

Kuali Rice 2.1.1-SNAPSHOT Installation Guide

Table of Contents

| | |
|--|----|
| 1. Overview | 1 |
| Suggested Operating Systems | 1 |
| Software Distributions | 1 |
| Obtaining the Software | 1 |
| Installation Steps | 2 |
| 2. Standalone Server Setup | 3 |
| Suggested Server Hardware | 3 |
| Installing and Configuring the Database Management System | 4 |
| Locations for Database Software | 4 |
| MySQL Database Preparation | 5 |
| Oracle Database Preparation | 6 |
| Suggested SQL Client Software | 8 |
| Install and Configure Required Software | 9 |
| Overview | 9 |
| Environment Variables | 9 |
| Java SDK | 10 |
| Maven | 11 |
| Install and Setup Apache Tomcat | 12 |
| Install Rice software from distribution | 12 |
| Building the Rice Database | 12 |
| Load Impex Data with Maven | 12 |
| Verifying your Database Installation | 13 |
| Configure Rice | 14 |
| Deploying the WAR file | 14 |
| Parameters | 17 |
| Setup a Keystore | 19 |
| Generating a Keystore | 19 |
| Configure KSB to use the keystore | 19 |
| 3. Tuning Quali Rice 2.1.1-SNAPSHOT | 20 |
| JVM Tuning | 20 |
| 4. Additional Configurations | 21 |
| Setting Up a Load-Balanced Clustered Production Environment | 21 |
| Running Multiple Instances of Rice Within a Single Tomcat Instance | 22 |
| Running a Staging and a Test Environment | 22 |
| Running Multiple Production Environments | 24 |
| Keystore Implementation Variations | 31 |
| 5. Creating a Client Application | 32 |
| Development Tools | 32 |
| IDE (Integrated Development Environment) | 32 |
| Database | 32 |
| JDK | 33 |
| Maven | 34 |
| Servlet Container | 34 |
| New Project Setup | 35 |
| Project Structure and Configuration Files | 36 |
| Configuring Your Rice Application | 38 |
| Importing into Eclipse | 38 |
| Appendix A. Example Server Configurations | 41 |
| Single Server Configuration | 41 |
| Multi Server Configuration | 41 |
| Web Servers | 41 |

Kuali Rice 2.1.1-
SNAPSHOT Installation Guide

| | |
|--|----|
| Tomcat Servers | 41 |
| Web Servers – Content/Shared File System | 42 |
| Glossary | 43 |

List of Figures

- 2.1. Oracle XE admin webapp 7
- 2.2. Rice Portal Main Menu 16
- 5.1. Eclipse Import 39
- 5.2. Eclipse Import Project Directory 40

List of Tables

| | |
|---|----|
| 1.1. Rice Software Distribution Types | 1 |
| 2.1. Locations for Database Software | 4 |
| 2.2. Core | 17 |
| 2.3. Database | 17 |
| 2.4. KSB | 18 |
| 2.5. KEN | 18 |
| 2.6. KEW | 18 |
| 5.1. Locations for Database Software | 33 |
| 5.2. Generated Files | 37 |
| 5.3. Configuration Parameters | 38 |

Chapter 1. Overview

Kuali Rice has the potential to run on most platforms that support a Java development environment (not simply a runtime environment), a servlet container, and an Oracle or MySQL relational database management system (RDBMS).

Note

Only platforms and configurations that have been tested and are known to work with Rice are described within this guide. Other platforms and configurations may work, but have not been tested. Please share any configurations that you have gotten to work with us by joining our [collaboration list](#).

Suggested Operating Systems

Since Kuali Rice is written in Java, it should in theory be able to run on any operating system that supports the required version of the Java runtime. However, it has been most actively tested on:

- Windows (XP, Vista, and 7)
- Mac OS X (10.6 and 10.7)
- Linux (Ubuntu)

Note that while Ubuntu Linux is the distribution most frequently used for testing, other Linux distributions such as Fedora, Red Hat Enterprise Linux, CentOS, Gentoo, and others should also be able to run Kuali Rice.

Additionally, Kuali Rice will likely work on other Unix operating systems such as Sun Microsystems Solaris and IBM AIX, although the software has not been tested here.

Software Distributions

The Kuali Rice software is available through three different distributions:

Table 1.1. Rice Software Distribution Types

| Distribution | Description |
|--------------|--|
| Binary | This distribution consists of all the necessary binaries, supporting files and database schemas and data for running Kuali Rice as a web application or within an embedded client application. |
| Source | The source code and build scripts necessary for compiling and building Rice, a process described in the appendices. |
| Server | Rice in the form of a web application archive (WAR) along with database schemas and data. |

Obtaining the Software

1. **Download:** The Rice software can be downloaded from <http://kuali.org/rice/download>
2. **Maven Repository** - <http://nexus.kuali.org/content/groups/public/>

3. **Subversion Repository** - <http://svn.kuali.org/repos/rice/tags/rice-2.1.1-SNAPSHOT>

Installation Steps

All Kuali Rice 2.1.1-SNAPSHOT installations follow the same core steps:

1. Install a database and JDBC drivers.
2. Install and configure a JDK, Rice, and other required software.
3. Set up ImpEx process to create the database schema and populate it.
4. Configure Rice software.
5. Test the installation

This Guide will provide installation instructions for the Rice standalone server as well as instructions on how to set up Eclipse to create a client application with Rice.

Chapter 2. Standalone Server Setup

This chapter is designed to provide simple step-by-step instructions on how to set up a Kualu Rice standalone server intended for enterprise deployment. The same steps can be used to set up a standalone server locally for development purposes, you would just install the database on the same machine.

The steps to install and setup a standalone server with Kualu Rice are:

1. Determine your expected load and storage needs and consult the Suggested Server Hardware section for guidance. Install OS.
2. Install & configure the database management system.
3. Install & configure required software
4. Install and configure Tomcat.
5. Install and configure Rice.
6. Launch the sample application.
7. Set up a Keystore.

Note

Rather than install and setup as the root user on systems designed for production, you may want to create a non-privileged user named something like 'rice' to use for this purpose.

Suggested Server Hardware

Note that hardware needs may vary depending on the amount of expected load, the operating system being used, and the number of applications that are integrated with Kualu Rice. Kualu Rice is typically deployed as a standalone server with the database server separate from the application server.

The recommended minimum requirements are as follows:

- Processor 1.5 GHz or faster (2 GHz preferred)
- 1024 MB (1 GB) of RAM or more
- 100 Mbit/s network card (gigabit preferred)
- 200 MB of hard disk space (for Tomcat server and web application)

Note

Additional space needed if storing attachments.

See [Example Server Configurations](#) in the appendix for examples of hardware and software configurations of Kualu Rice servers.

Installing and Configuring the Database Management System

Kuali Rice was developed using two relational database management systems: **MySQL** and **Oracle**. The typical production install involves running the Rice server separate from the database server, however both can be run on the same machine for development purposes.

Rice runs, and has been tested with the following versions:

- **Oracle**
 - Oracle Database 10g
 - Oracle Database 11g
 - Oracle Express Edition (XE)

Use the Oracle JDBC Driver to connect to these databases.

Ensure that the Oracle database you intend to use encodes character data in a UTF variant by default. For Oracle XE, this entails downloading the "Universal" flavor of the binary, which uses AL32UTF8.

- **MySQL**
 - MySQL 5.1.+

Use the MySQL Connector/J (5.1.+) to connect to MySQL databases.

You should be able to adapt Rice to other standard relational databases (e.g., Sybase, Microsoft SQL Server, DB2, etc.). However, this Installation Guide does not provide information for running Rice with these products.

Locations for Database Software

Below are locations from which Oracle and MySQL could be downloaded at the time of release of Rice 2.1.1-SNAPSHOT.

Table 2.1. Locations for Database Software

| Software | Download Location |
|---|---|
| Oracle Standard and Enterprise Editions | http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html |
| Oracle Express Edition | http://www.oracle.com/technetwork/database/express-edition/downloads/index.html |
| Oracle JDBC DB Driver | http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html |
| MySQL | http://www.mysql.com/downloads/ |
| MySQL Connector/J JDBC Driver | http://www.mysql.com/products/connector/j/ |

MySQL Database Preparation

Kuali Rice supports both MySQL and Oracle databases. However, MySQL is easier to install than Oracle and uses less machine resources so many developers prefer to use that when getting started with Rice.

Installation

The installation steps for MySQL are going to be different for each platform. Please download the latest version MySQL Server from the location listed in the [Locations for Database Software](#) section of this document and follow the installation instructions for your platform.

Note

You may be required to create an account on the MySQL site in order to download the software.

Please be sure to follow the instructions for installing MySQL on your platform very carefully. If downloading for Mac OS X, **be careful** to download the appropriate version for your platform (32-bit vs. 64-bit and 10.6 vs 10.7)

Configuration

There are a few MySQL database configuration options that are required in order for Kuali Rice to work properly. These will need to be set in either your **my.cnf** or **my.ini** file. The location and names of these files will differ depending on which platform you are working on. For details on where these files can be found, see the following document:

<http://dev.mysql.com/doc/refman/5.1/en/option-files.html>

Once you have located this file, please add the following options, paying special attention to the line that needs to be commented out:

```
[mysqld]
max_allowed_packet=20M
transaction-isolation=READ-COMMITTED
lower_case_table_names=1
max_connections=1000
innodb_locks_unsafe_for_binlog=1
...
# Be sure to comment this out if it's in the file!!!
#log-bin=mysql-bin
```

Note that the **[mysqld]** section may already be in your my.cnf file. If so, you can just add the options listed above underneath that section

Caution

It is very important that you comment out **log-bin**. Otherwise you will end up with some very bad problems later!

Verification

Before verifying your mysql installation you will need to ensure that MySQL is running. Some of the platform-specific packages will set this up automatically (or allow you to install yourself in the case of

Mac OS X). If MySQL is not starting automatically you can start it using a command like the following example from Mac OS X:

```
sudo /usr/local/mysql/bin/mysqld_safe
```

This will start the MySQL server.

To verify that you can actually connect to the server, execute the following at the command line:

```
mysql -u root -p
```

This should bring you to a command line client interface for the MySQL server. Type "show databases;" and press return. You should see output similar to the following.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.50-log MySQL Community Server (GPL)
```

```
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show databases;
+-----+
| Database                |
+-----+
| information_schema      |
| mysql                   |
| test                     |
+-----+
3 rows in set (0.13 sec)

mysql>
```

Install the JDBC Driver

Kuali Rice uses the MySQL Connector/J product as the native JDBC driver. Please download this driver from the location specified in the [Locations for Database Software](#) section of this Guide.

Once you have downloaded the JDBC driver that corresponds to your version of MySQL, copy it to **/java/drivers**. **/java/drivers** is a hard coded directory that the Rice scripts use as a default directory in which to search for drivers when the installation scripts are running.

Oracle Database Preparation

Installation

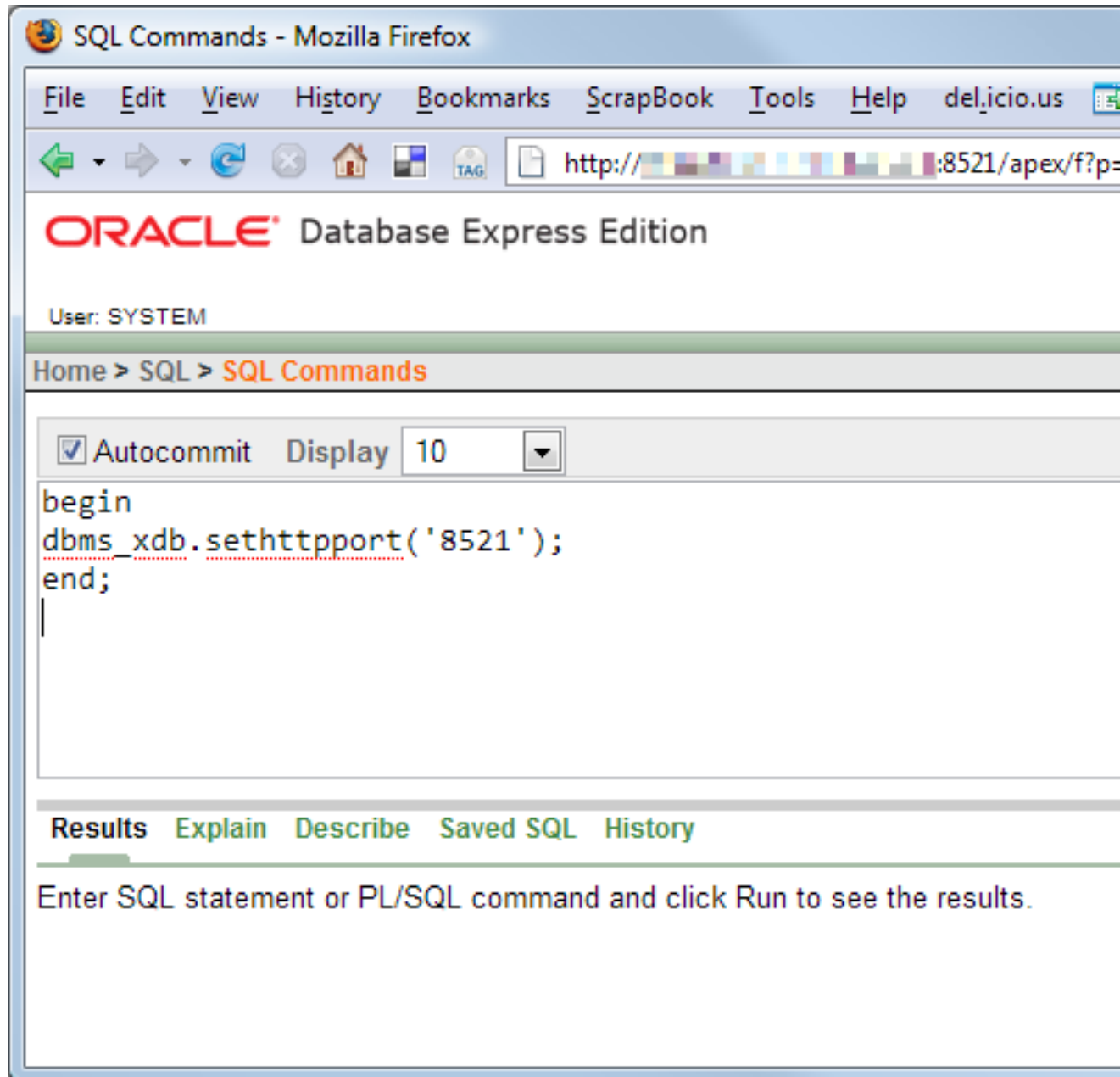
The installation steps for Oracle are going to be different for each platform and version of Oracle. Please download from the location listed in the [Locations for Database Software](#) section of this document and follow the installation instructions for your platform.

To run the database completely on your local machine, we recommend installing Oracle Express (XE). Please refer to the [Locations for Database Software](#) section of this Installation Guide to find the download location for this software.

Configuration

By default, OracleXE registers a web user interface on port 8080. This is the same port that the standalone version of Rice is preconfigured to use. To avoid a port conflict, you must change the port that the OracleXE web user interface uses with the Oracle XE admin webapp:

Figure 2.1. Oracle XE admin webapp



If you prefer, you can use the Oracle SQL tool described here to change the OracleXE web user interface port: <http://daust.blogspot.com/2006/01/xe-changing-default-http-port.html>

Please edit your hosts file with an entry to refer to your Oracle database. When this Installation Guide refers to the Oracle database host server, it will be referred to in the examples as koracle.

Now edit the hosts file and add this:

```
<ip address of mysql server> koracle
```

Verification

To connect to the supporting Oracle database (i.e., run scripts, view database tables, etc.), we recommend installing the Squirrel SQL client. Please see the [section on Squirrel SQL](#) for more information.

Install the JDBC Driver

Kuali Rice uses the standard Oracle JDBC driver as the native JDBC driver.

1. Please download this driver from the location specified in the [Locations for Database Software](#) section of this Guide.
2. Once you have downloaded the JDBC driver that corresponds to your version of Oracle, copy it to **/java/drivers**. **/java/drivers** is a hard coded directory that the Rice scripts use as a default directory in which to search for drivers when the installation scripts are running.
3. Use the [maven-install-plugin](#) to copy ojdbc14.jar into your local maven repository.

