Kuali Rice 2.0.0-b3 Installation Guide

Table of Contents

List of Figures

List of Tables

Chapter 1. What is Kuali Rice?

Overview and Benefits of Kuali Rice

Kuali Rice (also simply known as Rice) is an open source, module-based, enterprise class, set of integrated middleware products that allow both Kuali and non-Kuali applications to be built in an agile fashion so developers can create custom end-user business applications quickly and efficiently. Services are exposed through the Kuali Service Bus (KSB) and can be consumed by other Rice applications.

Rice employs the Service Oriented Architecture (SOA) concept and is structured with both a server-side piece and a client-side piece. This framework allows end developers to build robust systems with common enterprise workflow functionality and with customizable and configurable user interfaces that have a clean and universal look and feel.

On the server side, Kuali Rice is implemented as a group of services within a Servlet container. It can also run as a module within a web server such as Apache. Kuali Rice implements the Java Servlet specification from Sun Microsystems. This allows developers to design software that adds dynamic content to web servers using the Java programming language. Servlets are a server side technology that responds to web clients (typically web browsers) through a request/response paradigm.

On the client side, Kuali Rice has a flexible framework of pieces that can be included in a Rice client application.

The Rice Standalone Server is built on the client-server model and is provided as a web archive file (WAR). The Standalone version allows client applications to be configured to interface with the Rice server.

The designers of Kuali Rice built it with a modular architecture where each module performs a specific function that offers services to applications. The Rice architecture has five major modules:

- Kuali Service Bus (KSB)
- Kuali Enterprise Workflow (KEW)
- Kuali Enterprise Notification (KEN)
- Kuali Identity Management (KIM)
- Kuali Nervous System (KNS)

Note

The Kuali Nervous System is not administered from the Rice Standalone Server because it is a framework and not an application.

Rice provides a reusable development framework that encourages a simplified approach to developing true business functionality in modular applications.

Application and service developers can focus on solutions to solve business issues rather than on the technology. The Rice framework takes care of complex technical issues so that each application or service that adopts the framework can interoperate with little or no complexity. The framework also simplifies interoperation with services exposed by other applications.

In addition, Rice supports the sophisticated workflow processes typically required in higher education. It addresses workflow processes that involve human interaction (i.e., approval) as part of the flow.

Operate Securely

Rice has built-in security integration with support for data encryption, pluggable authentication, and pluggable authorization.

Licensing

Portions of Kuali are copyrighted by other parties, who are not listed here. Refer to NOTICE.txt and the licenses directory for complete copyright and licensing information. Questions about licensing should be directed to licensing@kuali.org.

Kuali Rice is available from the Kuali Foundation as open source software under the Educational Community License, version 2.0.

Chapter 2. Software Architecture

Kuali Rice Implementations

Kuali Rice is available for two types of implementations, Server (also known as client-server) and Bundled (packaged with Kuali applications).

Standalone Server

The Server is the most versatile implementation of Rice. It is a web application that can provide services to multiple applications that integrate with Kuali Rice at your institution. The server distribution contains a web archive or WAR file for the Kuali Rice standalone server. The Server distribution is the one you should use when your enterprise wants to run multiple Kuali applications or when you want to integrate other applications with Rice.

Bundled

When an application bundles all of the Rice functionality (including what is usually handled by the standalone server) into the client application, it's called a Bundled Distribution. For example, Kuali Financial Systems (KFS) has done this for their past releases. In those KFS releases, you did not need to set up and install a Rice standalone server; the necessary Rice functionality was bundled with KFS.

Bundled Distributions are not recommended for enterprise implementations, but are good for quick start, testing, and demonstrations.

Software Distributions

Source Code Distribution

The Source Code Distribution is available if you want to build Rice from scratch and create the standalone or binary libraries yourself.

Binary Distribution

The Binary Distribution (also known as the client distribution) is a collection of JAR files. It is used when other applications need to use your Rice implementation and you won't be using the Rice web application. It is designed for embedding Rice and can be used as a set of libraries for client applications.

The Binary Distribution of Kuali Rice is implemented as an application framework consisting of application programming interfaces (APIs), libraries, and the web framework. This allows you to construct a Kuali Rice application. All JARs and web content are included in this version.

Server Distribution

This is the distribution that contains the standalone server WAR.

Which Distribution to Use

In a typical enterprise deployment of Kuali Rice, a Standalone Rice server hosts numerous shared services and provides the most versatility. The composition of Rice contains an application framework — the APIs,

Libraries, and web framework that are used to construct a Rice application. You can configure subsequent Rice client applications to interact with these services as needed.

Obtaining Distributions and Source Code

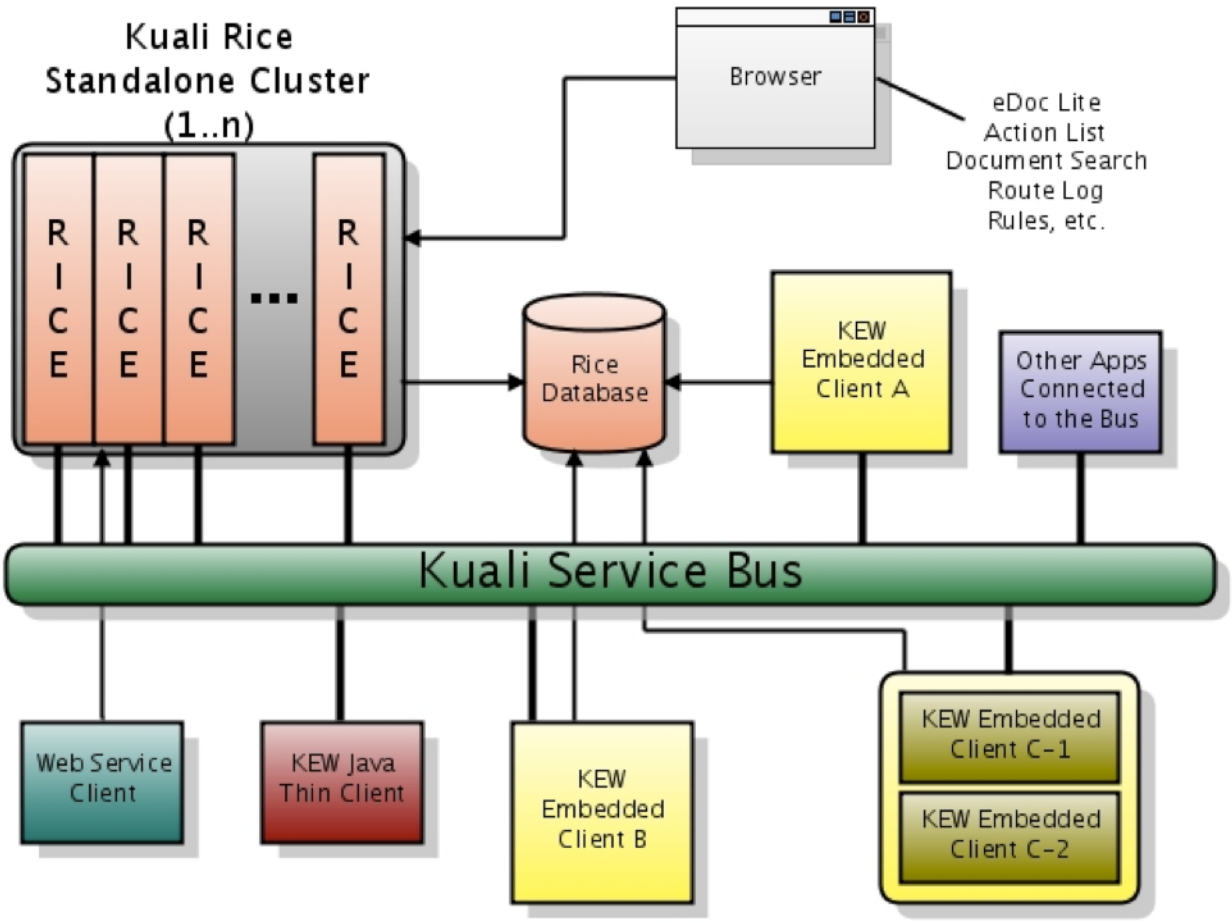
All three distributions, as well as the source code of the latest production release are available at here.

