAllFreeMarkerDirectives	This method takes no parameters, and returns a List of hashes containing a 'name', 'class', and 'description' field. (Implemented by: org.jboss.windup.reporting.freemarker.GetAllFreeMarkerDirectivesMethod)
generateGUID	Generates a unique identifier as a String. (Implemented by: org.jboss.windup.reporting.freemarker.GenerateGUIDMethod)


3.7. SEND FEEDBACK FORM

Access this feedback form from the report landing page by clicking the **Send feedback** link.

This form allows you to rate the product, talk about what you like, and make suggestions for improvements.

Figure 3.16. Send Feedback Form

Got feedback?

 Please provide your feedback below:

Rate this page* 😄 Awesome! 😊 Good 😐 Meh! 😞 Bad 🤬 Horrible!

What do you like?*

What needs to be improved?*

Attach file No file selected.

Name

Email

Include data about your current environment, like the browser and page URL. This helps us understand your feedback better.

[What is included in the data about my current environment?](#)

CHAPTER 4. EXPORT THE REPORT IN CSV FORMAT

RHAMT provides the ability to export the report data, including the classifications and hints, to a flat file on your local file system. The export function currently supports the CSV file format, which presents the report data as fields separated by commas (,).

The CSV file can be imported and manipulated by spreadsheet software such as Microsoft Excel, OpenOffice Calc, or LibreOffice Calc. Spreadsheet software provides the ability to sort, analyze, evaluate, and manage the result data from an RHAMT report.

4.1. EXPORT THE REPORT

To export the report into a CSV file, run RHAMT with `--exportCSV` argument. A CSV file will be created in the directory specified by the `--output` argument for each application analyzed. All discovered issues, spanning all the analyzed applications, will be included in `AllIssues.csv` found in the root directory of the report.

Access the Report from the Application Report

If the CSV report is exported, then all of the CSV issues are available for download in the [Issues Report](#). These issues may be downloaded by selecting the **Download All Issues CSV** button in the top-right section of the issues report, as seen in the following image.

Figure 4.1. Issues Report with CSV Download

Issue by Category	Incidents Found	Story Points per Incident	Level of Effort	Total Story Points
JMX MBean object name (javax.management.ObjectName)	1	1	Trivial change or 1-1 library swap	1

4.2. IMPORT THE CSV FILE INTO A SPREADSHEET PROGRAM

1. Launch the spreadsheet software, for example, Microsoft Excel.
2. Choose **File** → **Open**.
3. Browse to the CSV exported file and select it.
4. The data is now ready to analyze in the spreadsheet software.

For more information or to resolve any issues, check the help for your spreadsheet software.

4.3. OVERVIEW OF THE CSV DATA STRUCTURE

The CSV formatted output file contains the following data fields:

Rule Id

The ID of the rule that generated the given item.

Problem type

hint or *classification*

Title

The title of the *classification* or *hint*. This field summarizes the issue for the given item.

Description

The detailed description of the issue for the given item.

Links

URLs that provide additional information about the issue. A link consists of two attributes: the link and a description of the link.

Application

The name of the application for which this item was generated.

File Name

The name of the file for the given item.

File Path

The file path for the given item.

Line

The line number of the file for the given item.

Story points

The number of story points, which represent the level of effort, assigned to the given item.

CHAPTER 5. MAVENIZE YOUR APPLICATION

RHAMT provides the ability to generate an Apache Maven project structure based on the application provided. This will create a directory structure with the necessary Maven Project Object Model (POM) files that specify the appropriate dependencies.

Note that this feature is not intended to create a final solution for your project. It is meant to give you a starting point and identify the necessary dependencies and APIs for your application. Your project may require further customization.

5.1. GENERATE THE MAVEN PROJECT STRUCTURE

You can generate a Maven project structure for the provided application by passing in the `--mavenize` flag when executing RHAMT.

The following example runs RHAMT using the `jee-example-app-1.0.0.ear` test application.

```
$ RHAMT_HOME/bin/rhamt-cli --input /path/to/jee-example-app-1.0.0.ear --
output /path/to/output --target eap:6 --packages com.acme org.apache --
mavenize
```

This generates the Maven project structure in the `/path/to/output/mavenized` directory.



NOTE

You can only use the `--mavenize` option when providing a compiled application for the `--input` argument. This feature is not available when running RHAMT against source code.