```
mvn install:install-file -DgroupId=com.oracle -DartifactId=ojdbc14 -Dversion=10.
```

4. You should see output similar to this if the jar gets installed correctly.

```
[INFO] Installing ojdbc14.jar to /.m2/repository/com/oracle/ojdbc14/10.2.0.3.0/o
```

Suggested SQL Client Software

To examine and test your database setup, SQL client software is useful. Any SQL client software that will connect to a MySQL or Oracle database will work. Two tools used by the development team are the **mysql** command-line client and Squirrel SQL.

mysql client software

The **mysql** command-line client only works with MySQL and is usually installed with the MySQL Server software. An example of connection to MySQL as root and then switching to a database named **test** can be found below:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.50-log MySQL Community Server (GPL)
```

```
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> use test;
Database changed
```

Squirrel SQL

Tools like Squirrel SQL use JDBC to access the database and will work with both MySQL and Oracle databases. You can download and install it from the following URL:

- <http://squirrel-sql.sourceforge.net/>

With MySQL

Connecting to a MySQL database with the name **test** would have a JDBC url like the following:
jdbc:mysql://localhost:3306/test.

With Oracle

Connecting to an Oracle database would have a JDBC url like the following:
jdbc:oracle:thin:@localhost:1521:XE.

The Rice SQL files use slash '/' as the statement delimiter. You may have to configure your SQL client appropriately so it can run the Rice SQL. In SquirrelL, you do this in Session->Session Properties->SQL->Statement Separator.

Install and Configure Required Software

Overview

Kuali Rice requires the following software to be setup and configured:

- Sun Microsystems Java Development Kit (JDK 1.6.x)

Warning

You must use a JDK and not a Java runtime environment (JRE); the JDK you use must be version 1.6.x. Additionally, Rice has not been tested on JDKs other than Sun. So alternative implementations like OpenJDK should be used at your own risk.

- Maven 3

Environment Variables

First, some environment variables need to be configured.

Mac OS X

Environment variables in Mac OS X can be set in a number of ways, but here we will show how to modify or create the .profile files in your user home directory. On OS X your user home directory is typically located at /Users/<username>.

An example .profile can be found below:

```
M2_HOME=/usr/local/maven
MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=768m"
GROOVY_HOME=/usr/local/groovy
JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Versions/1.6/Home
CATALINA_HOME=/usr/local/tomcat
MYSQL_HOME=/usr/local/mysql

PATH="$ANT_HOME/bin:$M2_HOME/bin:$GROOVY_HOME/bin:$CATALINA_HOME:$MYSQL_HOME/bin:$
export PATH ANT_HOME ANT_OPTS M2_HOME MAVEN_OPTS GROOVY_HOME JAVA_HOME CATALINA_HO
```

Note

It is important to export your environment variables once they are defined as the file above does.

Windows XP

To get to the screen where you can define environment variables on Windows XP follow these steps:

1. Click on the "Start" button in the bottom left-hand corner.
2. On the resulting screen, right click on "My Computer".
3. In the context menu, click on "Properties".
4. This will open up the "System Properties" dialog window.
5. Click on the "Advanced" tab.
6. Click on the "Environment Variables" button.
7. You will see the screen where you can edit existing environment variables or define new ones.

Windows Vista and Windows 7

To get to the screen where you can define environment variables on Windows Vista or Windows 7 follow these steps:

1. Click on the "Start" button in the bottom left-hand corner.
2. On the resulting screen, right click on "Computer".
3. In the context menu, click on "Properties".
4. This will open up the Control Panel "System" dialog.
5. Click on the "Advanced system settings".
6. In the resulting window, click on the "Environment Variables..." button.
7. You will see the screen where you can edit existing environment variables or define new ones.

Note

The windows command line console must be closed and reopened in order for environment variable changes to be effective.

Java SDK

Installation

You should download and install the latest version of JDK 6. If you are on Windows, you can download it from the following URL: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

If you are on a Mac, then Java 6 should already be installed if you are up to date with the latest updates from Apple.

Configuration

You will also want to set up your **JAVA_HOME** environment variable to point to the installation directory of your JDK. In both Windows and Mac environments, the **java** executable program should already be on your path. But if it is not, you will want to include **JAVA_HOME/bin** in your **PATH** environment variable.

If you do not know how to do this, see the **Environment Variables** section above for your platform.

Verification

In order to verify that your JDK has been installed successfully, open a command prompt and type the following:

```
java -version
```

You should see output similar to the following:

```
java version "1.6.0_26"  
Java(TM) SE Runtime Environment (build 1.6.0_26-b03-383-11A511)  
Java HotSpot(TM) 64-Bit Server VM (build 20.1-b02-383, mixed mode)
```

If you receive an error indicating that the "java" command could not be found, please ensure that the java command is on your machine's **PATH** environment variable.

To prevent potential out of memory errors when running Rice, you should set your **JAVA_OPTS** environment to a value like the following:

```
JAVA_OPTS="-Xmx1024m -XX:MaxPermSize=768m"
```

If you do not know how to do this, see the [Environment Variables](#) section above for your platform.

Maven

Maven is the primary build tool used by the Kualu Rice project. Maven is based on a project object model (POM) that defines various standards and conventions surrounding the organization of a project. This facilitates a set of standard build goals and lifecycle phases (such as compile, test, package, etc.)

Installation

To download version 3 of Maven, use the following link: <http://maven.apache.org/download.html>

Once you have downloaded the zip file, unzip it to a location of your choosing.

Configuration

You will want to set your **M2_HOME** environment variable to point to the location where you unzipped Maven. You will additionally want to include **M2_HOME/bin** in your **PATH** environment variable so that maven can be executed from the command line without having to specify the full path.

Finally, to prevent potential out of memory errors when compiling Rice with Maven, you should set your **MAVEN_OPTS** environment to a value like the following:

```
MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=768m"
```

If you do not know how to do this, see the [Environment Variables](#) section above for your platform.

Verification

In order to verify that Maven has been installed successfully and is available on the path, open a command prompt and type the following:

```
mvn -version
```

You should see output like the following:

```
Apache Maven 3.0.3 (r1075438; 2011-02-28 10:31:09-0700)
Maven home: /usr/local/maven
Java version: 1.6.0_26, vendor: Apple Inc.
Java home: /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home
Default locale: en_US, platform encoding: MacRoman
OS name: "mac os x", version: "10.7.2", arch: "x86_64", family: "mac"
```

If you receive an error indicating that the "mvn" command could not be found, please ensure that the directory that includes the mvn executable (**M2_HOME/bin**) is on your machine's **PATH** environment variable.

Install and Setup Apache Tomcat

Kuali Rice 2.1.1-SNAPSHOT supports the following Tomcat versions:

- Tomcat 6 (Servlet API 2.5, JSP 2.1)
- Tomcat 7 (Servlet API 3.0, JSP 2.2)

Please visit the [Apache Tomcat site](#) for information on how to install and configure Tomcat.

Other servlet containers can be used with Kuali Rice, but this guide will focus on Tomcat.

Install Rice software from distribution

The quickest way to get Rice installed on a standalone server is to download the **Standalone Server** distribution from the [kuali.org download site](#). Once downloaded, decompress the software.

Building the Rice Database

Load Impex Data with Maven

1. Change to the directory **db/demo** located where you decompressed the software.
2. Verify that Maven can connect to your database instance.

Running Locally:

- MySQL

```
mvn validate -Pdb,mysql -Dimpex.dba.password=[dbpassword]
```

Note

If the 'root' user does not have a password (which is the default on a new MySQL install), drop the '-Dimpex.dba.password' parameter from all MySQL commands or use 'NONE' for the password:

```
mvn validate -Pdb,mysql
```

or

```
mvn validate -Pdb,mysql -Dimpex.dba.password=NONE
```

- Oracle

```
mvn validate -Pdb,oracle -Dimpex.dba.password=[dbapassword]
```

Connecting to a remote database:

- MySQL

```
mvn validate -Pdb,mysql -Dimpex.dba.url=jdbc:mysql://[your-mysql-instance]/ -D
```

- Oracle

```
mvn validate -Pdb,oracle -Dimpex.url=jdbc:oracle:thin:@[your-oracle-server]:15
```

3. Load the data set.

Running Locally:

- MySQL

```
mvn clean install -Pdb,mysql -Dimpex.dba.password=[dbapassword]
```

- Oracle

```
mvn clean install -Pdb,oracle -Dimpex.dba.password=[dbapassword]
```

Connecting to a remote database:

- MySQL

```
mvn clean install -Pdb,mysql -Dimpex.dba.url=jdbc:mysql://[your-mysql-instance]
```

- Oracle

```
mvn clean install -Pdb,oracle -Dimpex.url=jdbc:oracle:thin:@[your-oracle-server]
```

4. Wait for the maven process to build your database (it may take a little while, especially if this is the first time you've done it and maven has to download impex and other plugins).

5. You should get the following message at the end of the process.

```
[ INFO ] -----  
[ INFO ] BUILD SUCCESS  
[ INFO ] -----
```

Verifying your Database Installation

At this point, your Kualu Rice database should be successfully installed. To verify this, log into your database and verify the number of tables that are present. There should be at least 200 (the number will be different for mysql and oracle).

Configure Rice

Deploying the WAR file

1. Copy the kr-dev.war file from the base directory of the server distribution to the directory that contains web applications in your servlet container. For Tomcat, this is **[Tomcat-root-directory]/webapps**.
2. Copy the database-specific JDBC driver to the **[Tomcat-root-directory]/lib**. Examples:

- MySQL

```
cp -p /java/drivers/mysql-connector-java-5.1.5-bin.jar /usr/local/tomcat/lib
```

- Oracle

```
cp -p /java/drivers/ojdbc14.jar /usr/local/tomcat/lib
```

3. Configure the rice-config.xml File. By default when it starts, Rice attempts to read the **rice-config.xml** configuration file from the paths in this order:
 - a. /usr/local/rice/rice-config.xml
 - b. \${rice.base}../../conf/rice-config.xml
 - c. \${rice.base}../conf/rice-config.xml
 - d. \${additional.config.locations}

The value for **rice.base** is calculated using different locations until a valid location is found. Kualif calculates it using these locations in this sequence:

- a. ServletContext.getRealPath("/")
- b. **catalina.base** system property
- c. The current working directory

On Windows it also checks the following location:

- a. %USERPROFILE%\kuali\main\dev

An example **rice-config.xml** file is included in the server distribution under **config/web/src/main/config/example-config**.

For MySQL, just modify the following database parameters in the **rice-config.xml** file. The values should conform to the values you will use with the ImpEx tool via the **impex-build.properties** file. ImpEx is set up in the [Installing and Configuring the Database Management System](#) section of this guide.

```
datasource.url=jdbc:mysql://localhost:3306/rice
datasource.username=rice
datasource.password=kualirice
datasource.url=jdbc:mysql://remoteMySQLServerComputerName:3306/rice
datasource.username=rice
datasource.password=kualirice
```

If you are using Oracle, the JDBC URL will have this general form:

```
datasource.url=jdbc:oracle:thin:@remoteMySQLServerComputerName:1521:ORACLE_SID
```

Note

Additional configuration parameters are described in the [Parameters](#) section of this guide.

4. At this point, you are ready to try to bring up the Tomcat server with the Rice web application:

```
cd /usr/local/tomcat/bin
./startup.sh
```

5. Check if Tomcat and Rice started successfully:

```
cd /usr/local/tomcat/logs
tail -n 500 -f catalina.out
```

6. If your Rice server started up successfully, browse to the site **http://yourlocalip:8080/kr-dev**. You should see the Rice portal screen which will look something like this:

Figure 2.2. Rice Portal Main Menu

Parameters

The tables below have the basic set of parameters for **rice-config.xml** that you need to get an instance of Rice running. Please use these tables as a beginning reference to modify your **rice-config.xml** file.

Warning

Make sure the **application.url** and database user name and password are set correctly.