Chapter 3. Technical Overview

Architectural Diagram

This is a high-level picture of what an Enterprise Deployment of Rice might look like. This diagram includes representations of various client applications interacting with the Rice standalone server:

Figure 3.1. Kuali Rice 2.0.0-b3 Architectural Diagram



Modules

Kuali Rice has five core modules, linked together by the Kuali Service Bus (KSB):

1. KSB (Kuali Service Bus)

   Kuali Service Bus is a simple service bus geared toward easy service integration in an SOA.

2. KEW (Kuali Enterprise Workflow)

   Kuali Enterprise Workflow provides a common routing and approval engine that facilitates the automation of business processes across the enterprise. KEW was specifically designed to address the

requirements of higher education, so it is particularly well suited for routing mediated transactions across departmental boundaries.

3. KEN (Kuali Enterprise Notification)

   Kuali Enterprise Notification acts as a enabler for all university business-related communications by allowing end-users and other systems to push informative messages to the campus community in a secure and consistent manner.

4. KIM (Kuali Identity Management)

   Kuali Identity Management provides central management features for person identity characteristics, groups, roles, permissions, and their relationships to each other. All integration with KIM is accomplished using simple and consistent service APIs (Java or Web Service). KIM is built like all of the Kuali applications with Spring at its core, so that you can implement your own Identity Management (IdM) solutions behind the Service APIs. This provides you with the option to override the reference service implementations with your own to integrate with other Identity and Access Management systems at your enterprise.

5. KNS (Kuali Nervous System)

   Kuali Nervous System is a software development framework that enables developers to quickly build business applications in an efficient and agile fashion. KNS is an abstracted layer of “glue” code that provides developers easy integration with the other Rice components.

Module Architecture

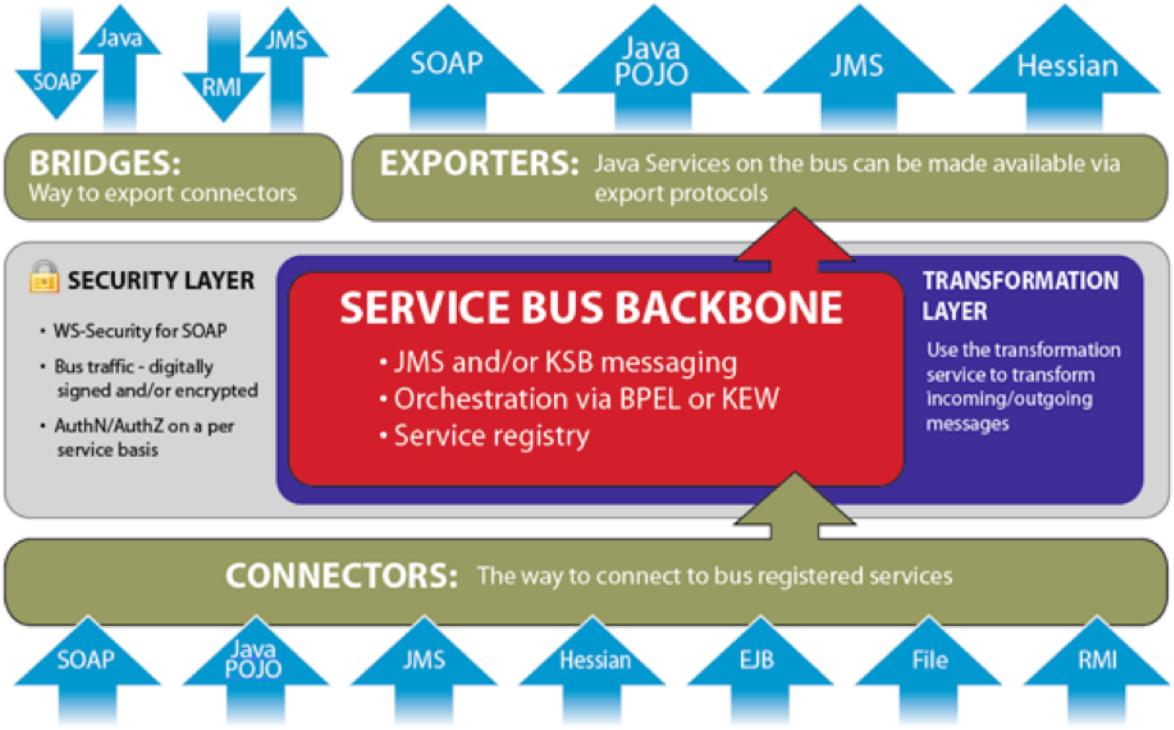
Kuali Rice is designed to run in a clustered environment and can be run on virtual machines.

KSB (Kuali Service Bus)

The Kuali Service Bus (KSB) is a lightweight service bus designed so developers can quickly develop and deploy services for remote and local consumption. You deploy services to the bus either using the Spring tool or programmatically. Services must be named when they are deployed to the bus. Services are acquired from the bus using their name.

At the heart of the KSB is a service registry. This registry is a listing of all services available for consumption on the bus. The registry provides the bus with the information necessary to achieve load balancing, failover, and more.

Figure 3.2. Kuali Service Bus



KSB Features

- **Transactional Asynchronous Messaging** – Call services asynchronously to support a 'fire and forget' model of calling services. Messaging participates in any existing JTA transactions (messages are not sent until the current running transaction is committed and are not sent if the transaction is rolled back). This increases the performance of service-calling code because it does not wait for a response.
- **Synchronous Messaging** - Call any service on the bus using a request-response paradigm.
- **Queue Style Messaging** - Execute Java services using message queues. When a message is sent to a queue, only one of the services listening for messages on the queue is given the message.
- **Topic Style Messaging** - Execute Java services using messaging topics. When a message is sent to a topic, all services listening for messages on the topic receive the message.
- **Quality of Service** - This KSB feature determines how queues and topics handle messages with problems. Time-to-live is supported, giving the message a configured amount of time to be handled successfully before exception handling is invoked for that message type. Messages can be given a specified number of retry attempts before exception handling is invoked. An increasing delay separates each calling. Exception handlers can be registered with each queue and topic for custom behavior when messages fail and Quality of Service limits have been reached.
- **Discovery** - Automatically discover services along the bus by service name. You do not need end-point URLs to connect to services.
- **Reliability** - Should problems arise, messages sent to services via queues or synchronous calls automatically fail-over to any other services bound to the same name on the bus. Services that are not

available are removed from the bus until they come back online, at which time they will be rediscovered for messaging.