You can also use the `--mavenizeGroupId` option to specify the `<groupId>` to be used for the POM files. If unspecified, RHAMT will attempt to identify an appropriate `<groupId>` for the application, or will default to `com.mycompany.mavenized`.

5.2. REVIEW THE MAVEN PROJECT STRUCTURE

The `/path/to/output/mavenized/APPLICATION_NAME/` directory will contain the following items:

- A [root POM file](#). This is the `pom.xml` file at the top-level directory.
- A [BOM file](#). This is the POM file in the directory ending with `-bom`.
- One or more [application POM files](#). Each module has its POM file in a directory named after the archive.

The example `jee-example-app-1.0.0.ear` application is an EAR archive that contains a WAR and several JARs. There is a separate directory created for each of these artifacts. Below is the Maven project structure created for this application.

```
/path/to/output/mavenized/jee-example-app/
  jee-example-app-bom/pom.xml
  jee-example-app-ear/pom.xml
  jee-example-services2-jar/pom.xml
```

```
jee-example-services-jar/pom.xml
jee-example-web-war/pom.xml
pom.xml
```

Review each of the generated files and customize as appropriate for your project. To learn more about Maven POM files, see the [Introduction to the POM](#) section of the Apache Maven documentation.

Root POM File

The root POM file for the `jee-example-app-1.0.0.ear` application can be found at `/path/to/output/mavenized/jee-example-app/pom.xml`. This file identifies the directories for all of the project modules.

The following modules are listed in the root POM for the example `jee-example-app-1.0.0.ear` application.

```
<modules>
  <module>jee-example-app-bom</module>
  <module>jee-example-services2-jar</module>
  <module>jee-example-services-jar</module>
  <module>jee-example-web-war</module>
  <module>jee-example-app-ear</module>
</modules>
```



NOTE

Be sure to reorder the list of modules if necessary so that they are listed in an appropriate build order for your project.

The root POM is also configured to use the [Red Hat JBoss Enterprise Application Platform Maven repository](#) to download project dependencies.

BOM File

The Bill of Materials (BOM) file is generated in the directory ending in `-bom`. For the example `jee-example-app-1.0.0.ear` application, the BOM file can be found at `/path/to/output/mavenized/jee-example-app/jee-example-app-bom/pom.xml`. The purpose of this BOM is to have the versions of third-party dependencies used by the project defined in one place. For more information on using a BOM, see the [Introduction to the Dependency Mechanism](#) section of the Apache Maven documentation.

The following dependencies are listed in the BOM for the example `jee-example-app-1.0.0.ear` application

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.6</version>
    </dependency>
    <dependency>
      <groupId>commons-lang</groupId>
      <artifactId>commons-lang</artifactId>
      <version>2.5</version>
```

```
</dependency>
</dependencies>
</dependencyManagement>
```

Application POM Files

Each application module that can be mavenized has a separate directory containing its POM file. The directory name contains the name of the archive and ends in a **-jar**, **-war**, or **-ear** suffix, depending on the archive type.

Each application POM file lists that module's dependencies, including:

- Third-party libraries
- Java EE APIs
- Application submodules

For example, the POM file for the **jee-example-app-1.0.0.ear** EAR, **/path/to/output/mavenized/jee-example-app/jee-example-app-ear/pom.xml**, lists the following dependencies.