Table 2.2. Core

| Parameter | Description | Examples or Values |
|-------------------------------|--|---|
| application.url | The external URL used to access the Rice web interface; edit only the fully-qualified domain name and port of the server | http://yourlocalip:8080/kuali-rice-url |
| app.context.name | Context name of the web application <ul style="list-style-type: none"> Essentially, the name of the WAR file Used to build the path for images and help URLs | kuali-rice-url (This value should not be changed) |
| log4j.settings.path | Path to log4j.properties file. If the file does not exist, you must create it. | /usr/local/ice/log4j.properties |
| log4j.settings.reloadInterval | interval (in minutes) to check for changes to the log4j.properties file | 5 |
| mail.smtp.host | SMTP host name or IP (This param is not in the default config.) | localhost |
| config.location | Location of configuration file to load environment-specific configuration parameters (This param is not in the default config.) | /usr/local/ice/ice-config-\${environment}.xml |
| sample.enabled | Enable the sample application | boolean |

Table 2.3. Database

| Parameter | Description | Examples or Values |
|-------------------------|---|--|
| datasource.obj.platform | Name of OJB platform to use for the database | Oracle9i or MySQL |
| datasource.platform | Rice platform implementation for the database | <ul style="list-style-type: none"> org.kuali.rice.core.database.platform.DerbyPlatform org.kuali.rice.core.database.platform.OraclePlatform org.kuali.rice.core.database.platform.MySQLPlatform |
| datasource.driver.name | JDBC driver for the database | <ul style="list-style-type: none"> org.apache.derby.jdbc.EmbeddedDriver, oracle.jdbc.driver.OracleDriver com.mysql.jdbc.Driver |
| datasource.username | User name for connecting to the server database | rice |

| Parameter | Description | Examples or Values |
|---------------------------------|---|--|
| datasource.password | Password for connecting to the server database | |
| datasource.url | JDBC URL of database to connect to | <ul style="list-style-type: none"> • jdbc:oracle:thin:@localhost:1521:XE • jdbc:mysql://localhost:3306/kuldemo |
| datasource.pool.minSize | Minimum number of connections to hold in the pool | an integer value suitable for your environment |
| datasource.pool.maxSize | Maximum number of connections to allocate in the pool | an integer value suitable for your environment |
| datasource.pool.maxWait | Maximum amount of time (in ms) to wait for a connection from the pool | 10000 |
| datasource.pool.validateOnQuery | Query to validate connections from the database | select 1 from dual |

Table 2.4. KSB

| Parameter | Description | Examples or Values |
|-------------------|--|--|
| serviceServletUrl | URL that maps to the KSBDispatcherServlet (include a trailing slash); This param is not in the default config. | |
| keystore.file | Path to the keystore file to use for security | /usr/local/rice/rice.keystore |
| keystore.alias | Alias of the standalone server's key | see section entitled Generating the Keystore |
| keystore.password | Password to access the keystore and the server's key | see section entitled Generating the Keystore |

Table 2.5. KEN

| Parameter | Description | Examples or Values |
|-------------------------|--|--------------------|
| notification.basewebapp | Base URL of the KEN web application (This param is not in the default config.) | |

Table 2.6. KEW

| Parameter | Description | Examples or Values |
|-------------------------|---|-------------------------|
| workflow.url | URL to the KEW web module | \${application.url}/kew |
| plugin.dir | Directory from which plugins will be loaded | /usr/local/rice/plugins |
| attachment.dir.location | Directory where attachments will be stored (This param is not in the default config.) | |

Setup a Keystore

For client applications to consume secured services hosted from a Rice server, you must generate a keystore. As an initial setup, you can use the keystore provided by Rice. Once a keystore is generated, you must configure the KSB to use the keystore.

Generating a Keystore

There are three ways to get this keystore:

1. If you are doing a source code build of Rice, it is in the directory `<source root>/security` and it has a file name of `rice.keystore`
2. The keystore is also located in the server distribution under the security directory.
3. You can generate the keystore yourself. Please refer to the [Security and Keystores](#) section in the [KSB chapter](#) of the [Technical Reference Guide](#) for the steps to accomplish this.

Configure KSB to use the keystore

You must have these params in the xml config to allow KSB to use the keystore:

```
<param name="keystore.file">/usr/local/rice/rice.keystore</param>
<param name="keystore.alias">rice</param>
<param name="keystore.password">r1c3pw</param>
```

- `keystore.file` - The location of the keystore
- `keystore.alias` - The alias used in creating the keystore above
- `keystore.password` - This is the password of the alias AND the keystore. This assumes that the keystore is set up so that these are the same.

Chapter 3. Tuning Kuali Rice 2.1.1-SNAPSHOT

Performance tuning is an art form in and of itself, and tuning Kuali Rice is no exception. Here are some items we've found that may help with your tuning issues. Additionally, we are collecting performance tuning information in the Kuali Rice wiki at <https://wiki.kuali.org/x/2hOeEg>.

JVM Tuning

To avoid OutOfMemoryError errors, tune the JVM by increasing the allocated memory.

Add these lines to the catalina.sh file in the tomcat/bin directory:

```
JAVA_OPTS="-Xmx=512m -XX:MaxPermSize=256m"
```

Chapter 4. Additional Configurations

There is a number of additional configurations that Rice supports. We've collected a few of them here and provided setup instructions below.

See [Example Server Configurations](#) in the appendix for examples of hardware and software configurations of Kualu Rice servers.

Setting Up a Load-Balanced Clustered Production Environment

This describes how to set up Rice instances for a load-balanced production environment across multiple servers.

1. The configuration parameter `environment` must be set to the text: **prd**
2. When the configuration parameter `environment` is set to **prd**, the code triggers:
 - a. Sending email to specified individuals
 - b. Turning off some of the Rice “back doors”

The high-level process for creating multiple Rice instances:

1. Ensure that these are set up properly so no additional configuration is needed during installation:
 - a. Quartz is configured properly for clustering (there are various settings that make this possible).
 - b. The initial software setup has the proper configuration to support a clustered production environment.
 - c. Rice’s initial settings are in the file, `common-config-defaults.xml`.

Here are some of the parameters in the `common-config-defaults.xml` that setup Quartz for clustering:

```
<param name="useQuartzDatabase" override="false">true</param>
<param name="ksb.org.quartz.scheduler.instanceId" override="false">AUTO</param>
<param name="ksb.org.quartz.scheduler.instanceName" override="false">KSBSchedule
<param name="ksb.org.quartz.jobStore.isClustered" override="false">true</param>
<param name="ksb.org.quartz.jobStore.tablePrefix" override="false">KRSB_QRTZ_</p>
</pre>
```

If it becomes necessary to pass additional parameters to Quartz during rice startup, just add parameters in the `rice-config.xml` file prefixed with **ksb.org.quartz.***

The parameter `useQuartzDatabase` MUST be set to **true** for Quartz clustering to work. (This is required because it uses the database to accomplish coordination between the different scheduler instances in the cluster.)

2. Ensure that all service bus endpoint URLs are unique on each machine: Make sure that each Rice server in the cluster has a unique `serviceServletUrl` parameter in the `rice-config.xml` configuration file.

One way to accomplish this is to pass a system parameter into the JVM that runs the Tomcat server that identifies the IP and port number of that particular Tomcat Server. For example:

```
-Dhttp.url=129.79.216.156:8806
```

You can then configure your `serviceServletUrl` in the `rice-config.xml` to use that IP and port number.

```
<param name="serviceServletUrl">http://${http.url}/${app.context.name}/remoting/
```

You could have different values for `serviceServletUrl` in the `rice-config.xml` on each machine in the cluster.

3. If you are using notes and attachments in workflow, then the `attachment.dir.location` parameter must point to a shared file system mount (one that is mounted by all machines in the cluster).
4. The specifics of setting up and configuring a shared file system location are part of how you set up your infrastructure environment. Those are beyond the scope of this Guide.
5. In general, to accomplish a load-balanced clustered environment, you must implement some type of load balancing technology with session affinity (i.e., it keeps the browser client associated with the specific machine in the cluster that it authenticated with). An example of a load balancing appliance-software is the open source product, Zeus.

Running Multiple Instances of Rice Within a Single Tomcat Instance

There are two different structural methods to run multiple instances of Rice within a single Tomcat instance. You can use either method:

1. Run a staging and a test environment. This requires a rebuild of the source code.
2. Run multiple instances of a production environment. This requires modification of the Tomcat `WEB-INF/web.xml`.

Running a Staging and a Test Environment

To show you how to set up a staging and a test environment within one Tomcat instance, this section presents the configuration recipe as though it were a Quick Start Best Practices section. This means that this section will be laid out using the Quick Start Best Practices section format and system directory structure. It presents a basic process, method, and guide to what you need to do to get a staging and test environment up within a single Tomcat instance. You could accomplish this functionality many different ways; these sections present one of those ways.

This describes how to set up the Rice instances of `kualirice-stg` and `kualirice-tst` instances pointing to the same database. However, you could set up two different databases, one for staging and one for testing. How you configure Rice for the scenario of a database for the “stg” instance and a separate database for the “tst” instance depends on how you want to set up Rice. That scenario is not documented here.

- We are assuming that you performed all the installation steps above to compile the software from source and deploy the example `kualirice.war` file. This example begins with rebuilding the source to create a test and staging instance compilation.
- You must compile the source code with a different environment variable. To add the environment variable, environment, to the WAR file’s `WEB-INF/web.xml` file, recompile the source code with this parameter:

```
ant -Drice.environment=some-environment-variable dist-war
```

- To begin: Log in as the rice user.
- Shut down your Tomcat server.

```
cd /usr/local/tomcat/bin
./shutdown.sh
Using CATALINA_BASE: /usr/local/tomcat
Using CATALINA_HOME: /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME: /usr/java/jdk1.6.0_16
```

- Recompile your WAR files with the specific environment variables:

```
cd /opt/software/kuali/src/rice-release-1-0-2-br
ant -Drice.environment=stg dist-war
cd target/
cp -p kr-stg.war /usr/local/tomcat/webapps/kualirice-stg.war
```

```
cd /opt/software/kuali/src/rice-release-1-0-2-br
ant -Drice.environment=tst dist-war
cp -p rice-tst.war /usr/local/tomcat/webapps/kualirice-tst.war
```

- Adding an environment variable to the application config variable will setup Rice to point to the two different instances. To allow each instance to point to the same database, edit the rice-config.xml and modify the application.url to correctly point your Rice to load the correct setup:

```
<param name="application.url">http://yourlocalip:8080/kualirice-${environment}</param>
```

- Now start up your Tomcat server:

```
cd /usr/local/tomcat/bin
./startup.sh
Using CATALINA_BASE: /usr/local/tomcat
Using CATALINA_HOME: /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME: /usr/java/jdk1.6.0_16
```

If your Rice instances started up successfully, browse to the sites <http://yourlocalip:8080/kualirice-stg> and <http://yourlocalip:8080/kualirice-tst>. You should see the Rice sample application for each site.

- Next, shut down your Tomcat server:

```
cd /usr/local/tomcat/bin
./shutdown.sh
Using CATALINA_BASE: /usr/local/tomcat
Using CATALINA_HOME: /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME: /usr/java/jdk1.6.0_16
```

- To create specific configuration parameters for the specific instances of Rice, add this to the **rice-config.xml**.

```
<param name="config.location">/usr/local/rice/rice-config-${environment}.xml</param>
```

- Next, copy the **rice-config.xml** to both staging and test to enter instance-specific configuration into each of the resulting xml files:

```
cd /usr/local/rice
cp -p rice-config.xml rice-config-stg.xml
cp -p rice-config.xml rice-config-tst.xml
```

- Remove anything from **rice-config.xml** that is specific to the stg or tst implementation. Put those specific stg or tst parameters in the **rice-config-stg.xml** or **rice-config-tst.xml** file, respectively.
- Now start up your Tomcat server:

```
cd /usr/local/tomcat/bin
./startup.sh
Using CATALINA_BASE: /usr/local/tomcat
Using CATALINA_HOME: /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME: /usr/java/jdk1.6.0_16
```

If your Rice instances started up successfully, browse to the sites <http://yourlocalip:8080/kualirice-stg> and <http://yourlocalip:8080/kualirice-tst>. You should see the Rice sample application for each site.

- As a best practice:
 - Put all common properties and settings across all Rice instances in the **rice-config.xml**.
 - Put instance-specific settings in **rice-config-stg.xml** and **rice-config-tst.xml**.

Running Multiple Production Environments

This describes how to set up two production Rice instances running side by side.

Items specific to running a Production Platform:

1. The configuration parameter **environment** must be set to the text: **prd**
2. When the configuration parameter **environment** is set to **prd**, the code:
 - a. Sends email to specified individuals
 - b. Turns off some of the Rice “back doors”

This assumes that you performed all the installation steps above to compile the software from source and deploy the example **kualirice.war** file. This example starts from rebuilding the source to accomplish a test and staging instance compilation.

The high-level process for creating multiple Rice instances:

1. Create a **riceprd1** and **riceprd2** database for the first production and second production instance, respectively.
2. Build the WAR file from the source code.
3. Unzip the WAR file in a temporary work directory.
4. Add an environment variable, **prd1**, to the **WEB-INF/web.xml** in the **unzipped-war-file-directory**.
5. Re-zip the WAR file into **kualirice-prd1.war**.

6. Copy **kualirice-prd1.war** to `/usr/local/tomcat/webapps`.
7. Change the environment variable from **prd1** to **prd2** in the **WEB-INF/web.xml** in the **unzipped-war-file-directory**.
8. Re-zip the WAR file into **kualirice-prd2.war**.
9. Copy **kualirice-prd2.war** to `/usr/local/tomcat/webapps`.
10. In `/usr/local/rice`, copy **rice-config.xml** to **rice-config-prd1.xml**.
11. In `/usr/local/rice`, copy **rice-config.xml** to **rice-config-prd2.xml**.
12. In **rice-config.xml**, remove any instance-specific parameters.
13. Modify **rice-config-prd1.xml** for instance-specific parameters.
14. Modify **rice-config-prd2.xml** for instance-specific parameters.
15. Start up Tomcat.