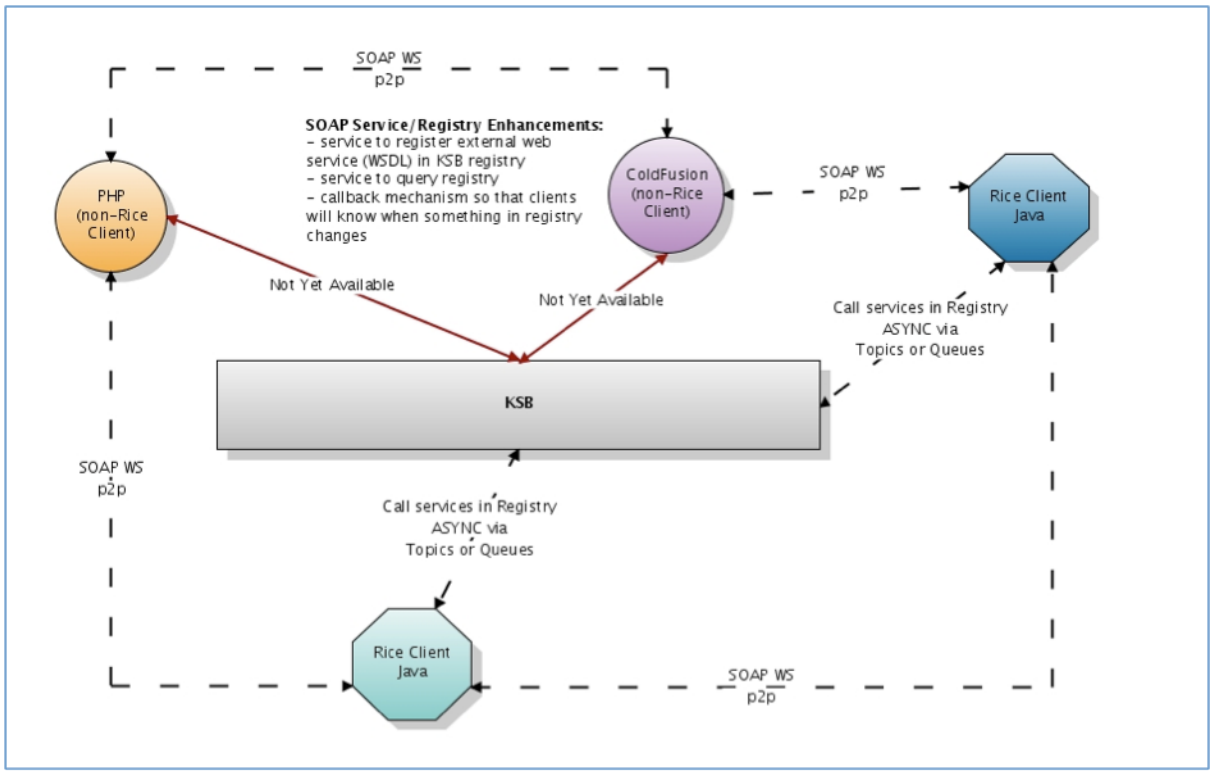
- **Persisted Callback** - Send callback objects with any message. These objects will be called each time a service is "requested." This provides a mechanism to pass along a message. In this way, deployed services can communicate back to a "callback registrant," such as an application registering a callback, with application data even as that data is moving through the system.
- **Primitive Business Activity Monitoring** - If turned on, each call to every service, including the parameters passed into that service, is recorded.
- **Spring-Based Integration** - KSB is designed with Spring-based integration in mind. For example, you might make an existing Spring-based POJO available for remote asynchronous calls.
- **Programmatic Integration** - If you do not use Spring configuration, you can configure KSB programmatically. Services can also be added and removed from the bus programmatically at runtime.

Bean Based Services

Typically, KSB programming is centered on exposing Spring-configured beans to other calling code using a number of different protocols. Using this paradigm, the client developer and the organization can rapidly build and consume services.

Overview of Supported Service Protocols

Figure 3.3. Supported Service Protocols



Note

This drawing is conceptual and not representative of true deployment architecture.

Essentially, the KSB is a registry with service-calling behavior on the client end (for Java clients). All policies and behaviors (aysnc vs. sync) are coordinated on the client.

KSB offers clients some very attractive messaging features:

- Synchronization of message sending with currently running transaction (In other words, all messages sent during a transaction are ONLY sent if the transaction is successfully committed.)
- Failover: If a call to a service comes back with a 404 (or various other network-related errors), the client will try to call other services of the same name on the bus. This is for both sync and async calls.
- Load balancing: Clients will round-robin call services of the same name on the bus. Proxy instances are, however, bound to single machines. This is useful if you want to keep a line of communication open to a single machine for long periods of time
- Topics and Queues: Used for controlling the execution of services
- Persistent messages: When using message persistence, a message cannot be lost. It will be persisted until it is sent.
- Message Driven Service Execution: Bind standard JavaBean services to messaging queues for message-driven beans

KEW (Kuali Enterprise Workflow)

The Kuali Enterprise Workflow (KEW) is a content-based routing engine. To enter the routing process, a user creates a document from a process definition and submits it to the workflow engine for routing. The engine then makes routing decisions based on the XML content of the document.

KEW is built for educational institutions to use for business transactions in the form of electronic documents that require approval from multiple parties. For example, these types of transactions are capably handled with KEW:

- Transfer funds
- Hire and terminate employees
- Complete and approve timesheets
- Drop a course

KEW Features

KEW is a set of services, APIs, and GUIs with these features:

- **Action List** for each user, also known as a user's work list
- **Document searching**
- **Route log**: Document audit trail
- **Flexible process definition**: Splits, joins, parallel branches, sub-processes, dynamic process generation
- **Rules engine**

- **Email notification**
- **Notes and attachments**
- Wide array of **pluggable components** to customize routing and other pieces of the system
- **eDocLite**: Framework for creating simple documents quickly
- **Plugin architecture**: Packaging and deployment of application plugins or deployment of routing components to the Rice standalone server at runtime

KEN (Kuali Enterprise Notification)

Kuali Enterprise Notification (KEN) acts as a broker for all university business-related communications by allowing end-users and other systems to push informative messages to the campus community in a secure and consistent manner. All notifications process asynchronously and are delivered to a single list where other messages such as workflow-related items (KEW action items) also reside. In addition, end-users can configure their profile to have certain types of messages delivered to other end-points such as email, mobile phones, etc.

Why Use KEN?

1. Easily leverage its functionality through the KSB or over SOAP
2. Access a full list of archives and logs so that you can easily find messages that were sent in the past
3. Eliminate sifting through your email inbox to find what you need
4. It guarantees delivery of messages, even to large numbers of recipients

KEN Features

A Single List for All Notifications: Find the things you have to do, things you want to know about, and things you need to know about. This includes workflow items from KEW, all in one place.

Eliminate Email Pains: Don't sift through piles of spam to find that one thing you need to do. You control who uses KEN, so there is no spam.

Flexible Content Types: No core programming is needed to customize the fields and data for a notification. You may use XML, XSD, and XSL to dynamically extend, validate, and render new content types.

Multiple Integration Interfaces: Use KEN’s Java services and web services (exposed on the KSB) to send messages from one system to another, or use the Rice generic message-sending form (with workflow built in) to send messages by hand.