```
<dependencies>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.6</version>
  </dependency>
  <dependency>
    <groupId>org.jboss.seam</groupId>
    <artifactId>jee-example-web-war</artifactId>
    <version>1.0</version>
    <type>war</type>
  </dependency>
  <dependency>
    <groupId>org.jboss.seam</groupId>
    <artifactId>jee-example-services-jar</artifactId>
    <version>1.0</version>
  </dependency>
  <dependency>
    <groupId>org.jboss.seam</groupId>
    <artifactId>jee-example-services2-jar</artifactId>
    <version>1.0</version>
  </dependency>
</dependencies>
```

CHAPTER 6. OPTIMIZE RHAMT PERFORMANCE

RHAMT 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, RHAMT 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 RHAMT.

6.1. TIPS TO OPTIMIZE PERFORMANCE

6.1.1. Application and Command-line Suggestions

Try these suggestions first before upgrading hardware.

- If possible, execute RHAMT against the source code instead of the archives. This eliminates the need to decompile additional JARs and archives.
- Specify a comma-separated list of the packages to be evaluated by RHAMT using the `--packages` argument on the `RHAMT_HOME/bin/rhamt-cli` command line. If you omit this argument, RHAMT will decompile everything, which has a big impact on performance.
- Specify the `--excludeTags` argument where possible to exclude them from processing.
- Avoid decompiling and analyzing any unnecessary packages and files, such as proprietary packages or included dependencies. For additional information, see [Configure RHAMT to Exclude Files and Packages](#).
- Increase your ulimit when analyzing large applications. See [this Red Hat Knowledgebase article](#) for instructions on how to do this for Red Hat Enterprise Linux.
- If you have access to a server that has better resources than your laptop or desktop machine, you may want to consider running RHAMT on that server.

6.1.2. Hardware Upgrade Suggestions

If the application and command-line suggestions above do not improve performance, you may need to upgrade your hardware.

- If you have access to a server that has better resources than your laptop/desktop, then you may want to consider running RHAMT on that server.
- 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 quad-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.

6.2. CONFIGURE RHAMT TO EXCLUDE FILES AND PACKAGES

6.2.1. Exclude Packages

RHAMT can exclude packages during decompilation and analysis to increase performance. References to these packages are still found in the application's source code, but avoids the decompilation and analysis of proprietary classes.

Any packages that match the defined value will be excluded. For instance, to exclude `com.acme.example` and `com.acme.roadrunner`, you would only need to specify `com.acme`.

Packages may be excluded by using either of the following methods.

- Using the `--excludePackages` argument, as described in the [CLI Arguments](#).
- Specify the packages in a file contained within one of the [ignored locations](#). Each package should be included on a separate line, and the file must end in `.package-ignore.txt`. For example, see `RHAMT_HOME/ignore/proprietary.package-ignore.txt`.

6.2.2. Exclude Files

RHAMT can exclude specific files, such as included libraries or dependencies, during scanning and report generation. Files to be ignored are defined inside of a file ending in `.rhamt-ignore.txt` or `.windup-ignore.txt` within one of the [ignored locations](#).

These files contain a regex string detailing the name to exclude, with one file listed per line. For instance, to exclude the library `ant.jar` and any Java source files beginning with `Example`, the following file would be used:

```
.*ant.jar
.*Example.*\.java
```

6.2.3. Locations to Search for Exclusion

RHAMT searches the following locations:

- `~/.rhamt/ignore/`
- `~/.windup/ignore/`
- `RHAMT_HOME/ignore/`
- Any files and folders specified by the `--userIgnorePath` argument.

Each of these files should adhere to the rules specified in [Exclude Packages](#) or [Exclude Files](#), depending on the type of content to be excluded.

APPENDIX A. REFERENCE MATERIAL

A.1. RHAMT COMMAND-LINE ARGUMENTS

The following is a detailed description of the available RHAMT command line arguments.



NOTE


To run the RHAMT command without prompting, for example when executing from a script, use **--batchMode** to take the default values for unspecified parameters and **--overwrite** to force delete the output directory. Also be sure to specify the required **--input** and **--target** arguments.

See the description for each argument for more details.

Table A.1. RHAMT CLI Arguments

Argument	Description
<code>--additionalClassPath</code>	A space-delimited list of additional JAR files or directories to add to the class path so that they are available for decompilation or other analysis.
<code>--addonDir</code>	Add the specified directory as a custom add-on repository.
<code>--batchMode</code>	Flag to specify that RHAMT should be run in a non-interactive mode without prompting for confirmation. This mode takes the default values for any parameters not passed in to the command line.
<code>--debug</code>	Flag to run RHAMT in debug mode.
<code>--disableTattletale</code>	Flag to disable generation of the Tattletale report. If both enableTattletale and disableTattletale are set to true, then disableTattletale will be ignored and the Tattletale report will still be generated.
<code>--discoverPackages</code>	Flag to list all available packages in the input binary application.
<code>--enableClassNotFoundAnalysis</code>	Flag to enable analysis of Java files that are not available on the class path. This should not be used if some classes will be unavailable at analysis time.
<code>--enableCompatibleFilesReport</code>	Flag to enable generation of the Compatible Files report. Due to processing all files without found issues, this report may take a long time for large applications.

Argument	Description
<code>--enableTattletale</code>	Flag to enable generation of a Tattletale report for each application. This option is enabled by default when eap is in the included target. If both enableTattletale and disableTattletale are set to true, then disableTattletale will be ignored and the Tattletale report will still be generated.
<code>--excludePackages</code>	A space-delimited list of packages to exclude from evaluation. For example, entering com.mycompany.commonutilities would exclude all classes whose package name begins with com.mycompany.commonutilities .
<code>--excludeTags</code>	A space-delimited list of tags to exclude. When specified, rules with these tags will not be processed. To see the full list of tags, use the --listTags argument.
<code>--explodedApp</code>	Flag to indicate that the provided input directory contains source files for a single application. See the Input File Argument Tables for details.
<code>--exportCSV</code>	Flag to export the report data to a CSV file on your local file system. RHAMT creates the file in the directory specified by the --output argument. The CSV file can be imported into a spreadsheet program for data manipulation and analysis. For details, see Export the Report in CSV Format .
<code>--help</code>	Display the RHAMT help message.
<code>--immutableAddonDir</code>	Add the specified directory as a custom read-only add-on repository.
<code>--includeTags</code>	A space-delimited list of tags to use. When specified, only rules with these tags will be processed. To see the full list of tags, use the --listTags argument.
<code>--input</code>	A space-delimited list of the path to the file or directory containing one or more applications to be analyzed. This argument is required. See Specify the Input for more information.
<code>--install</code>	Specify add-ons to install. The syntax is GROUP_ID:ARTIFACT_ID[:VERSION] . For example, --install core-addon-x or --install org.example.addon:example:1.0.0 .

Argument	Description
--keepWorkDirs	Flag to instruct RHAMT to not delete temporary working files, such as the graph database and unzipped archives. This is useful for debugging purposes.
--list	Flag to list installed add-ons.
--listSourceTechnologies	Flag to list all available source technologies.
--listTags	Flag to list all available tags.
--listTargetTechnologies	Flag to list all available target technologies.
--mavenize	Flag to create a Maven project directory structure based on the structure and content of the application. This creates pom.xml files using the appropriate Java EE API and the correct dependencies between project modules. See also the --mavenizeGroupId option.
--mavenizeGroupId	When used with the --mavenize option, all generated pom.xml files will use the provided value for their <groupId> . If this argument is omitted, RHAMT will attempt to determine an appropriate <groupId> based on the application, or will default to com.mycompany.mavenized .
--online	Flag to allow network access for features that require it. Currently only validating XML schemas against external resources relies on Internet access. Note that this comes with a performance penalty.
--output	Specify the path to the directory to output the report information generated by RHAMT. See Specify the Output Directory for more information.
--overwrite	<p>Flag to force delete the existing output directory specified by --output. If you do not specify this argument and the --output directory exists, you are prompted to choose whether to overwrite the contents.</p> <div data-bbox="687 1722 1428 2040" style="background-color: #fff9c4; padding: 10px; border: 1px solid #ccc;"> <div style="display: flex; align-items: center;">  <div> <p>WARNING</p> <p>Be careful not to specify a report output directory that contains important information!</p> </div> </div> </div>

Argument	Description
<code>--packages</code>	A space-delimited list of the packages to be evaluated by RHAMT. It is highly recommended to use this argument. See Select Packages for more information.
<code>--remove</code>	Remove the specified add-ons. The syntax is <code>GROUP_ID:ARTIFACT_ID[:VERSION]</code> . For example, <code>--remove core-addon-x</code> or <code>--remove org.example.addon:example:1.0.0</code> .
<code>--skipReports</code>	Flag to indicate that HTML reports should not be generated. A common use of this argument is when exporting report data to a CSV file using <code>--exportCSV</code> .
<code>--source</code>	A space-delimited list of one or more source technologies, servers, platforms, or frameworks to migrate from. This argument, in conjunction with the <code>--target</code> argument, helps to determine which rulesets are used. Use the <code>--listSourceTechnologies</code> argument to list all available sources. See Set the Source Technology for more information.
<code>--sourceMode</code>	Flag to indicate that the application to be evaluated contains source files rather than compiled binaries. See the Input File Argument Tables for details.
<code>--target</code>	A space-delimited list of one or more target technologies, servers, platforms, or frameworks to migrate to. This argument, in conjunction with the <code>--source</code> argument, helps to determine which rulesets are used. Use the <code>--listTargetTechnologies</code> argument to list all available targets. See Set the Target Technology for more information.
<code>--userIgnorePath</code>	Specify a location, in addition to <code>`\${user.home}/.rhamt/ignore/`</code> , for RHAMT to identify files that should be ignored. For additional information on formatting these files, see Configure RHAMT to Exclude Files and Packages .
<code>--userRulesDirectory</code>	Specify a location, in addition to <code>RHAMT_HOME/rules/</code> and <code>`\${user.home}/.rhamt/rules/`</code> , for RHAMT to look for custom RHAMT rules. The value can be a directory containing ruleset files or a single ruleset file. The ruleset files must use the <code>.windup.xml</code> or <code>.rhamt.xml</code> suffix.
<code>--version</code>	Display the RHAMT version.

A.1.1. Specify the Input

A space-delimited list of the path to the file or directory containing one or more applications to be analyzed. This argument is required.

Usage