Here are the details:

- Start by logging in as the rice user.
- Shut down your Tomcat server.

```
cd /usr/local/tomcat/bin
./shutdown.sh
Using CATALINA_BASE: /usr/local/tomcat
Using CATALINA_HOME: /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME: /usr/java/jdk1.6.0_16
```

- [Set Up the ImpEx Process to Build the Database](#) for the process to create the **riceprd1** and **riceprd2** databases.
- Set your directory to the rice home directory:

```
cd ~
vi impex-build.properties
```

- For the **rice-prd1** database, modify this in the ImpEx file:

```
#
# Uncomment these for a local MySQL database
#
import.torque.database = mysql
import.torque.database.driver = com.mysql.jdbc.Driver
import.torque.database.url = jdbc:mysql://kmysql:3306/riceprd1
import.torque.database.user=riceprd1
import.torque.database.schema=riceprd1
import.torque.database.password=kualirice
```

- Save the file, change directory to the folder where the **ImpEx build.xml** is, and create the database:

```
cd /opt/software/kuali/db/trunk/impex
ant create-schema
```

```
ant satellite-update
```

- You may receive this error because the ANT and SVN processes cannot write to a directory on the hard drive:

```
Buildfile: build.xml
```

```
Warning: Reference torque-classpath has not been set at runtime, but was found during  
build file parsing, attempting to resolve. Future versions of Ant may support  
referencing ids defined in non-executed targets.
```

```
satellite-update:
```

```
Warning: Reference torque-classpath has not been set at runtime, but was found during  
build file parsing, attempting to resolve. Future versions of Ant may support  
referencing ids defined in non-executed targets.
```

```
satellite-init:
```

```
[echo] Running SVN update in /opt/software/kuali/devdb/rice-cfg-dbs  
[svn] <Update> started ...  
[svn] svn: '/opt/software/kuali/devdb/rice-cfg-dbs' is not a working copy  
[svn] svn: Cannot read from '/opt/software/kuali/devdb/rice-cfg-dbs/.svn/locks' :  
[svn] <Update> failed !
```

```
BUILD FAILED
```

```
/opt/software/kuali/db/trunk/impex/build.xml:825: Cannot update dir /opt/software
```

```
Total time: 3 seconds
```

- If you received the error above, go to the window where the root user is logged in and execute this command:

```
rm -rf /opt/software/kuali/devdb/rice-cfg-dbs
```

- Then return to where you have the rice user logged in and re-execute the command:

```
ant satellite-update
```

- The creation of the Rice **riceprd1** database should begin at this time.
- For the rice-prd2 database, modify this in the ImpEx file:

```
#  
# Uncomment these for a local MySQL database  
#  
import.torque.database = mysql  
import.torque.database.driver = com.mysql.jdbc.Driver  
import.torque.database.url = jdbc:mysql://kmysql:3306/riceprd2  
import.torque.database.user=riceprd2  
import.torque.database.schema=riceprd2  
import.torque.database.password=kualirice
```

- Save the file, change directory to the folder where the ImpEx build.xml is, and create the database:

```
cd /opt/software/kuali/db/trunk/impex  
ant create-schema  
ant satellite-update
```

- You may get this error because the ANT and SVN processes cannot write to a directory on the hard drive:


```
Buildfile: build.xml
Warning: Reference torque-classpath has not been set at runtime, but was found du
build file parsing, attempting to resolve. Future versions of Ant may support
referencing ids defined in non-executed targets.
```

```
satellite-update:
Warning: Reference torque-classpath has not been set at runtime, but was found du
build file parsing, attempting to resolve. Future versions of Ant may support
referencing ids defined in non-executed targets.
```

```
satellite-init:
[echo] Running SVN update in /opt/software/kuali/devdb/rice-cfg-dbs
[svn] <Update> started ...
[svn] svn: '/opt/software/kuali/devdb/rice-cfg-dbs' is not a working copy
[svn] svn: Cannot read from '/opt/software/kuali/devdb/rice-cfg-dbs/.svn/f
[svn] <Update> failed !
```

```
BUILD FAILED
/opt/software/kuali/db/trunk/impex/build.xml:825: Cannot update dir /opt/software
Total time: 3 seconds
```

- If you received the error above, go to the window where the root user is logged in and execute this command:

```
rm -rf /opt/software/kuali/devdb/rice-cfg-dbs
```

- Then return to where you have the rice user logged in and re-execute the command:

```
ant satellite-update
```

- The creation of the Rice **riceprd2** database should begin at this time.
- Create a temporary work directory where you can unzip the WAR file, once it has finished building. Recompile your WAR files with the specific environment variable:

1. Execute this as root:

```
cd /opt/software/kuali
mkdir work
chmod -R 777 /opt/software/kuali/work
```

2. Execute this as the rice user to create the **kualirice-prd1.war** file:

```
cd /opt/software/kuali/src/rice-release-1-0-2-br
ant -Drice.environment=prd dist-war
cd target/
cp -p kr-prd.war /opt/software/kuali/work
cd /opt/software/kuali/work
mkdir files
unzip kr-prd.war -d files
cd files/WEB-INF/
```

3. Edit the web.xml with VI and change the top parameters to these:

```
<context-param>
```

```
<param-name>environment</param-name>
<param-value>prd</param-value>
</context-param>

<context-param>
  <param-name>rice-prd-instance-name</param-name>
  <param-value>prd1</param-value>
</context-param>
```

4. Zip the `kualirice-prd1.war` file and deploy it:

```
cd ..
zip -9 -r kualirice-prd1.war *
mv kualirice-prd1.war /usr/local/tomcat/webapps/
```

5. Execute this as the `rice` user to create the **kualirice-prd2.war** file:

```
cd WEB-INF
```

6. Edit the `web.xml` with VI and change the top parameters to these:

```
<context-param>
  <param-name>environment</param-name>
  <param-value>prd</param-value>
</context-param>

<context-param>
  <param-name>rice-prd-instance-name</param-name>
  <param-value>prd2</param-value>
</context-param>
```

7. Zip the **kualirice-prd2.war** file and deploy it:

```
cd ..
zip -9 -r kualirice-prd2.war *
mv kualirice-prd2.war /usr/local/tomcat/webapps
```

8. Remove the work directory:

```
cd ../../
rm -rf work
```

Create a Rice-specific set of configuration files:

```
cd /usr/local/rice
cp -p rice-config.xml rice-config-prd1.xml
cp -p rice-config.xml rice-config-prd2.xml
```

- Set the following in the `rice-config.xml`
 - Set the **config.location** for each Rice instance-specific setting
 - Set the settings for all instances in the **rice-config.xml**
- A minimal **rice-config.xml** might look like this:

```
<config>
```

```
<param name="config.location">/usr/local/rice/rice-config- $\{\text{rice-prd-instance-na}$ 

    <!-- Please fill in a value for this parameter! -->
    <param name="application.url">http://10.93.94.206:8080/kualirice- $\{\text{rice-p}$ 

    <param name="notification.basewebappurl"> $\{\text{application.url}\}$ /ken</param>
<param name="workflow.url"> $\{\text{application.url}\}$ /en</param>

<param name="plugin.dir">/usr/local/rice/plugins</param>

<param name="attachment.dir.location">/usr/local/rice/kew_attachments</param>

    <!-- log4j settings -->
<param name="log4j.settings.path">/usr/local/rice/log4j.properties</param>
<param name="log4j.settings.reloadInterval">5</param>

    <!-- Keystore Configuration -->
<param name="keystore.file">/usr/local/rice/rice.keystore</param>
<param name="keystore.alias">rice</param>

<param name="keystore.password">kualirice</param>

<!-- Dummy Login Filter - use if you don't want to go through CAS -->
<param name="filter.login.class">org.kuali.rice.kew.web.DummyLoginFilter</param>
<param name="filtermapping.login.1">/*</param>

</config>
```

- A minimal **rice-config-prd1.xml** might look this:

```
<config>

    <!-- set some datasource defaults -->

    <!-- MySQL example -->

<param name="datasource.objb.platform">MySQL</param>

<param name="datasource.platform">org.kuali.rice.core.database.platform.MySQLData
<param name="datasource.url">jdbc:mysql://mysql:3306/riceprd1</param>
<param name="datasource.username">riceprd1</param>
<param name="datasource.password">kualirice</param>

<param name="datasource.driver.name">com.mysql.jdbc.Driver</param>
```

```
<param name="datasource.pool.maxWait">10000</param>
<param name="datasource.pool.validationQuery">select 1</param>
```

```
<!-- Oracle example
```

```
    <param name="datasource.obj.platform">Oracle9i</param>
    <param name="datasource.platform">org.kuali.rice.core.database.platform.Oracle</param>
<param name="datasource.url">jdbc:oracle:thin:@localhost:1521:XE</param>
<param name="datasource.username">rice</param>

<param name="datasource.password">*** password ***</param>

<param name="datasource.driver.name">oracle.jdbc.driver.OracleDriver</param>
<param name="datasource.pool.maxWait">10000</param>
<param name="datasource.pool.validationQuery">select 1 from dual</param>
-->
</config>
```

- A minimal **rice-config-prd2.xml** might look like this:

```
<config>
    <!-- set some datasource defaults -->

    <!-- MySQL example -->
<param name="datasource.obj.platform">MySQL</param>
<param name="datasource.platform">org.kuali.rice.core.database.platform.MySQLData</param>
<param name="datasource.url">jdbc:mysql://mysql:3306/ricprd2</param>
<param name="datasource.username">riceprd1</param>
<param name="datasource.password">kualirice</param>
<param name="datasource.driver.name">com.mysql.jdbc.Driver</param>
<param name="datasource.pool.maxWait">10000</param>
<param name="datasource.pool.validationQuery">select 1</param>

    <!-- Oracle example
    <param name="datasource.obj.platform">Oracle9i</param>
    <param name="datasource.platform">org.kuali.rice.core.database.platform.Oracle</param>
<param name="datasource.url">jdbc:oracle:thin:@localhost:1521:XE</param>
<param name="datasource.username">rice</param>
<param name="datasource.password">*** password ***</param>
<param name="datasource.driver.name">oracle.jdbc.driver.OracleDriver</param>
<param name="datasource.pool.maxWait">10000</param>
<param name="datasource.pool.validationQuery">select 1 from dual</param>
-->
</config>
```

- Now start up your Tomcat server:

```
cd /usr/local/tomcat/bin
./startup.sh
Using CATALINA_BASE: /usr/local/tomcat
Using CATALINA_HOME: /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
```

Using JRE_HOME: /usr/java/jdk1.6.0_16

If your Rice instances started up successfully, browse to the sites <http://yourlocalip:8080/kualirice-prd1> and <http://yourlocalip:8080/kualirice-prd2>. You should see the Rice sample application for each site.

Keystore Implementation Variations

If multiple instances of Rice are running under the same Tomcat instance, they can use the same keystore. You can set up multiple keystores for multiple instances, but you must insert a parameter for each instance in the **WEB-INF/web.xml** to point to the different keystores. Beyond this, the set up depends on how you want your Tomcat instance configured and your implementation-specific parameter settings.

Chapter 5. Creating a Client Application

Developing a Rice application is essentially no different than other J2EE applications. Any tool that can be used for creating J2EE apps can be used for a Rice app. Essentially Rice is a set of libraries that are used with your project (like many other libraries a J2EE app includes) and configured for your needs.

The essential tools for developing a project are documented in the next section.

Development Tools

IDE (Integrated Development Environment)

This is the tool you will use to develop the source code and resources for your project. It can be a simple text editor if you want, however it is recommended to use one of the Java IDE tools available. Of these Eclipse, IntelliJ, and NetBeans are the most popular in today's market. Any of these will be fine for developing a Rice project. However, as we will learn about next, Rice provides its own tooling to help getting started with Eclipse. Eclipse is chosen due to its high use and that it is a free open source tool. The latest release is 'Indigo' and can be downloaded here:

<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/indigosr1>

Database

Rice applications can use a MySQL or Oracle database for persisting application data. Rice itself will use the database for supporting the various Rice modules (workflow, identity management, and so on). With each Rice distribution datasets are provided that can be used to create the initial database schema. You can choose to load the 'bootstrap' dataset, which provides the baseline data needed to run Rice, or the 'demo' dataset which adds additional demo data (such as example KIM data and workflow doc types). Although it is possible to provide a shared database for development, it is recommended for productivity reasons for each developer to have a local database installed. Both MySQL and Oracle provide freely available databases for development. Currently Rice has been tested with the following versions:

- **Oracle**
 - Oracle Database 10g
 - Oracle Database 11g
 - Oracle Express Edition (XE)

Use the Oracle JDBC Driver to connect to these databases.

Ensure that the Oracle database you intend to use encodes character data in a UTF variant by default. For Oracle XE, this entails downloading the "Universal" flavor of the binary, which uses AL32UTF8.

- **MySQL**
 - MySQL 5.1.+

Use the MySQL Connector/J (5.1.+) to connect to MySQL databases.

Note for our chosen database we must also download the corresponding database driver. This is a jar file we will need to make available to our web container for connecting to the database.

These supported databases can be downloaded with the following URLs.

Table 5.1. Locations for Database Software

| Software | Download Location |
|---|---|
| Oracle Standard and Enterprise Editions | http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html |
| Oracle Express Edition | http://www.oracle.com/technetwork/database/express-edition/downloads/index.html |
| Oracle JDBC DB Driver | http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html |
| MySQL | http://www.mysql.com/downloads/ |
| MySQL Connector/J JDBC Driver | http://www.mysql.com/products/connector/j/ |

Note for working with a MySQL database the MySQL Workbench (available for free download) is very useful and can save time for those new to MySQL.

Once the database provider is installed, we can then load one of the provided datasets using the Kuali ImpEx tool. The ImpEx tool is a Kuali-developed application which is based on Apache Torque. It reads in database structure and data from XML files in a platform independent way and then creates the resulting database in either Oracle or MySQL. To use this tool we simply provide configuration about the location of the source dataset, along with connectivity information for our target database. This is done by creating a properties file named 'impex-build.properties' in the user home directory. Once the configuration is complete, we can invoke the tool using ant or maven and our database will be created.

JDK

In order to support compilation of the application source code a JDK must be installed (Note this must be the JDK and not a Java Runtime Environment – JRE). Rice requires a JDK version of 1.6.x. Additionally, Rice has only been tested with the Sun JDK implementation. Therefore use of other implementations such as OpenJDK may have problems.

For machines running Windows, JDK 6 can be downloaded at the following URL:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

If you are on a Mac, then Java 6 should already be installed if you are up to date with the latest updates from Apple.

You will also want to set up your JAVA_HOME environment variable to point to the installation directory of your JDK. In both Windows and Mac environments, the java executable program should already be on your path. But if it is not, you will want to include JAVA_HOME/bin in your PATH environment variable.