Audit Trail: Track exactly who received a notification and when they received it.

Multiple Ways to Notify: All messages are sent to a user's notification list; however, users can also choose to have "ticklers" sent to their email inboxes, their mobile phones, and more. You can also build pluggable "ticklers" using the KEN framework.

Robust Searching and List Capabilities: Search for notifications by multiple fields such as priority, type, senders, and more. Save searches for later, and take actions on your notifications right from your list.

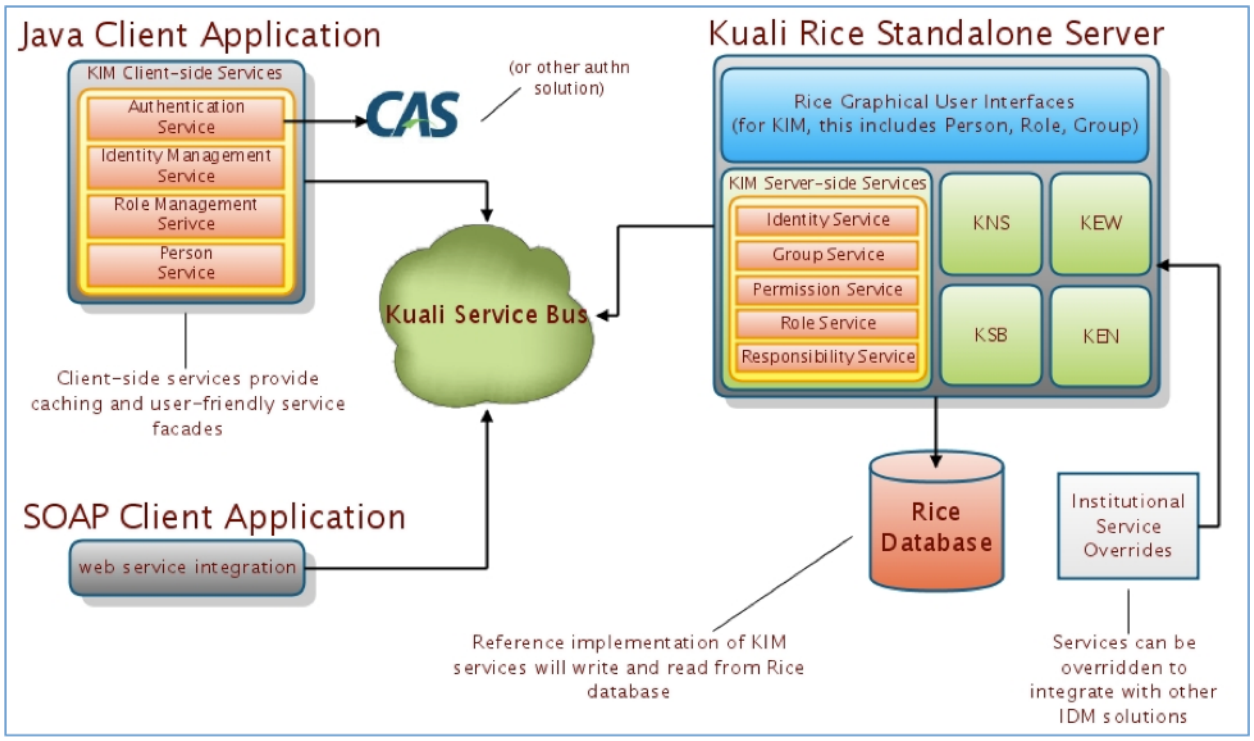
Security: Basic authorization comes out of the box along with single-sign-on. In addition, web service calls support SSL transport encryption and digital signing using X.509 certificates. KEN also allows you to build your own security plugins.

User and Group Management: Basic user and group management are provided, along with hooks for customizing KEN to point at other identity management systems, such as LDAP.

KIM (Kuali Identity Management)

The Kuali Identity Management (KIM) provides identity and access management services to Rice and other applications. All KIM services are available on the service bus with both SOAP and Java serialization endpoints. KIM provides a service layer and a set of GUIs that you can use to maintain identity information.

Figure 3.4. KIM Architecture Diagram



KIM Features

KIM provides a reference implementation of services. It also allows customization and/or replacement to facilitate integration with institutional services or other third-party identity management solutions. This allows the core KIM services to be overridden piecemeal. For example, you can override the Identity Service, but keep the Role Service.

KIM consists of these services, which encompass its API:

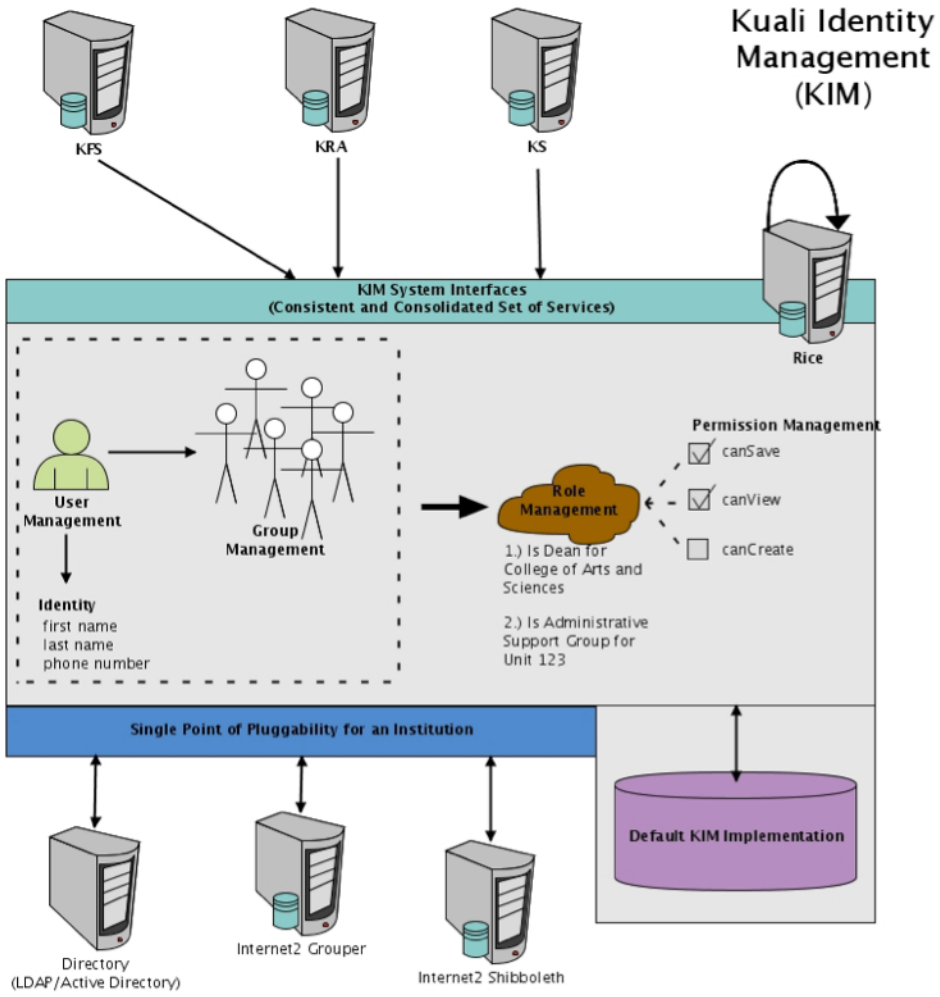
- Read-only services:
  - IdentityService
  - GroupService
  - PermissionService

- RoleService
- ResponsibilityService
- AuthenticationService

- Update services that allow write operations
- A permission service that evaluates permissions: KIM provides plug points for implementing custom logic for permission checking, such as permission checks based on hierarchical data.