```
--input INPUT_ARCHIVE_OR_DIRECTORY [...]
```

Depending on whether the input file type provided to the `--input` argument is a file or directory, it will be evaluated as follows depending on the additional arguments provided.

Directory

<code>--explodedApp</code>	<code>--sourceMode</code>	Neither Argument
The directory is evaluated as a single application.	The directory is evaluated as a single application.	Each subdirectory is evaluated as an application.

File

<code>--explodedApp</code>	<code>--sourceMode</code>	Neither Argument
Argument is ignored; the file is evaluated as a single application.	The file is evaluated as a compressed project.	The file is evaluated as a single application.

A.1.2. Specify the Output Directory

Specify the path to the directory to output the report information generated by RHAMT.

Usage

```
--output OUTPUT_REPORT_DIRECTORY
```

- If omitted, the report will be generated in an `INPUT_ARCHIVE_OR_DIRECTORY.report` directory.
- If the output directory exists, you will be prompted with the following (with a default of N).

```
Overwrite all contents of "/home/username/OUTPUT_REPORT_DIRECTORY"
(anything already in the directory will be deleted)? [y,N]
```

However, if you specify the `--overwrite` argument, RHAMT will proceed to delete and recreate the directory. See the description of this argument for more information.

A.1.3. Set the Source Technology

A space-delimited list of one or more source technologies, servers, platforms, or frameworks to migrate from. This argument, in conjunction with the `--target` argument, helps to determine which rulesets are used. Use the `--listSourceTechnologies` argument to list all available sources.

Usage

```
--source SOURCE_1 SOURCE_2
```

The **--source** argument now provides version support, which follows the [Maven version range syntax](#). This instructs RHAMT to only run the rulesets matching the specified versions. For example, **--source eap:5**.



WARNING

When migrating to JBoss EAP, be sure to specify the version, for example, **eap:6**. Specifying only **eap** will run rulesets for all versions of JBoss EAP, including those not relevant to your migration path.

See [Supported Migration Paths](#) in the RHAMT *Getting Started Guide* for which JBoss EAP version is appropriate for your source platform.

A.1.4. Set the Target Technology

A space-delimited list of one or more target technologies, servers, platforms, or frameworks to migrate to. This argument, in conjunction with the **--source** argument, helps to determine which rulesets are used. If you do not specify this option, you are prompted to select a target. Use the **--listTargetTechnologies** argument to list all available targets.

Usage

```
--target TARGET_1 TARGET_2
```

The **--target** argument now provides version support, which follows the [Maven version range syntax](#). This instructs RHAMT to only run the rulesets matching the specified versions. For example, **--target eap:7**.



WARNING

When migrating to JBoss EAP, be sure to specify the version in the target, for example, **eap:6**. Specifying only **eap** will run rulesets for all versions of JBoss EAP, including those not relevant to your migration path.

See [Supported Migration Paths](#) in the RHAMT *Getting Started Guide* for which JBoss EAP version is appropriate for your source platform.

A.1.5. Select Packages

A space-delimited list of the packages to be evaluated by RHAMT. It is highly recommended to use this argument.

Usage