In order to verify that your JDK has been installed successfully, open a command prompt and type the following:

```
java -version
```

You should see output similar to the following:

```
java version "1.6.0_26"  
Java(TM) SE Runtime Environment (build 1.6.0_26-b03-383-11A511)  
Java HotSpot(TM) 64-Bit Server VM (build 20.1-b02-383, mixed mode)
```

If you receive an error indicating that the "java" command could not be found, please ensure that the java command is on your machine's **PATH** environment variable.

Maven

Maven is the primary build tool used by the Kualu Rice project. Maven is based on a project object model (POM) that defines various standards and conventions surrounding the organization of a project. This facilitates a set of standard build goals and lifecycle phases (such as compile, test, package, etc.). Maven is particularly helpful in terms of dependency management. When building a Rice application using Maven, all of the dependent libraries will be pulled in automatically.

It is not a required for Rice enabled applications to be Maven projects. Again, Rice is essentially a set of jars that can be used with an application. However using Maven simplifies the setup process greatly. For example applications not using Rice must pull in and manage all of the third party libraries that are needed by Rice. That has an impact not only on initial project setup, but also each time that application is upgraded to a new Rice version.

To download version 3 of Maven, use the following link:

<http://maven.apache.org/download.html>

You will want to set your M2_HOME environment variable to point to the location where you unzipped Maven. You will additionally want to include M2_HOME/bin in your PATH environment variable so that maven can be executed from the command line without having to specify the full path.

In order to verify that Maven has been installed successfully and is available on the path, open a command prompt and type the following:

```
mvn -version
```

You should see output like the following:

```
Apache Maven 3.0.3 (r1075438; 2011-02-28 10:31:09-0700)  
Maven home: /usr/local/maven  
Java version: 1.6.0_26, vendor: Apple Inc.  
Java home: /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home  
Default locale: en_US, platform encoding: MacRoman  
OS name: "mac os x", version: "10.7.2", arch: "x86_64", family: "mac"
```

If you receive an error indicating that the "mvn" command could not be found, please ensure that the directory that includes the mvn executable (**M2_HOME/bin**) is on your machine's **PATH** environment variable.

Servlet Container

In order to run our Rice application we need have a servlet container. The servlet container serves the web requests for a J2EE application. There are many containers available for use, but Tomcat is most commonly used. Kualu Rice 2.0 supports the following Tomcat version:

- Tomcat 6 (Servlet API 2.5, JSP 2.1)

- Tomcat 7 (Servlet API 3.0, JSP 2.2)

For downloading and install instructions visit the Apache Tomcat site:

<http://tomcat.apache.org/>

For development purposes you can also choose to use an embedded application container such as Jetty. The Rice project provides a sample Jetty Server that can be used for your project. The next section will cover this in more detail.

New Project Setup

Now let's look at creating a new Rice enabled project. To do this, we will use a tool from the Rice project that performs most of the initial bootstrapping. The tool is included within the Rice project therefore we need to start by downloading the Rice 2.0 release. The Rice distribution can be downloaded at:

<http://kuali.org/download>

Or the project may be checked out through Subversion with the following repository location:

<https://svn.kuali.org/repos/rice>

Note the full project must be checked out, not just the tool. The tool creates the initial artifacts by copying from the Rice working copy.

The Rice project contents should be placed into a folder in the local file system. A standard practice is to create a top level directory named 'java', followed by a 'projects' directory, and then a directory named 'rice' that contains the actual project ('/java/projects/rice').

The particular tool we will be using was written in Groovy, therefore we need to download the Groovy runtime. This can be downloaded at the following URL:

<http://groovy.codehaus.org/Download>

Install instructions are also available on the above site. For users of Windows, a Windows-Installer can be downloaded which will install Groovy and perform any necessary configuration (including add groovy to your path).

Once groovy is installed we are ready to run the create project script. Start up a console (on Windows can use the PowerShell) and change into the directory that contains the Rice project (eg '/java/projects/rice'). From the root project folder, change into the scripts folder. This folder should contain a file named 'createproject.groovy'.

There are a few options supported by the create project script, but let's start with the most basic way of running. The command we will give is:

```
groovy createproject.groovy -name PROJECT_NAME
```

First we are invoking the groovy executable (this assumes groovy is on your path, if not the full path to the groovy executable needs to be specified). Groovy then expects the name of the script we want to run, which is 'creatproject.groovy'. Next we specify the one required argument for the create project script which is the name for the project we want to create. Assuming we want to create a new project named 'MyRiceApp', the command would be the following:

```
groovy createproject.groovy -name MyRiceApp
```

After typing the command hit enter to start the script. You should then see a prompt as follows:\

```
=====
                          WARNING
=====
This program will delete the following directory and replace it
with a new project:
  /java/projects/MyRiceApp
It will also create or replace the
following files in USER_HOME:
  1) C:\Users\jkneal.ADS\kuali/main/dev/MyRiceApp-config.xml
  2) C:\Users\jkneal.ADS\kuali/main/dev/rice.keystore
If this is not what you want, please supply more information:
  usage: groovy createproject -name PROJECT_NAME [-pdir PROJECT_DIR] [-
RICE_DIR] [-mdir MAVEN_HOME]
  Do you want to continue (yes/no)?
```

Type 'yes' and then enter to resume the program. You will then see logging output from the script about various files being created, the maven build, and finally should see printed instructions and how to complete the project setup.

Notice we did not tell the script where to put our new project nor where to find the Rice project. This is because the script defaults to the project location of '/java/projects'. If we want our project to be generated in a different location, we can do so by passing the directory path with the '-pdir' argument:

```
groovy createproject.groovy -name MyRiceApp -pdir /home/myapps
```

The project directory given will be the parent for the project folder. The script will create another folder within this with the same name as the given project name.

Similarly if our source Rice project is in another directory, we can specify that using the '-rdir' argument:

```
groovy createproject.groovy -name MyRiceApp -rdir /home/myapps/rice
```

Unlike this project directory argument, this does specify the full path to the project (nothing will be appended).

Finally the create project script gives us a couple more options for the project generation. We can include the Rice sampleapp in our project by passing the `-sampleapp` flag:

```
groovy createproject.groovy -name MyRiceApp -sampleapp
```

Having the various examples of the sampleapp can be very useful in particular if doing development with the KRAD framework.

Lastly, we can have a project generated that is setup to go against a standalone Rice instance. To do this we pass the `-standalone` flag:

```
groovy createproject.groovy -name MyRiceApp -standalone
```

Project Structure and Configuration Files

The result of running the create project script is a new maven based Rice client project. This includes the directory structures for building out your application, along with the necessary configuration files. Let's start by looking the directories that were created.

- Project Root (eg `‘/java/projects/myapp’`) – This is the root folder that was created to hold all the project contents. Within this folder you will find three sub-folders, a `‘.classpath’`, `‘.project’`, `‘instructions.txt’`, and `‘pom.xml’` file.
- `.settings` – This folder contains settings configuration for the Eclipse IDE
- `src` – This folder is for the application source files and resources. Within this folder is the standard maven directory breakdown:
 - `src/main/java` – Contains Java source code
 - `src/main/resources` – Contains resource files (XML and other resources)
 - `src/main/webapp` – Contains the application web content (JSP, tags, images, CSS, Script)
- `target` – This folder holds the build output such as generated classes and wars.

Along with the directories several files are created. These are as follows:

Table 5.2. Generated Files

| File | Description |
|--|--|
| <code>.classpath</code> | Eclipse file for managing the application classpath |
| <code>.project</code> | Eclipse project file |
| <code>pom.xml</code> | Maven Project File |
| <code>{project}-RiceDataSourceSpringBeans.xml</code> | Spring XML file containing Rice datasource configurations |
| <code>{project}-RiceJTASpringBeans.xml*</code> | Spring XML file containing JTA transaction configuration |
| <code>{project}-RiceSpringBeans.xml*</code> | Spring XML file containing the Rice module Configurers |
| <code>SpringBeans.xml*</code> | Spring XML file for Application beans |
| <code>{project}-SampleAppModuleBeans.xml*</code> | Spring XML file for Sample App beans (only created if <code>–sampleapp</code> option was given) |
| <code>OJB-repository-sampleapp.xml*</code> | OJB configuration file for the Sample App (only created if <code>–sampleapp</code> option was given) |
| <code>META-INF/{project}-config.xml*</code> | Default Rice configuration properties |
| <code>src/main/webapp/META-INF/web.xml</code> | Standard web deployment descriptor for J2EE applications |

Note

All of these files are located within the `src/main/resources` directory

In addition to the files created within the project, two files are created in the `‘{user home}/kuali/main/dev’` folder. These include:

- `{project}-config.xml` – Configuration file for application. This is where the settings for the database and other configurations are given.

- rice.keystore – Provides a secure key for consuming secured services running on a Rice server

Configuring Your Rice Application

Next we need to provide some configuration for our application that is custom to our environment (for example database connectivity). We can do this by modifying the properties available in {project}-config.xml (located in the /kuali/main/dev folder in user home).

Although there are many configuration properties available for customization, the following are required for getting started:

Table 5.3. Configuration Parameters

| Parameter | Description | Example |
|-------------------------|---|---|
| datasource.url | JDBC URL of database to connect to | jdbc:oracle:thin:@localhost:1521:XE jdbc:mysql://localhost:3306/kuldemo |
| datasource.username | User name for connecting to the server database | rice |
| datasource.password | Password for connecting to the server database | |
| datasource.obj.platform | Name of OJB platform to use for the database | Oracle9i or MySQL |
| datasource.platform | Rice platform implementation for the database | org.kuali.rice.core.database.platform.OraclePla org.kuali.rice.core.database.platform.MySQLP |
| datasource.driver.name | JDBC driver for the database | oracle.jdbc.driver.OracleDriver com.mysql.jdbc.Driver |

Importing into Eclipse

Now we have our project setup and are ready to begin development. Note at this point the application is completely runnable. We could do a maven deploy, copy the generated war to our tomcat server, and start up the application. However we are going to first import our project to Eclipse so that we will be ready to develop further application code.

Navigate to the Eclipse installation directory. There you should find an executable named 'eclipse.exe'. Once this file is found double click to start the IDE. When Eclipse starts up for the first time it will ask you to choose a workspace. This is a directory that Eclipse place newly created projects and will also read current projects from. A standard within the community is to use '/java/projects' for your working space. Note you can select the checkbox to use the directory as your default and Eclipse will not prompt on the next startup.

Eclipse Memory: It is generally needed and recommended to allocate additional JVM memory for the Eclipse runtime. This can be done by opening up the file named 'eclipse.ini' that exists in the root installation directory. At the end of the file you specify VM arguments as follows:

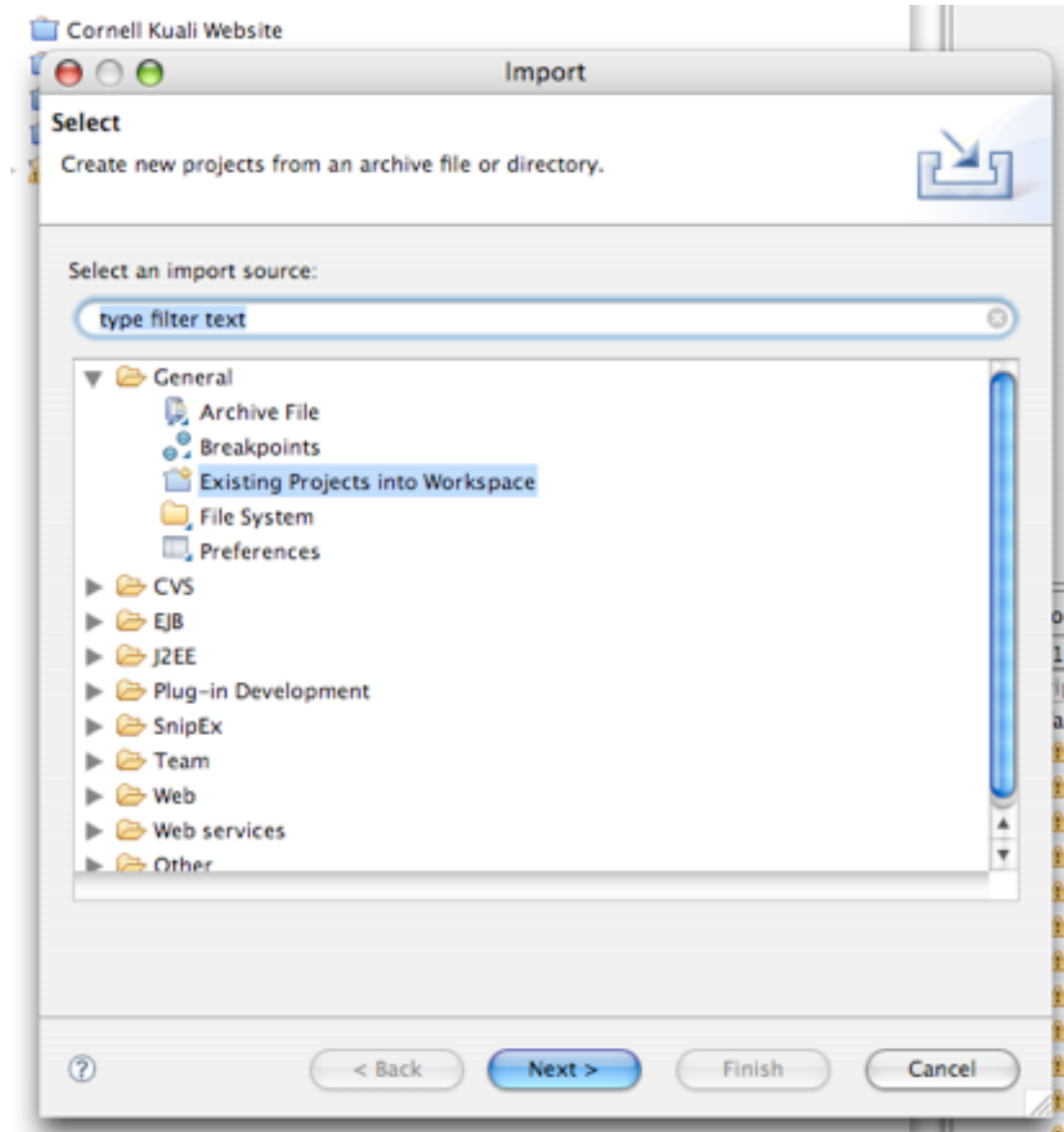
```
-vmargs
-Xms128m
-Xmx1024m
-XX:MaxPermSize=512m
```

The amount of memory allocated depends on the host machine. The above settings are for a machine with 4g of memory.