A more detailed visual of the KIM architecture:

Figure 3.5. KIM Architecture Detail

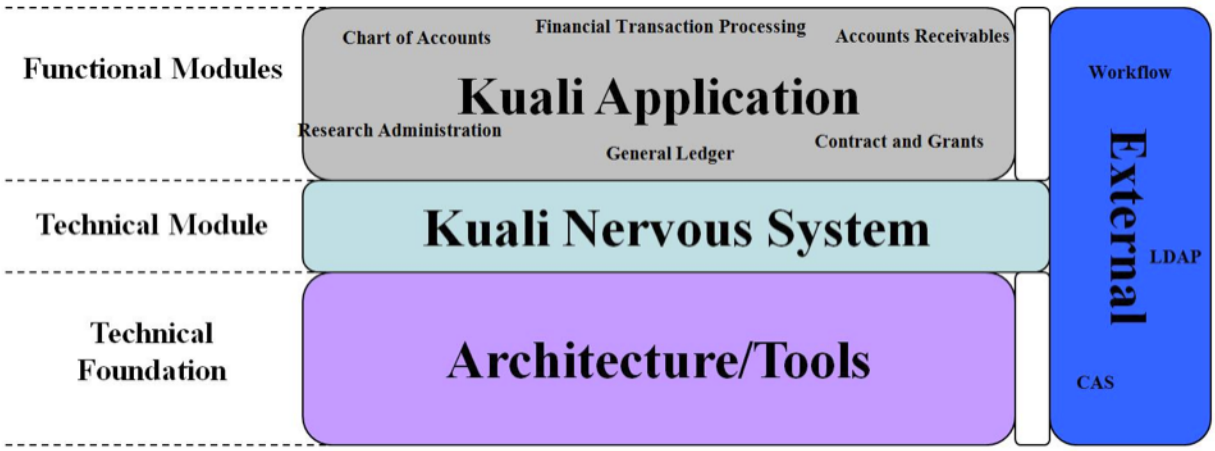


KNS (Kuali Nervous System)

The Kuali Nervous System (KNS) is the core of the Kuali Rice system. It embraces a document-centric (business process) model that uses workflow as a central concept. It is also the web application

development framework for Rice and is the core technical module in Rice, leveraging reusable code components to provide functionality.

Figure 3.6. Kuali Nervous System



What is KNS?

The Kuali Nervous System is a:

- Framework to enforce consistency
- Means to adhere to development standards and architectural principles
- Stable core for efficient development
- Means of reducing the amount of code written through code re-use

KNS Architectural Diagram

The KNS architecture provides a number of important services that are vital to the overall operation and effectiveness of the system.

Figure 3.7. KIM Architecture Detail

Physical Architecture Overview

Production Platform

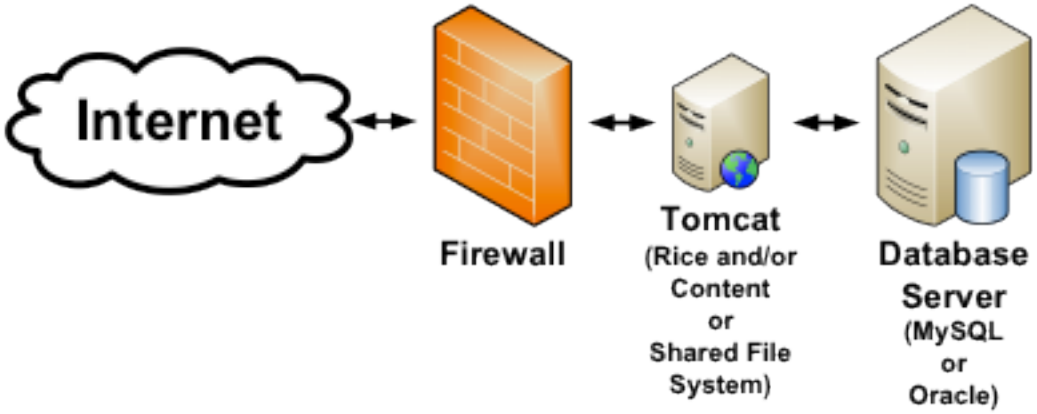
Since the builders of the Rice platform constructed it on open source technologies, your scale and use of Rice software determine the layout of the logical and physical hardware you need to support your implementation. Below are several conceptual models for implementation of Rice that are certainly not end solutions. Your solution depends on your implementation scale and budget.

Production Platform

The production platform that you deploy for your implementation can vary quite widely. The first example, the most basic platform structure, would be the most economical solution in terms of hardware. The Tomcat

server can serve up all web service and HTTP requests and store all content. (Of course, you could load-balance multiple Tomcat servers across machines.) A picture of the logical structure:

Figure 3.8. Conceptual Production Architecture, example 1

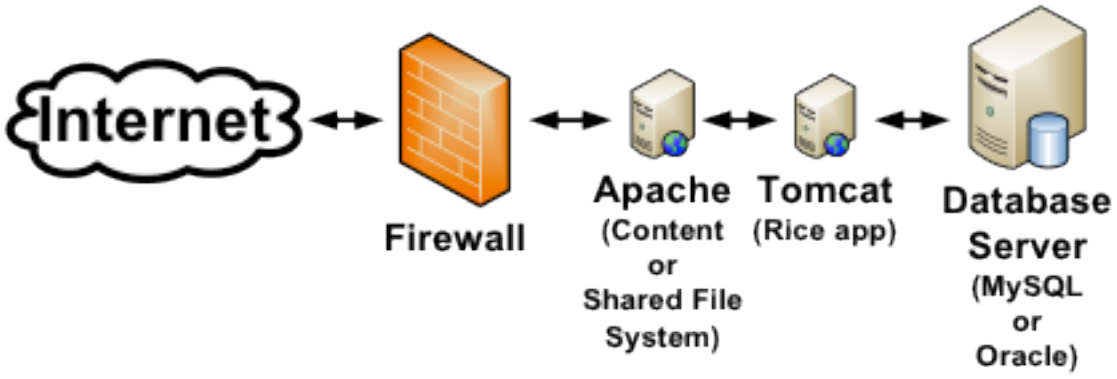


For this architecture, we recommend this minimum:

- Server running Tomcat container: Minimum 2 GB main memory

The next example has you offload the web requests and content to an Apache HTTP server in front of the Tomcat server:

Figure 3.9. Conceptual Production Architecture, example 1

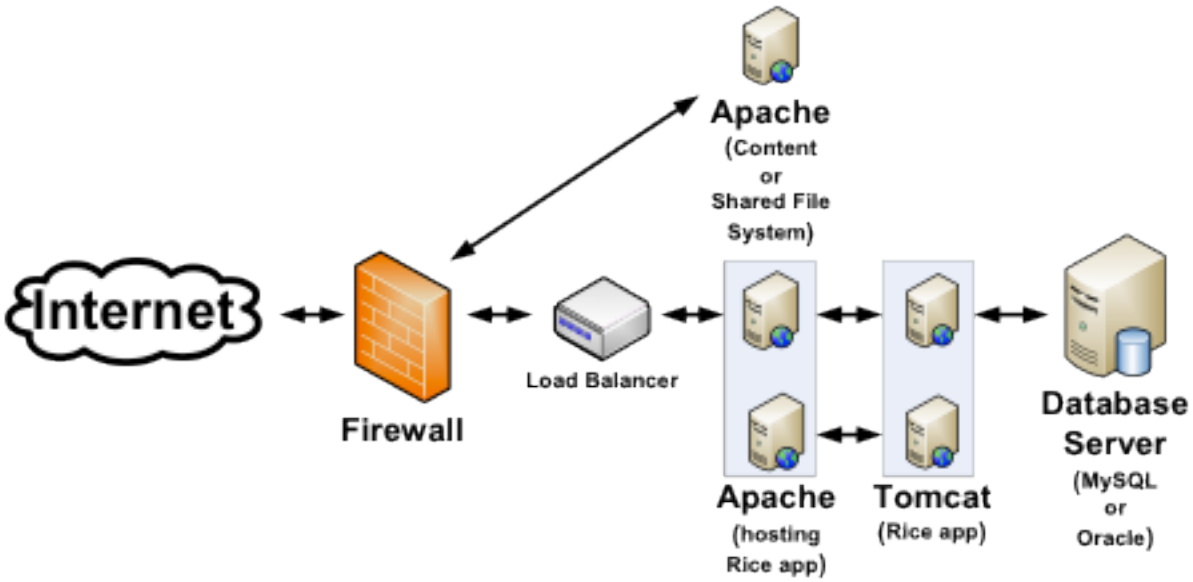


For this architecture, we recommend this minimum:

- Server running Apache: Minimum 1 GB main memory
- Server running Tomcat container: Minimum 2 GB main memory

And finally, the recommended solution, where the focus of the environment structure is based upon maximal scaling for the Rice application:

Figure 3.10. Recommended Conceptual Production Architecture



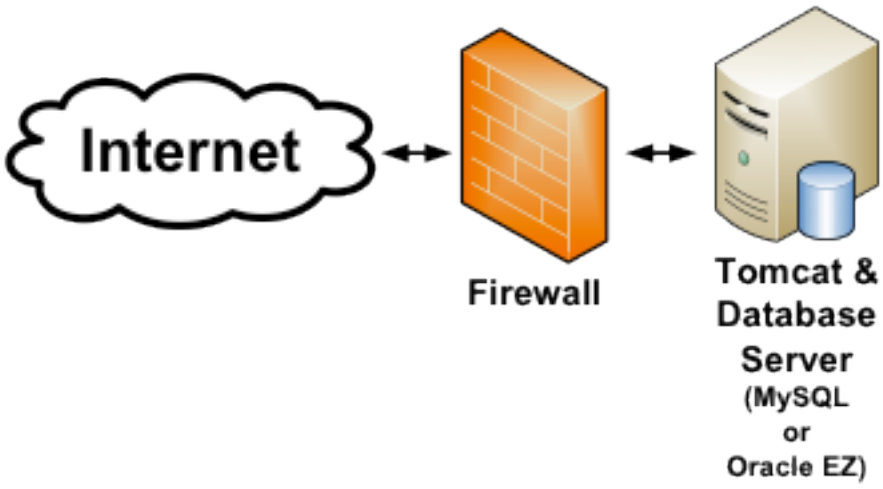
For this architecture, we recommend this minimum:

- Load Balancer
- Each server running Apache: Minimum 1 GB main memory
- Each server running Tomcat container: Minimum 2 GB main memory

Development Platform

The most basic platform for development has the Tomcat container and a MySQL server running on the same machine as your development tools:

Figure 3.11. Recommended Conceptual Production Architecture



The best option is to use the Tomcat container to serve up web service and HTTP requests. This has the least number of software layers between your development/debugging/IDE environment and the Tomcat container.

Chapter 4. Installation Steps

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).

Warning

Only platforms and configurations that have been tested and are known to work with Rice are described within this guide.

Minimum System Requirements and Recommendations

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 Kuali Rice.

Kuali Rice is typically deployed with the database server separate from the application server. Requirements below are for the Rice Standalone 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)
    - Additional space needed if storing attachments

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 and 7)
- Mac OS X (10.6)
- 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 these operating systems, although the software has not been tested here:

- Sun Microsystems Solaris

- IBM AIX

Required Software

All of these packages are required on the server with Rice:

- 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.

- Servlet container (Apache Tomcat 5.5+, or Jetty 6.1.1)
- Apache Ant 1.8.1
- Maven 3

Required Database

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.

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

- **Oracle 10g**
  - Oracle Database 10g Release 2 (10.2.0.4)
  - Oracle Database 10g Release 2 (10.2.0.4) JDBC Driver
- **MySQL**
  - MySQL 5.1.+
  - MySQL Connector/J (5.1.+)

You can 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.

Other Required Software

Subversion 1.5 or greater if checking out projects from Kuali source control

Sources for Required Software

Table 4.1.

Software	Download Location
Sun Java JDK 1.6.x	http://java.sun.com/javase/downloads/index.jsp
Apache Ant 1.7.1	http://mirror.olnevhost.net/pub/apache/ant/binaries/apache-ant-1.7.1-bin.tar.gz

Software	Download Location
Maven 2.0.9	http://archive.apache.org/dist/maven/binaries/apache-maven-2.0.9-bin.tar.gz
MySQL Connector/J JDBC Driver	http://www.mysql.com/products/connector/j/
Oracle 10g XPress	http://www.oracle.com/technology/products/database/xe/index.html
Oracle JDBC DB Driver	http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc_10201.html
Squirrel SQL client	http://squirrel-sql.sourceforge.net/
Tomcat 5.5.27	http://mirrors.homepagle.com/apache/tomcat/tomcat-5/v5.5.27/bin/apache-tomcat-5.5.27.zip

Software Distributions

The Kuali Rice software is available through three different distributions:

Table 4.2.

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

The Kuali Foundation makes the most current version of the Rice software available at: http://rice.kuali.org/download

Installation Checklist

To install Kuali Rice 1.0.3, follow these steps:

1. Install a database. Refer to the appendices for detailed instructions on how to do this.
2. Create a non-privileged user that will run Rice.
3. Install a JDK
4. Configure the Operating System.
5. Install the JDBC drivers.
6. Set up ImpEx process to create the database schema and populate it.
7. Obtain the WAR file from the Rice server distribution.

8. Install and configure a servlet container.

9. Test the installation

Create a Non-privileged User for Rice To Use

Typically, you use this command to add a user to CentOS:

```
# useradd –d /home/rice –s `which bash` rice
# passwd rice
New UNIX password: <enter kualirice>
Retype new UNIX password: <enter kualirice>
passwd: all authentication tokens updated successfully.
```

Distribution

To begin installation with a Kuali Rice provided distribution, just uncompress the software you retrieved from download location at the Rice web site. For the purposes of this guide, we will be focusing on installation of the server distribution.