```
--packages PACKAGE_1 PACKAGE_2 PACKAGE_N
```

- In most cases, you are interested only in evaluating custom application class packages and not standard Java EE or third party packages. The **PACKAGE_N** argument is a package prefix; all subpackages will be scanned. For example, to scan the packages **com.mycustomapp** and **com.myotherapp**, use **--packages com.mycustomapp com.myotherapp** argument on the command line.
- While you can provide package names for standard Java EE third party software like **org.apache**, it is usually best not to include them as they should not impact the migration effort.



WARNING

If you omit the **--packages** argument, every package in the application is scanned, which can impact performance. It is best to provide this argument with one or more packages. For additional tips on how to improve performance, see [Optimize RHAMT Performance](#).

A.2. RULE STORY POINTS

A.2.1. What are Story Points?

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

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

A.2.2. How Story Points are Estimated in Rules

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

Level of Effort	Story Points	Description
Information	0	An informational warning with very low or no priority for migration.
Trivial	1	The migration is a trivial change or a simple library swap with no or minimal API changes.
Complex	3	The changes required for the migration task are complex, but have a documented solution.

Level of Effort	Story Points	Description
Redesign	5	The migration task requires a redesign or a complete library change, with significant API changes.
Rearchitecture	7	The migration requires a complete rearchitecture of the component or subsystem.
Unknown	13	The migration solution is not known and may need a complete rewrite.

A.2.3. Task Category

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

Mandatory

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

Optional

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

Potential

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

Information

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

For more information on categorizing tasks, see [Using Custom Rule Categories](#) in the *Rules Development Guide*.

A.3. ADDITIONAL RESOURCES

A.3.1. Get Involved

To help make Red Hat Application Migration Toolkit cover most application constructs and server configurations, including yours, you can help with any of the following items.

- Send an email to jboss-migration-feedback@redhat.com and let us know what RHAMT migration rules should cover.
- Provide example applications to test migration rules.
- Identify application components and problem areas that may be difficult to migrate.

- Write a short description of these problem migration areas.
- Write a brief overview describing how to solve the problem migration areas.
- Try Red Hat Application Migration Toolkit on your application. Be sure to [report any issues](#) you encounter.
- Contribute to the Red Hat Application Migration Toolkit rules repository.
 - Write a Red Hat Application Migration Toolkit rule to identify or automate a migration process.
 - Create a test for the new rule.
 - Details are provided in the [Rules Development Guide](#).
- Contribute to the project source code.
 - Create a core rule.
 - Improve RHAMT performance or efficiency.
 - See the [Core Development Guide](#) for information about how to configure your environment and set up the project.

Any level of involvement is greatly appreciated!

A.3.2. Important Links

- RHAMT forums: <https://developer.jboss.org/en/windup>
- RHAMT JIRA issue trackers
 - Core RHAMT: <https://issues.jboss.org/browse/WINDUP>
 - RHAMT Rules: <https://issues.jboss.org/browse/WINDUPRULE>
- RHAMT mailing list: jboss-migration-feedback@redhat.com
- RHAMT on Twitter: [@JBossWindup](#)
- RHAMT IRC channel: Server FreeNode ([irc.freenode.net](#)), channel [#windup](#) ([transcripts](#)).

A.3.3. Known RHAMT Issues

You can review known issues for RHAMT here: [Open RHAMT issues](#).

A.3.4. Report Issues with RHAMT

Red Hat Application Migration Toolkit uses JIRA as its issue tracking system. If you encounter an issue executing RHAMT, please file a JIRA Issue.



NOTE

If you do not have one already, you must sign up for a JIRA account in order to create a JIRA issue.

A.3.4.1. Create a JIRA Issue

1. Open a browser and navigate to the JIRA [Create Issue](#) page.
If you have not yet logged in, click the **Log In** link at the top right side of the page and enter your credentials.
2. Choose the following options and click the **Next** button.
 - **Project**
For core RHAMT issues, choose *Red Hat Application Migration Toolkit (WINDUP)*.

For issues with RHAMT rules, choose: *Red Hat Application Migration Toolkit rules (WINDUPRULE)*.
 - **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.
 - **Attachment:** If the application or archive causing the issue does not contain sensitive information and you are comfortable sharing it with the RHAMT development team, attach it to the issue using the **browse** button.
4. Click the **Create** button to create the JIRA issue.

Revised on 2019-03-26 20:05:27 UTC