When working with Eclipse for the first time, there are additional plugins you will likely want to get. None of these are required by Rice and depend on your institutional development environment and how you plan to create your project. However, most projects today use SVN or GIT for source code control. Therefore an additional Eclipse plugin is needed for communicating with the repository. Also if you have chosen to use Maven (or used the create project script) the Eclipse Maven plugin will be very useful as well.

To bring a new project into eclipse, select the File-Import menu option. This should bring up a dialog as show in next figure.

Figure 5.1. Eclipse Import



For the import source select 'Existing Projects info Workspace'. This should bring up a dialog that looks like the following figure.

Figure 5.2. Eclipse Import Project Directory



Here click the 'Browse' button to locate the directory for the project. After selecting the project location click the 'Finish' button. Eclipse will then import the project contents and you are ready to begin coding!

Appendix A. Example Server Configurations

Single Server Configuration

An example setup of a single Tomcat server running MySQL server:

- CentOS v5.3 x64
- 4 GB Ram
- Intel Q6600 Quad Core Processor or better
- 1 TB RAID 1 configuration – SATA II 3.0 Gbps
- Apache
- Tomcat
- MySQL 5.1.x

Multi Server Configuration

An example of a multiple server configuration:

Web Servers

- CentOS v5.3 x64
- Intel (64 bit architecture)
- 1 GB Ram
- 80 GB RAID 1 configuration - SATA II 3.0 Gbps
- Apache
- Tomcat connector

Tomcat Servers

- CentOS v5.3 x64
- Intel Q6600 Quad Core Processor or better
- 4 GB Ram
- 80 GB RAID 1 configuration – SATA II 3.0 Gbps
- Tomcat

Web Servers – Content/Shared File System

- CentOS v5.3 x64
- Intel (64 bit architecture)
- 1 GB Ram
- 1 TB RAID 1 configuration
- SATA II 3.0 Gbps
- Apache
- Tomcat connector

Glossary

A

| | |
|--------------------------|--|
| Action List | A list of the user's notification and workflow items. Also called the user's Notification List. Clicking an item in the Action List displays details about that notification, if the item is a notification, or displays that document, if it is a workflow item. The user will usually load the document from their Action List in order to take the requested action against it, such as approving or acknowledging the document. |
| Action List Type | This tells you if the Action List item is a notification or a more specific workflow request item. When the Action List item is a notification, the Action List Type is "Notification." |
| Action Request | A request to a user or Workgroup to take action on a document. It designates the type of action that is requested, which includes: <ul style="list-style-type: none">• Approve: requests an approve or disapprove action.• Complete: requests a completion of the contents of a document. This action request is displayed in the Action List after the user saves an incomplete document.• Acknowledge: requests an acknowledgment by the user that the document has been opened - the doc will not leave the Action List until acknowledgment has occurred; however, the document routing will not be held up and the document will be permitted to transition into the processed state if necessary.• FYI: a notification to the user regarding the document. Documents requesting FYI can be cleared directly from the Action List. Even if a document has FYI requests remaining, it will still be permitted to transition into the FINAL state. |
| Action Request Hierarchy | Action requests are hierarchical in nature and can have one parent and multiple children. |
| Action Requested | The action one needs to take on a document; also the type of action that is requested by an Action Request. Actions that may be requested of a user are: <ul style="list-style-type: none">• Acknowledge: requests that the users states he or she has reviewed the document.• Approve: requests that the user either Approve or Disapprove a document.• Complete: requests the user to enter additional information in a document so that the content of the document is complete.• FYI: intended to simply makes a user aware of the document. |
| Action Taken | An action taken on a document by a Reviewer in response to an Action Request. The Action Taken may be: <ul style="list-style-type: none">• Acknowledged: Reviewer has viewed and acknowledged document.• Approved: Reviewer has approved the action requested on document. |

- Blanket Approved: Reviewer has requested a blanket approval up to a specified point in the route path on the document.
- Canceled: Reviewer has canceled the document. The document will not be routed to any more reviewers.
- Cleared FYI: Reviewer has viewed the document and cleared all of his or her pending FYI(s) on this document.
- Completed: Reviewer has completed and supplied all data requested on document.
- Created Document: User has created a document
- Disapproved: Reviewer has disapproved the document. The document will not be routed to any subsequent reviewers for approval. Acknowledge Requests are sent to previous approvers to inform them of the disapproval.
- Logged Document: Reviewer has added a message to the Route Log of the document.
- Moved Document: Reviewer has moved the document either backward or forward in its routing path.
- Returned to Previous Node: Reviewer has returned the document to a previous routing node. When a Reviewer does this, all the actions taken between the current node and the return node are removed and all the pending requests on the document are deactivated.
- Routed Document: Reviewer has submitted the document to the workflow engine for routing.
- Saved: Reviewer has saved the document for later completion and routing.
- Superuser Approved Document: [Superuser](#) has approved the entire document, any remaining routing is cancelled.
- Superuser Approved Node: Superuser has approved the document through all nodes up to (but not including) a specific node. When the document gets to that node, the normal Action Requests will be created.
- Superuser Approved Request: Superuser has approved a single pending Approve or Complete Action Request. The document then goes to the next routing node.
- Superuser Cancelled: Superuser has canceled the document. A Superuser can cancel a document without a pending Action Request to him/her on the document.
- Superuser Disapproved: Superuser has disapproved the document. A Superuser can disapprove a document without a pending Action Request to him/her on the document.

| | |
|---------------------|--|
| | <ul style="list-style-type: none">• Superuser Returned to Previous Node: Superuser has returned the document to a previous routing node. A Superuser can do this without a pending Action Request to him/her on the document. |
| Activated | The state of an action request when it has been sent to a user's Action List. |
| Activation | The process by which requests appear in a user's Action List |
| Activation Type | Defines how a route node handles activation of Action Requests. There are two standard activation types: <ul style="list-style-type: none">• Sequential: Action Requests are activated one at a time based on routing priority. The next Action Request isn't activated until the previous request is satisfied.• Parallel: All Action Requests at the route node are activated immediately, regardless of priority |
| Active Indicator | An indicator specifying whether an object in the system is active or not. Used as an alternative to complete removal of an object. |
| Ad Hoc Routing | A type of routing used to route a document to users or groups that are not in the Routing path for that Document Type. When the Ad Hoc Routing is complete, the routing returns to its normal path. |
| Annotation | Optional comments added by a Reviewer when taking action. Intended to explain or clarify the action taken or to advise subsequent Reviewers. |
| Approve | A type of workflow action button. Signifies that the document represents a valid business transaction in accordance with institutional needs and policies in the user's judgment. A single document may require approval from several users, at multiple route levels, before it moves to final status. |
| Approver | The user who approves the document. As a document moves through Workflow, it moves one route level at a time. An Approver operates at a particular route level of the document. |
| Attachment | The pathname of a related file to attach to a Note. Use the "Browse..." button to open the file dialog, select the file and automatically fill in the pathname. |
| Attribute Type | Used to strongly type or categorize the values that can be stored for the various attributes in the system (e.g., the value of the arbitrary key/value pairs that can be defined and associated with a given parent object in the system). |
| Authentication | The act of logging into the system. The Out of the box (OOTB) authentication implementation in Rice does not require a password as it is intended for testing purposes only. This is something that must be enabled as part of an implementation. Various authentication solutions exist, such as CAS or Shibboleth, that an implementer may want to use depending on their needs. |
| Authorization | Authorization is the permissions that an authenticated user has for performing actions in the system. |
| Author Universal ID | A free-form text field for the full name of the Author of the Note, expressed as "Lastname, Firstname Initial" |

B

| | |
|---------------------------|--|
| Base Rule Attribute | <p>The standard fields that are defined and collected for every Routing Rule. These include:</p> <ul style="list-style-type: none">• Active: A true/false flag to indicate if the Routing Rule is active. If false, then the rule will not be evaluated during routing.• Document Type: The Document Type to which the Routing Rule applies.• From Date: The inclusive start date from which the Routing Rule will be considered for a match.• Force Action: a true/false flag to indicate if the review should be forced to take action again for the requests generated by this rule, even if they had taken action on the document previously.• Name: the name of the rule, this serves as a unique identifier for the rule. If one is not specified when the rule is created, then it will be generated.• Rule Template: The Rule Template used to create the Routing Rule.• To Date: The inclusive end date to which the Routing Rule will be considered for a match. |
| Blanket Approval | <p>Authority that is given to designated Reviewers who can approve a document to a chosen route point. A Blanket Approval bypasses approvals that would otherwise be required in the Routing. For an authorized Reviewer, the Doc Handler typically displays the Blanket Approval button along with the other options. When a Blanket Approval is used, the Reviewers who are skipped are sent Acknowledge requests to notify them that they were bypassed.</p> |
| Blanket Approve Workgroup | <p>A workgroup that has the authority to Blanket Approve a document.</p> |
| Branch | <p>A path containing one or more Route Nodes that a document traverses during routing. When a document enters a Split Node multiple branches can be created. A Join Node joins multiple branches together.</p> |
| Business Rule | <ol style="list-style-type: none">1. Describes the operations, definitions and constraints that apply to an organization in achieving its goals.2. A restriction to a function for a business reason (such as making a specific object code unavailable for a particular type of disbursement). Customizable business rules are controlled by Parameters. |

C

| | |
|-------------|---|
| Campus | <p>Identifies the different fiscal and physical operating entities of an institution.</p> |
| Campus Type | <p>Designates a campus as physical only, fiscal only or both.</p> |
| Cancel | <p>A workflow action available to document initiators on documents that have not yet been routed for approval. Denotes that the document is void and should be disregarded. Canceled documents cannot be modified in any way and do not route for approval.</p> |

| | |
|--------------------------------------|---|
| Canceled | A routing status. The document is denoted as void and should be disregarded. |
| CAS - Central Authentication Service | http://www.jasig.org/cas - An open source authentication framework. Kuali Rice provides support for integrating with CAS as an authentication provider (among other authentication solutions) and also provides an implementation of a CAS server that integrates with Kuali Identity Management. |
| Client | A Java Application Program Interface (API) for interfacing with the Kuali Enterprise Workflow Engine. |
| Client/Server | The use of one computer to request the services of another computer over a network. The workstation in an organization will be used to initiate a business transaction (e.g., a budget transfer). This workstation needs to gather information from a remote database to process the transaction, and will eventually be used to post new or changed information back onto that remote database. The workstation is thus a Client and the remote computer that houses the database is the Server. |
| Close | A workflow action available on documents in most statuses. Signifies that the user wishes to exit the document. No changes to Action Requests, Route Logs or document status occur as a result of a Close action. If you initiate a document and close it without saving, it is the same as canceling that document. |
| Comma-separated value | A file format using commas as delimiters utilized in import and export functionality. |
| Complete | A pending action request to a user to submit a saved document. |
| Completed | The action taken by a user or group in response to a request in order to finish populating a document with information, as evidenced in the Document Route Log. |
| Country Restricted Indicator | Field used to indicate if a country is restricted from use in procurement. If there is no value then there is no restriction. |
| Creation Date | The date on which a document is created. |
| CSV | See comma-separated value |
| D | |
| Date Approved | The date on which a document was most recently approved. |
| Date Finalized | The date on which a document enters the FINAL state. At this point, all approvals and acknowledgments are complete for the document. |
| Deactivation | The process by which requests are removed from a user's Action List |
| Delegate | A user who has been registered to act on behalf of another user. The Delegate acts with the full authority of the Delegator. Delegation may be either Primary Delegation or Secondary Delegation . |
| Delegate Action List | A separate Action List for Delegate actions. When a Delegate selects a Delegator for whom to act, an Action List of all documents sent to the Delegator is displayed. |

For both [Primary](#) and [Secondary Delegation](#) the Delegate may act on any of the entries with the full authority of the Delegator.

| | |
|--------------------|--|
| Disapprove | A workflow action that allows a user to indicate that a document does not represent a valid business transaction in that user's judgment. The initiator and previous approvers will receive Acknowledgment requests indicating the document was disapproved. |
| Disapproved | A status that indicates the document has been disapproved by an approver as a valid transaction and it will not generate the originally intended transaction. |
| Doc Handler | The Doc Handler is a web interface that a Client uses for the appropriate display of a document. When a user opens a document from the Action List or Document Search, the Doc Handler manages access permissions, content format, and user options according to the requirements of the Client. |
| Doc Handler URL | The URL for the Doc Handler . |
| Doc Nbr | See Document Number . |
| Document | Also see E-Doc . An electronic document containing information for a business transaction that is routed for Actions in KEW. It includes information such as Document ID, Type, Title, Route Status, Initiator, Date Created, etc. In KEW, a document typically has XML content attached to it that is used to make routing decisions. |
| Document Id | See Document Number . |
| Document Number | A unique, sequential, system-assigned number for a document |
| Document Operation | A workflow screen that provides an interface for authorized users to manipulate the XML and other data that defines a document in workflow. It allows you to access and open a document by Document ID for the purpose of performing operations on the document. |
| Document Search | A web interface in which users can search for documents. Users may search by a combination of document properties such as Document Type or Document ID, or by more specialized properties using the Detailed Search. Search results are displayed in a list similar to an Action List. |
| Document Status | See also Route Status . |
| Document Title | The title given to the document when it was created. Depending on the Document Type, this title may have been assigned by the Initiator or built automatically based on the contents of the document. The Document Title is displayed in both the Action List and Document Search. |
| Document Type | The Document Type defines the routing definition and other properties for a set of documents. Each document is an instance of a Document Type and conducts the same type of business transaction as other instances of that Document Type. Document Types have the following characteristics: <ul style="list-style-type: none">• They are specifications for a document that can be created in KEW |