Uncompress the server distribution downloaded from the Rice site:

1. Verify that you are logged in as the **root** user

2. Change directory to where the distributions are located

   ```
   # cd /opt/software/distribution
   ```

3. Uncompress the distribution

   a. Follow the steps below to uncompress the Binary distribution:

      ```
      # mkdir binary
      # unzip rice-1.0.3-bin.zip -d binary
      # chmod -R 777 /opt/software
      ```

   b. Follow the steps below to uncompress the Server distribution:

      ```
      # mkdir server
      # unzip rice-1.0.3-server.zip -d server
      # chmod -R 777 /opt/software
      ```

OS Setup

It is a best practice to modify your system parameters as described here. These are just suggested system parameters. Use your best judgment about how to set up your system.

- Verify that you are starting out in the **/etc** directory:

  ```
  computername $ pwd
  /etc
  ```

- Add this to **/etc/profile** to setup ant and maven first in the path:

  ```
  JAVA_HOME=/usr/java/jdk1.6.0_16
  ```

```
CATALINA_OPTS="-Xmx512m -XX:MaxPermSize=256m"
ANT_HOME=/usr/local/ant
ANT_OPTS="-Xmx1512m -XX:MaxPermSize=192m"
MAVEN_HOME=/usr/local/maven
MAVEN_OPTS="-Xmx768m -XX:MaxPermSize=128m"
GROOVY_HOME=/usr/local/groovy

PATH=$JAVA_HOME/bin:$ANT_HOME/bin:$MAVEN_HOME/bin:$GROOVY_HOME/bin:$PATH
export JAVA_HOME CATALINA_OPTS ANT_HOME ANT_OPTS MAVEN_OPT GROOVY_HOME
```

- After you have completed your update of the /etc/profile file, please execute this command to immediately update your profile with the changes from above:

  ```
  source /etc/profile
  ```

Install the JDBC Drivers

Warning

You must install the JDBC (Java Database Connectivity) driver for MySQL and Oracle to satisfy the requirement for the structure of the Ant target in the build process.

Rice software database drivers directory defaults

/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.

MySQL JDBC Driver

Kuali Rice uses the MySQL Connector/J product as the native JDBC driver.

You can obtain the driver from the MySQL website Once you have downloaded the JDBC driver that corresponds to your version of MySQL, copy it to **/java/drivers**.

Oracle JDBC driver

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

You must download the Oracle JDBC from the Oracle website directly due to licensing restrictions. Please refer to the Sources for Required Software section in this Guide to find the download location for this software. Once you have downloaded the JDBC driver, copy it to **/java/drivers**.

Set Up the ImpEx Process to Build the Database

Overview

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.

Obtaining the ImpEx Tool

The ImpEx tool is included in the Rice binary and server distributions. It is located in the **database/database-impex** directory of the relevant archive. If you are building Rice from source, the ImpEx tool can be acquired from Subversion:

```
svn co https://test.kuali.org/svn/kul-cfg-dbs/trunk
```

Oracle ImpEx Preconfiguration Setup

For Oracle users, you must run two scripts on your Oracle database before continuing on to the ImpEx development database ingestion process. Please contact your DBA and have him or her run these two setup scripts as the **Oracle SYS user**. These scripts will be in the **database/database-impex** distribution (or the impex subdirectory if you checked out the tool from svn). These are the commands, and they must be run in this order:

1. **create_kul_developer.sql** (Creates the **KUL_DEVELOPER** role and applies grants)
2. **system_grants.sql** (Creates the **KULUSERMAINT** user for Rice Oracle system maintenance)

Note

Please inform your DBA that the above scripts must be run under the Oracle SYS account.

After your DBA runs the above scripts, you must run two more scripts as the kulusermaint:

1. **create_admin_user.sql** (Creates the **kuluser_admin** user)
2. **kuluser_maint_pk.sql** (Creates the package containing the functions that the Rice system uses to do Oracle system maintenance)

ImpEx Configuration Overview

The Rice team released four different sets of data with Rice 1.0.3. The server distribution contains only the first two listed, whereas the binary distribution contains all four. You can find these datasets in the /database directory of the relevant archive:

1. **bootstrap-server-dataset** – This is the core dataset. The Rice standalone server cannot function properly without this data.
2. **demo-server-dataset** – This is a superset of the bootstrap-server-dataset which, in addition to necessary bootstrap data, includes sample security groups and workflow documents.
3. **bootstrap-client-dataset** – This dataset is necessary for creating a Rice client application
4. **demo-client-dataset** – This is a superset of the data in the bootstrap-client-dataset. In addition to necessary bootstrap data, it includes tables used by some of the Rice sample client applications.

This guide does not deal with developing a client application; for more information on creating a Rice client application and the use of the client datasets, see the Global Technical Reference Guide.

impex-build.properties Reference

After you have uncompressed the distribution (Binary or Server) you will use for Rice, login as the user that you use to run the Tomcat server:

1. 1. Go to the sub-directory, **<directory where uncompressed>/database/database-impex**. This directory is the same for each of the distributions.
2. Copy the **impex-build.properties.sample** file to your home directory, renaming the file to **impex-build.properties**.

Specific impex-build.properties database parameter setup

For MySQL Users

1. Configure the **drivers.directory**, the directory where the MySQL and Oracle JDBC drivers are located.
2. Configure **import.torque.database.***. Set the user and password to the user under which you want to run the Rice software.
3. Configure **import.admin.user** and **import.admin.password**.

For Local MySQL Server

Set import.admin.url as follows:

```
import.admin.url=jdbc:mysql://localMySQLServerComputerName:MySQL-port-number/
```

Example: **import.admin.url=jdbc:mysql://localhost:3306/**

For Remote MySQL Server

Set import.admin.url as follows:

```
import.admin.url=jdbc:mysql://remoteMySQLServerComputerName:MySQL-port-number/
```

Example: **import.admin.url=jdbc:mysql://192.168.25.22:3306/**

To setup the database from one of the distributions, change directories to /database/database-impex and use these commands:

```
ant create-schema
ant import
```

For Oracle users

1. Configure the **drivers.directory**, the directory where the MySQL and Oracle JDBC drivers are located.
2. Configure **import.torque.database.*** for the Oracle setup. Set the user and password to the user under which you want to run the Rice software.
3. Configure **import.admin.user** and **import.admin.password**.

   ```
   import.admin.url= ${import.torque.database.url}
   ```
4. Configure **oracle.usermaint.***
5. To setup the database from one of the distributions, use these Ant commands:

   ```
   ant create-schema
   ```

```
ant import
```

Verifying your Database Installation

At this point, your Kuali 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).

Installing Kuali Rice 1.0.3 Standalone Server

UNIX-Like Operating Environments

Setting Up Apache Tomcat

Once you have downloaded Tomcat , please install it using these steps:

- Log in as root
- Copy or download the file **apache-tomcat-5.5.27.zip** to **/opt/software/tomcat** using this code:

```
cd /opt/software/Tomcat
unzip apache-tomcat-5.5.27.zip -d /usr/local
ln -s /usr/local/apache-tomcat-5.5.27 /usr/local/tomcat
chown -R rice:rice /usr/local/apache-tomcat-5.5.27
cd /usr/local/tomcat/bin
chmod -R 755 *.sh
su - rice
cd /usr/local/tomcat/bin
./startup.sh
```

- You should see something like this:

```
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
```

Now that Tomcat is running:

1. To test that Tomcat came up successfully, try to browse to **http:// yourlocalip:8080**.
2. If you successfully browsed to http://yourlocalip:8080, then execute this command as the rice user:

```
./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
```

Warning

At times, Tomcat can have a session related problem with OJB where if you stop the server it won't start again. This can be fixed by deleting the SESSIONS.ser file in the %TOMCAT_HOME %/work/Catalina/%host name%/%webapp% directory

This can be prevented by setting the saveOnRestart property to be false in the web application's context.xml file as documented here: http://tomcat.apache.org/tomcat-5.5-doc/config/manager.html.