- They contain identifying information as well as policies and other attributes
- They defines the Route Path executed for a document of that type (Process Definition)
- They are hierarchical in nature may be part of a hierarchy of Document Types, each of which inherits certain properties of its [Parent Document Type](#).
- They are typically defined in XML, but certain properties can be maintained from a graphical interface

| | |
|-------------------------|---|
| Document Type Hierarchy | A hierarchy of Document Type definitions. Document Types inherit certain attributes from their parent Document Types. This hierarchy is also leveraged by various pieces of the system, including the Rules engine when evaluating rule sets and KIM when evaluating certain Document Type-based permissions. |
| Document Type Label | The human-readable label assigned to a Document Type. |
| Document Type Name | The assigned name of the document type. It must be unique. |
| Document Type Policy | These advise various checks and authorizations for instances of a Document Type during the routing process. |
| Drilldown | A link that allows a user to access more detailed information about the current data. These links typically take the user through a series of inquiries on different business objects. |
| Dynamic Node | An advanced type of Route Node that can be used to generate complex routing paths on the fly. Typically used whenever the route path of a document cannot be statically defined and must be completely derived from document data. |

E

| | |
|-----------------|--|
| ECL | <ol style="list-style-type: none">1. An acronym for Educational Community License.2. All Quali software and material is available under the Educational Community License and may be adopted by colleges and universities without licensing fees. The open licensing approach also provides opportunities for support and implementation assistance from commercial affiliates. |
| E-Doc | An abbreviation for electronic documents, also a shorthand reference to documents created with eDocLite. |
| eDocLite | A framework for quickly building workflow-enabled documents. Allows you to define document screens in XML and render them using XSL style sheets. |
| Embedded Client | A type of client that runs an embedded workflow engine. |
| Employee Status | Found on the Person Document; defines the employee's current employment classification (for example, "A" for Active). |
| Employee Type | Found on the Person Document; defines the employee's position classification (for example, "P" for Professional). |

| | |
|--------------------------|---|
| Entity | An Entity record houses identity information for a given Person, Process, System, etc. Each Entity is categorized by its association with an Entity Type. |
| Entity Attribute | Entities have directory-like information called Entity Attributes that are associated with them Entity Attributes make up the identity information for an Entity record. |
| Entity Type | Provides categorization to Entities. For example, a “System” could be considered an Entity Type because something like a batch process may need to interface with the application. |
| Exception | A workflow routing status indicating that the document routed to an exception queue because workflow has encountered a system error when trying to process the document. |
| Exception Messaging | The set of services and configuration options that are responsible for handling messages when they cannot be successfully delivered. Exception Messaging is set up when you configure KSB using the properties outlined in KSB Module Configuration. |
| Exception Routing | A type of routing used to handle error conditions that occur during the routing of a document. A document goes into Exception Routing when the workflow engine encounters an error or a situation where it cannot proceed, such as a violation of a Document Type Policy or an error contacting external services. When this occurs, the document is routed to the parties responsible for handling these exception cases. This can be a group configured on the document or a responsibility configured in KIM. Once one of these responsible parties has reviewed the situation and approved the document, it will be resubmitted to the workflow engine to attempt the processing again. |
| Extended Attributes | Custom, table-driven business object attributes that can be established by implementing institutions. |
| Extension Rule Attribute | One of the rule attributes added in the definition of a rule template that extends beyond the base rule attributes to differentiate the routing rule. A Required Extension Attribute has its "Required" field set to True in the rule template. Otherwise, it is an Optional Extension Attribute. Extension attributes are typically used to add additional fields that can be collected on a rule. They also define the logic for how those fields will be processed during rule evaluation. |

F

| | |
|--------------------------------|---|
| Field Lookup | The round magnifying glass icon found next to fields throughout the GUI that allow the user to look up reference table information and display (and select from) a list of valid values for that field. |
| Final | A workflow routing status indicating that the document has been routed and has no pending approval or acknowledgement requests. |
| Flexible Route Management | A standard KEW routing scheme based on rules rather than dedicated table-based routing. |
| FlexRM (Flexible Route Module) | The Route Module that performs the Routing for any Routing Rule is defined through FlexRM. FlexRM generates Action Requests when a Rule matches the |

data value contained in a document. An abbreviation of "Flexible Route Module."
A standard KEW routing scheme that is based on rules rather than dedicated table-based routing.

Force Action

A true/false flag that indicates if previous Routing for approval will be ignored when an [Action Request](#) is generated. The flag is used in multiple contexts where requests are generated (e.g., rules, ad hoc routing). If Force Action is False, then prior Actions taken by a user can satisfy newly generated requests. If it is True, then the user needs to take another Action to satisfy the request.

FYI

A workflow action request that can be cleared from a user's Action List with or without opening and viewing the document. A document with no pending approval requests but with pending Acknowledge requests is in Processed status. A document with no pending approval requests but with pending FYI requests is in Final status. See also [Ad Hoc Routing](#) and [Action Request](#).

G

Group

A Group has members that can be either [Principals](#) or other Groups (nested). Groups essentially become a way to organize Entities (via Principal relationships) and other Groups within logical categories.

Groups can be given authorization to perform actions within applications by assigning them as members of [Roles](#).

Groups can also have arbitrary identity information (i.e., [Group Attributes](#) hanging from them. Group Attributes might be values for "Office Address," "Group Leader," etc.

Groups can be maintained at runtime through a user interface that is capable of workflow.

Group Attribute

Groups have directory-like information called Group Attributes hanging from them. "Group Phone Number" and "Team Leader" are examples of Group Attributes.

Group Attributes make up the identity information for a Group record.

Group Attributes can be maintained at runtime through a user interface that is capable of workflow.

H

Hierarchical Tree Structure

A hierarchical representation of data in a graphical form.

I

Initialized

The state of an Action Request when it is first created but has not yet been Activated (sent to a user's Action List).

Initiated

A workflow routing status indicating a document has been created but has not yet been saved or routed. A Document Number is automatically assigned by the system.

Initiator A user role for a person who creates (initiates or authors) a new document for routing. Depending on the permissions associated with the Document Type, only certain users may be able to initiate documents of that type.

Inquiry A screen that allows a user to view information about a business object.

J

Join Node The point in the routing path where multiple branches are joined together. A Join Node typically has a corresponding [Split Node](#) for which it joins the branches.

K

KC - Kualii Coeus TODO

KCA - Kualii Commercial Affiliates A designation provided to commercial affiliates who become part of the Kualii Partners Program to provide for-fee guidance, support, implementation, and integration services related to the Kualii software. Affiliates hold no ownership of Kualii intellectual property, but are full KPP participants. Affiliates may provide packaged versions of Kualii that provide value for installation or integration beyond the basic Kualii software. Affiliates may also offer other types of training, documentation, or hosting services.

KCB – Kualii Communications Broker KCB is logically related to KEN. It handles dispatching messages based on user preferences (email, SMS, etc.).

KEN - Kualii Enterprise Notification A key component of the Enterprise Integration layer of the architecture framework. Its features include:

- Automatic Message Generation and Logging
- Message integrity and delivery standards
- Delivery of notifications to a user’s Action List

KEW – Kualii Enterprise Workflow Kualii Enterprise Workflow is a general-purpose electronic routing infrastructure, or workflow engine. It manages the creation, routing, and processing of electronic documents (eDocs) necessary to complete a transaction. Other applications can also use Kualii Enterprise Workflow to automate and regulate the approval process for the transactions or documents they create.

KFS – Kualii Financial System Delivers a comprehensive suite of functionality to serve the financial system needs of all Carnegie-Class institutions. An enhancement of the proven functionality of Indiana University's Financial Information System (FIS), KFS meets GASB and FASB standards while providing a strong control environment to keep pace with advances in both technology and business. Modules include financial transactions, general ledger, chart of accounts, contracts and grants, purchasing/accounts payable, labor distribution, budget, accounts receivable and capital assets.

KIM – Kualii Identity Management A Kualii Rice module, Kualii Identity Management provides a standard API for persons, groups, roles and permissions that can be implemented by an institution. It also provides an out of the box reference implementation that allows for a university to use Kualii as their Identity Management solution.

| | |
|--|--|
| KNS – Kuali Nervous System | A core technical module composed of reusable code components that provide the common, underlying infrastructure code and functionality that any module may employ to perform its functions (for example, creating custom attributes, attaching electronic images, uploading data from desktop applications, lookup/search routines, and database interaction). |
| KPP - Kuali Partners Program | The Kuali Partners Program (KPP) is the means for organizations to get involved in the Kuali software community and influence its future through voting rights to determine software development priorities. Membership dues pay staff to perform Quality Assurance (QA) work, release engineering, packaging, documentation, and other work to coordinate the timely enhancement and release of quality software and other services valuable to the members. Partners are also encouraged to tender functional, technical, support or administrative staff members to the Kuali Foundation for specific periods of time. |
| KRAD - Kuali Rapid Application Development | TODO |
| KRMS - Kuali Rules Management System | TODO |
| KS - Kuali Student | Delivers a means to support students and other users with a student-centric system that provides real-time, cost-effective, scalable support to help them identify and achieve their goals while simplifying or eliminating administrative tasks. The high-level entities of person (evolving roles-student, instructor, etc.), time (nested units of time - semesters, terms, classes), learning unit (assigned to any learning activity), learning result (grades, assessments, evaluations), learning plan (intentions, activities, major, degree), and learning resources (instructors, classrooms, equipment). The concierge function is a self-service information sharing system that aligns information with needs and tasks to accomplish goals. The support for integration of locally-developed processes provides flexibility for any institution's needs. |
| KSB – Kuali Service Bus | Provides an out-of-the-box service architecture and runtime environment for Kuali Applications. It is the cornerstone of the Service Oriented Architecture layer of the architectural reference framework. The Kuali Service Bus consists of: <ul style="list-style-type: none"> • A services registry and repository for identifying and instantiating services • Run time monitoring of messages • Support for synchronous and asynchronous service and message paradigms |
| Kuali | <ol style="list-style-type: none"> 1. Pronounced "ku-wah-lee". A partnership organization that produces a suite of community-source, modular administrative software for Carnegie-class higher education institutions. See also Kuali Foundation 2. (n.) A humble kitchen wok that plays an important role in a successful kitchen. |
| Kuali Foundation | Employs staff to coordinate partner efforts and to manage and protect the Foundation's intellectual property. The Kuali Foundation manages a growing portfolio of enterprise software applications for colleges and universities. A lightweight Foundation staff coordinates the activities of Foundation members for critical software development and coordination activities such as source code control, release engineering, packaging, documentation, project management, |

software testing and quality assurance, conference planning, and educating and assisting members of the Kualu Partners program.

Kualu Rice

Provides an enterprise-class middleware suite of integrated products that allow both Kualu and non-Kualu applications to be built in an agile fashion, such that developers are able to react to end-user business requirements in an efficient manner to produce high-quality business applications. Built with Service Oriented Architecture (SOA) concepts in mind, KR enables developers to build robust systems with common enterprise workflow functionality, customizable and configurable user interfaces with a clean and universal look and feel, and general notification features to allow for a consolidated list of work "action items." All of this adds up to providing a re-usable development framework that encourages a simplified approach to developing true business functionality as modular applications.

L

Last Modified Date

The date on which the document was last modified (e.g., the date of the last action taken, the last action request generated, the last status changed, etc.).

M

Maintenance Document

An e-doc used to establish and maintain a table record.

Message

The full description of a [notification message](#). This is a specific field that can be filled out as part of the Simple Message or Event Message form. This can also be set by the programmatic interfaces when sending notifications from a client system.

Message Queue

Allows administrators to monitor messages that are flowing through the Service Bus. Messages can be edited, deleted or forwarded to other machines for processing from this screen.

N

Namespace

A Namespace is a way to scope both [Permissions](#) and [Entity Attributes](#). Each Namespace instance is one level of scoping and is one record in the system. For example, "KRA" or "KC" or "KFS" could be a Namespace. Or you could further break those up into finer-grained Namespaces such that they would roughly correlate to functional modules within each application. Examples could be "KRA Rolodex", "KC Grants", "KFS Chart of Accounts".

Out of the box, the system is bootstrapped with numerous Rice namespaces which correspond to the different modules. There is also a default namespace of "KUALU".

Namespaces can be maintained at runtime through a maintenance document.

Note Text

A free-form text field for the text of a Note

Notification Content

This section of a [notification message](#) which displays the actual full message for the notification along with any other content-type-specific fields.

Notification Message The overall Notification item or Notification Message that a user sees when she views the details of a notification in her Action List. A Notification Message contains not only common elements such as Sender, Channel, and Title, but also content-type-specific fields.

O

OOTB Stands for "out of the box" and refers to the base deliverable of a given feature in the system.

Optimistic Locking A type of "locking" that is placed on a database row by a process to prevent other processes from updating that row before the first process is complete. A characteristic of this locking technique is that another user who wants to make modifications at the same time as another user is permitted to, but the first one who submits their changes will have them applied. Any subsequent changes will result in the user being notified of the optimistic lock and their changes will not be applied. This technique assumes that another update is unlikely.

Optional Rule Extension Attribute An Extension Attribute that is not required in a Rule Template. It may or may not be present in a [Routing Rule](#) created from the Template. It can be used as a conditional element to aid in deciding if a Rule matches. These Attributes are simply additional criteria for the Rule matching process.

Org Doc # The originating document number.

Organization Refers to a unit within the institution such as department, responsibility center, campus, etc.

Organization Code Represents a unique identifier assigned to units at many different levels within the institution (for example, department, responsibility center, and campus).

P

Parameter Component Code Code identifying the parameter Component.

Parameter Description This field houses the purpose of this parameter.

Parameter Name This will be used as the identifier for the parameter. Parameter values will be accessed using this field and the namespace as the key.

Parameter Type Code Code identifying the parameter type. Parameter Type Code is the primary key for its' table.

Parameter Value This field houses the actual value associated with the parameter.

Parent Document Type A Document Type from which another [Document Type](#) derives. The child type can inherit certain properties of the parent type, any of which it may override. A Parent Document Type may have a parent as part of a hierarchy of document types.

Parent Rule A Routing Rule in KEW from which another Routing Rule derives. The child Rule can inherit certain properties of the parent Rule, any of which it may override. A Parent Rule may have a parent as part of a hierarchy of Rules.

Permission Permissions represent fine grained actions that can be mapped to functionality within a given system. Permissions are scoped to [Namespace](#) which roughly correlate to modules or sections of functionality within a given system.

A developer would code authorization checks in their application against these permissions.

Some examples would be: "canSave", "canView", "canEdit", etc.

Permissions are aggregated by [Roles](#).

Permissions can be maintained at runtime through a user interface that is capable of workflow; however, developers still need to code authorization checks against them in their code, once they are set up in the system.

Attributes

1. Id - a system generated unique identifier that is the primary key for any Permission record in the system
2. Name - the name of the permission; also a human understandable unique identifier
3. Description - a full description of the purpose of the Permission record
4. Namespace - the reference to the associated [Namespace](#)

Relationships

1. Permission to [Role](#) - many to many; this relationship ties a Permission record to a Role that is authorized for the Permission
2. Permission to [Namespace](#) - many to one; this relationship allows for scoping of a Permission to a Namespace that contains functionality which keys its authorization checking off of said

| | |
|---------------------|---|
| Person Identifier | The username of an individual user who receives the document ad hoc for the Action Requested |
| Person Role | Creates or maintains the list used in selection of personnel when preparing the Routing Form document. |
| Pessimistic Locking | A type of lock placed on a database row by a process to prevent other processes from reading or updating that row until the first process is finished. This technique assumes that another update is likely. |
| Plugins | A plugin is a packaged set of code providing essential services that can be deployed into the Rice standalone server. Plugins usually contains only classes used in routing such as custom rules or searchable attributes, but can contain client application specific services. They are usually used only by clients being implemented by the 'Thin Client' method |
| Post Processor | A routing component that is notified by the workflow engine about various events pertaining to the routing of a specific document (e.g., node transition, status change, action taken). The implementation of a Post Processor is typically specific to a particular set of Document Types. When all required approvals are completed, the engine notifies the Post Processor accordingly. At this point, the Post Processor is responsible for completing the business transaction in the manner appropriate to its Document Type. |

| | |
|------------------------|--|
| Posted Date/Time Stamp | A free-form text field that identifies the time and date at which the Notes is posted. |
| Postal Code | Defines zip code to city and state cross-references. |
| Preferences | User options in an Action List for displaying the list of documents. Users can click the Preferences button in the top margin of the Action List to display the Action List Preferences screen. On the Preferences screen, users may change the columns displayed, the background colors by Route Status, and the number of documents displayed per page. |
| Primary Delegation | The Delegator turns over full authority to the Delegate. The Action Requests for the Delegator only appear in the Action List of the Primary Delegate. The Delegation must be registered in KEW or KIM to be in effect. |
| Principal | <p>A Principal represents an Entity that can authenticate into the system. One can roughly correlate a Principal to a login username. Entities can exist in KIM without having permissions or authorization to do anything; therefore, a Principal must exist and must be associated with an Entity in order for it to have access privileges. All authorization that is not specific to Groups is tied to a Principal.</p> <p>In other words, an Entity is for identity while a Principal is for access management.</p> <p>Also note that an Entity is allowed to have multiple Principals associated with it. The use case typically given here is that a person may apply to a school and receive one log in for the application system; however, once accepted, they may receive their official login, but use the same identity information set up for their Entity record.</p> |
| Processed | A routing status indicating that the document has no pending approval requests but still has one or more pending acknowledgement requests. |

R

| | |
|-----------------------------------|--|
| Recipient Type | The type of entity that is receiving an Action Request. Can be a user, workgroup, or role. |
| Required Rule Extension Attribute | An Extension Attribute that is required in a Rule Template. It will be present in every Routing Rule created from the Template. |
| Responsibility | See Responsible Party . |
| Responsibility Id | A unique identifier representing a particular responsibility on a rule (or from a route module) This identifier stays the same for a particular responsibility no matter how many times a rule is modified. |
| Responsible Party | The Reviewer defined on a routing rule that receives requests when the rule is successfully executed. Each routing rule has one or more responsible parties defined. |
| Reviewer | A user acting on a document in his/her Action List and who has received an Action Request for the document. |
| Rice | An abbreviation for Kualu Rice. |
| Role | Roles aggregate Permissions When Roles are given to Entities (via their relationship with Principals) or Groups an authorization for all associated Permissions is granted. |

| | |
|-----------------|---|
| Route Header Id | Another name for the Document Id . |
| Route Log | Displays information about the routing of a document. The Route Log is usually accessed from either the Action List or a Document Search. It displays general document information about the document and a detailed list of Actions Taken and pending Action Requests for the document. The Route Log can be considered an audit trail for a document. |
| Route Module | A routing component that the engine uses to generate action requests at a particular Route Node . FlexRM (Flexible Route Module) is a general Route Module that is rule-based. Clients can define their own Route Modules that can conduct specialized Routing based on routing tables or any other desired implementation. |
| Route Node | <p>Represents a step in the routing process of a document type. Route node "instances" are created dynamically as a document goes through its routing process and can be defined to perform any function. The most common functions are to generate Action Requests or to split or join the route path.</p> <ul style="list-style-type: none">• Simple: do some arbitrary work• Requests: generate action requests using a Route Module or the Rules engine• Split: split the route path into one or more parallel branches• Join: join one or more branches back together• Sub Process: execute another route path inline• Dynamic: generate a dynamic route path |
| Route Path | The path a document follows during the routing process. Consists of a set of route nodes and branches. The route path is defined as part of the document type definition. |
| Route Status | <p>The status of a document in the course of its routing:</p> <ul style="list-style-type: none">• Approved: These documents have been approved by all required reviewers and are waiting additional postprocessing.• Cancelled: These documents have been stopped. The document's initiator can 'Cancel' it before routing begins or a reviewer of the document can cancel it after routing begins. When a document is cancelled, routing stops; it is not sent to another Action List.• Disapproved: These documents have been disapproved by at least one reviewer. Routing has stopped for these documents.• Enroute: Routing is in progress on these documents and an action request is waiting for someone to take action.• Exception: A routing exception has occurred on this document. Someone from the Exception Workgroup for this Document Type must take action on this document, and it has been sent to the Action List of this workgroup.• Final: All required approvals and all acknowledgements have been received on these documents. <u>No changes are allowed to a document that is in Final status.</u> |

- **Initiated:** A user or a process has created this document, but it has not yet been routed to anyone's Action List.
- **Processed:** These documents have been approved by all required users, and is completed on them. They may be waiting for Acknowledgements. No further action is needed on these documents.
- **Saved:** These documents have been saved for later work. An author (initiator) can save a document before routing begins or a reviewer can save a document before he or she takes action on it. When someone saves a document, the document goes on that person's Action List.

Routed By User The user who submits the document into routing. This is often the same as the Initiator. However, for some types of documents they may be different.

Routing The process of moving a document through its route path as defined in its Document Type. Routing is executed and administered by the workflow engine. This process will typically include generating Action Requests and processing actions from the users who receive those requests. In addition, the Routing process includes callbacks to the Post Processor when there are changes in document state.

Routing Priority A number that indicates the routing priority; a smaller number has a higher routing priority. Routing priority is used to determine the order that requests are activated on a route node with sequential activation type.

Routing Rule A record that contains the data for the [Rule Attributes](#) specified in a [Rule Template](#). It is an instance of a Rule Template populated to determine the appropriate Routing. The Rule includes the Base Attributes, Required Extension Attributes, Responsible Party Attributes, and any Optional Extension Attributes that are declared in the Rule Template. Rules are evaluated at certain points in the routing process and, when they fire, can generate Action Requests to the responsible parties that are defined on them.

Technical considerations for a Routing Rules are:

- Configured via a GUI (or imported from XML)
- Created against a Rule Template and a Document Type
- The Rule Template and its list of Rule Attributes define what fields will be collected in the Rule GUI
- Rules define the users, groups and/or roles who should receive action requests
- Available Action Request Types that Rules can route
 - Complete
 - Approve
 - Acknowledge
 - FYI
- During routing, Rule Evaluation Sets are "selected" at each node. Default is to select by Document Type and Rule Template defined on the Route Node

- Rules match (or ‘fire’) based on the evaluation of data on the document and data contained on the individual rule
- Examples
 - If dollar amount is greater than \$10,000 then send an Approval request to Joe.
 - If department is “HR” request an Acknowledgment from the HR.Acknowledgers workgroup.

Rule Attribute

Rule attributes are a core KEW data element contained in a document that controls its Routing. It participates in routing as part of a Rule Template and is responsible for defining custom fields that can be rendered on a routing rule. It also defines the logic for how rules that contain the attribute data are evaluated.

Technical considerations for a Rule Attribute are:

- They might be backed by a Java class to provide lookups and validations of appropriate values.
- Define how a Routing Rule evaluates document data to determine whether or not the rule data matches the document data.
- Define what data is collected on a rule.
- An attribute typically corresponds to one piece of data on a document (i.e dollar amount, department, organization, account, etc.).
- Can be written in Java or defined using XML (with matching done by XPath).
- Can have multiple GUI fields defined in a single attribute.

Rule QuickLinks

A list of document groups with their document hierarchies and actions that can be selected. For specific document types, you can create the rule delegate.

Rule Template

A Rule Template serves as a pattern or design for the routing rules. All of the Rule Attributes that include both Required and _Optional_ are contained in the Rule Template; it defines the structure of the routing rule of FlexRM. The Rule Template is also used to associate certain Route Nodes on a document type to routing rules.

Technical considerations for a Rule Templates are:

- They are a composition of Rule Attributes
- Adding a ‘Role’ attribute to a template allows for the use of the Role on any rules created against the template
- When rule attributes are used for matching on rules, each attribute is associated with the other attributes on the template using an implicit ‘and’ logic attributes
- Can be used to define various other aspects to be used by the rule creation GUI such as rule data defaults (effective dates, ignore previous, available request types, etc)

S

| | |
|---------------------------|---|
| Save | A workflow action button that allows the Initiator of a document to save their work and close the document. The document may be retrieved from the initiator's Action List for completion and routing at a later time. |
| Saved | A routing status indicating the document has been started but not yet completed or routed. The Save action allows the initiator of a document to save their work and close the document. The document may be retrieved from the initiator's action list for completion and routing at a later time. |
| Searchable Attributes | <p>Attributes that can be defined to index certain pieces of data on a document so that it can be searched from the Document Search screen.</p> <p>Technical considerations for a Searchable Attributes are:</p> <ul style="list-style-type: none">• They are responsible for extracting and indexing document data for searching• They allow for custom fields to be added to Document Search for documents of a particular type• They are configured as an attribute of a Document Type• They can be written in Java or defined in XML by using Xpath to facilitate matching |
| Secondary Delegation | <p>The Secondary Delegate acts as a temporary backup Delegator who acts with the same authority as the primary Approver/the Delegator when the Delegator is not available. Documents appear in the Action Lists of both the Delegator and the Delegate. When either acts on the document, it disappears from both Action Lists.</p> <p>Secondary Delegation is often configured for a range of dates and it must be registered in KEW or KIM to be in effect.</p> |
| Service Registry | Displays a read-only view of all of the services that are exposed on the Service Bus and includes information about them (for example, IP Address, or Endpoint URL). |
| Simple Node | A type of node that can perform any function desired by the implementer. An example implementation of a simple node is the node that generates Action Requests from route modules. |
| SOA | An acronym for Service Oriented Architecture. |
| Special Condition Routing | This is a generic term for additional route levels that might be triggered by various attributes of a transaction. They can be based on the type of document, attributes of the accounts being used, or other attributes of the transaction. They often represent special administrative approvals that may be required. |
| Split Node | A node in the routing path that can split the route path into multiple branches. |
| Spring | The Spring Framework is an open source application framework for the Java platform. |
| State | Defines U.S. Postal Service codes used to identify states. |
| Status | On an Action List; also known as Route Status. The current location of the document in its routing path. |

| | |
|---------------------------|---|
| Submit | A workflow action button used by the initiator of a document to begin workflow routing for that transaction. It moves the document (through workflow) to the next level of approval. Once a document is submitted, it remains in 'ENROUTE' status until all approvals have taken place. |
| Superuser | A user who has been given special permission to perform Superuser Approvals and other Superuser actions on documents of a certain Document Type. |
| Superuser Approval | Authority given Superusers to approve a document of a chosen Route Node. A Superuser Approval action bypasses approvals that would otherwise be required in the Routing. It is available in Superuser Document Search. In most cases, reviewers who are skipped are not sent Acknowledge Action Requests. |
| Superuser Document Search | A special mode of Document Search that allows Superusers to access documents in a special Superuser mode and perform administrative functions on those documents. Access to these documents is governed by the user's membership in the Superuser Workgroup as defined on a particular Document Type. |

T

| | |
|-------------|--|
| Thread pool | A technique that improves overall system performance by creating a pool of threads to execute multiple tasks at the same time. A task can execute immediately if a thread in the pool is available or else the task waits for a thread to become available from the pool before executing. |
| Title | <p>A short summary of the notification message. This field can be filled out as part of the Simple Message or Event Message form. In addition, this can be set by the programmatic interfaces when sending notifications from a client system.</p> <p>This field is equivalent to the "Subject" field in an email.</p> |

U

| | |
|------|---|
| URL | An acronym for Uniform Resource Locator. |
| User | A person who can log in and use the application. This term is synonymous with "Principal" in KIM. "Whereas Entity Id represents a unique Person, Principal Id represents a set of login information for that Person." |

V

| | |
|--------|--|
| Viewer | A user(s) who views a document during the routing process. This includes users who have action requests generated to them on a document. |
|--------|--|

W

| | |
|--------------------|--|
| Web Service Client | A type of client that connects to a standalone KEW server using Web Services. |
| Wildcard | A character that may be substituted for any of a defined subset of all possible characters. |
| Workflow | Electronic document routing, approval and tracking. Also known as Workflow Services or Kualu Enterprise Workflow (KEW). The Kualu infrastructure service |

that electronically routes an e-doc to its approvers in a prescribed sequence, according to established business rules based on the e-doc content. See also [Kuali Enterprise Workflow](#).

Workflow Engine

The component of KEW that handles initiating and executing the route path of a document.

Workflow QuickLinks

A web interface that provides quick navigation to various functions in KEW. These include:

- Quick EDoc Watch: The last five Actions taken by this user. The user can select and repeat these actions.
- Quick EDoc Search: The last five EDocs searched for by this user. The user can select one and repeat that search.
- Quick Action List: The last five document types the user took action with. The user can select one and repeat that action.

X

XML

See also [XML Ingestor](#).

1. An acronym for Extensible Markup Language.
2. Used for data import/export.

XML Ingestor

A workflow function that allows you to browse for and upload XML data.

XML RuleAttribute

Similar in functionality to a RuleAttribute but built using XML only