Setting Up the Rice Application

Deploying the WAR file

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 Apache Tomcat 5.5.x, this is **[Tomcat-root-directory]/webapps**.

Setting Up the Database JDBC Drivers for Tomcat

Copy the database-specific JDBC driver to the [Tomcat-root-directory]/common/lib.

- Login as the root user:

```
su – rice
```

- MySQL

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

- Oracle

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

Configuring 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:

1. /usr/local/rice/rice-config.xml
2. ${rice.base}../../../conf/rice-config.xml
3. ${rice.base}../../conf/rice-config.xml
4. ${additional.config.locations}

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

1. ServletContext.getRealPath("/")
2. **catalina.base** system property
3. The current working directory

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

To get the **rice-config.xml** and other Rice configuration files into their correct locations, follow these steps:

- Login as root:

```
mkdir /usr/local/rice
chown rice:rice /usr/local/rice
chmod 755 /usr/local/rice
cd /opt/software/kuali/src/rice-release-1-0-2-br/web/src/main/config/example-con
cp -p rice-config.xml /usr/local/rice
cp -p log4j.properties /usr/local/rice
cd /usr/local/rice
chown rice:rice log4j.properties
chown rice:rice rice-config.xml
su - rice
cd /usr/local/rice
```

  Modify the database parameters in the **rice-config.xml** file. The values should conform to the values you used with the ImpEx tool (listed in the **impex-build.properties** file).

```
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
```

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

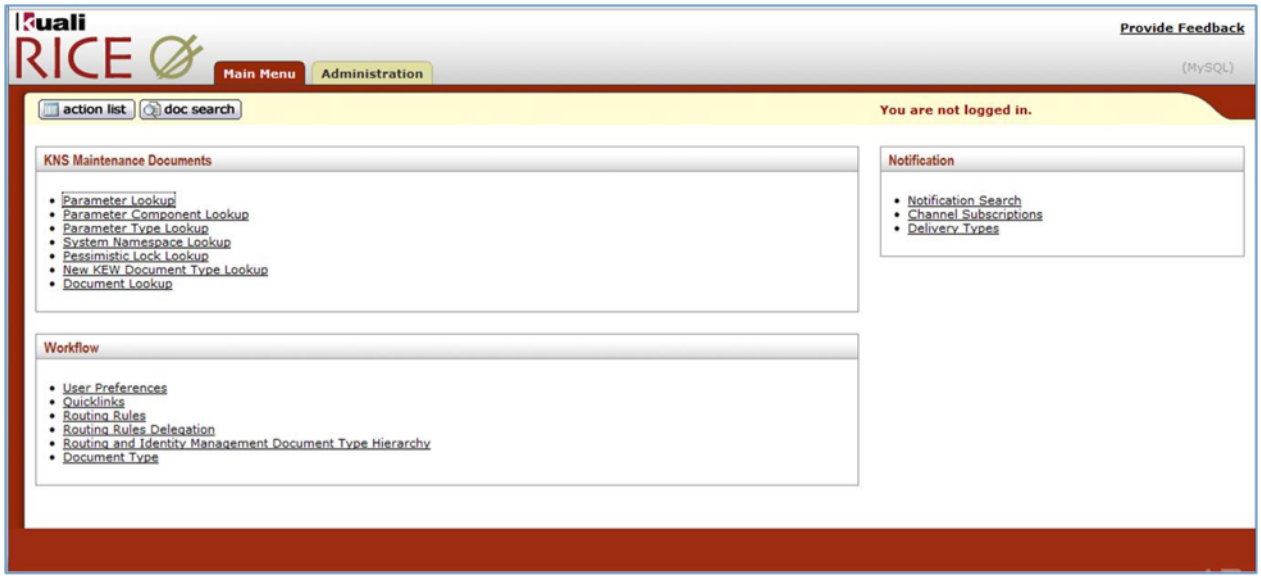
```
cd /usr/local/tomcat/bin
./startup.sh
```

- Check if Tomcat and Rice started successfully:

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

  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 4.1. 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 4.3. 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 • 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/rice/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	/usr/local/rice/rice-config-${environment}.xml

Parameter	Description	Examples or Values
	parameters (This param is not in the default config.)	
sample.enabled	Enable the sample application	boolean

Table 4.4. Database

Parameter	Description	Examples or Values
datasource.ojb.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.DerbyPlatform • org.kuali.rice.core.database.platform.OraclePlatform • org.kuali.rice.core.database.platform.MySQLPlatform
datasource.driver.name	JDBC driver for the database	• org.apache.derby.jdbc.EmbeddedDriver, • oracle.jdbc.driver.OracleDriver • com.mysql.jdbc.Driver
datasource.username	User name for connecting to the server database	rice
datasource.password	Password for connecting to the server database	
datasource.url	JDBC URL of database to connect to	• 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.validationQuery	Query to validate connections from the database	select 1 from dual

Table 4.5. 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

Parameter	Description	Examples or Values
keystore.password	Password to access the keystore and the server's key	see section entitled Generating the Keystore

Table 4.6. KEN

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

Table 4.7. 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.)	

Chapter 5. Generating the 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. 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**

   ### Note

   r1c3pw is the password used for the example provided.

2. The keystore is also located in the server distribution under the security directory.

   ### Note

   keypass and storepass should be the same. r1c3pw is the password used for the example provided

3. You can generate the keystore yourself. Please refer to the KSB 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 6. Tuning Kuali Rice 1.0.3

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"
```

Appendix A. Installing the Database Management Systems

Kuali Rice was developed using two relational database management systems:

- MySQL
- Oracle

MySQL Database Preparation

Install the MySQL database management system.

Steps to Install the Standalone Rice Platform

If you are installing MySQL after the initial operating system installation, use the RHEL update manager or yum on CentOS and install these packages:

- mysql
- mysql-server

If you did not install MySQL with the distribution, execute this command line (this assumes that you installed a CentOS 5.3 distribution):

- Check if MySQL is installed:

```
rpm -qa | grep mysql
```

- If the command has the following text in the results, then go down to the step where you check if MySQL is set to start at the appropriate run-levels:

```
mysql-server-5.1.xx-x.el5
mysql-5.1.xx-x.el5
```

- If the command returns no results, no MySQL packages are installed. In that case, do this:

```
yum -y install mysql
yum -y install mysql-server
```

- Next check that the MySQL server is set to start at the appropriate run-levels:

```
chkconfig --list | grep mysqld
mysqld          0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

- If the word “on” does not appear after the 3, 4, and 5, the MySQL server is set to be started manually. To set the MySQL server to start at the appropriate run-levels, execute this:

```
chkconfig --level 345 mysqld on
```

- Now, double check the run-levels for the MySQL server:

```
chkconfig --list | grep mysqld
mysqld          0:off   1:off   2:off   3:on   4:on   5:on   6:off
```

- Check if the MySQL server has been started automatically:

```
ps -ef | grep mysql
```

- If you get the following output:

```
root 4829 3577 0 22:57 pts/1 00:00:00 grep mysql
```

- Then, start the MySQL daemon with the following command:

```
/etc/init.d/mysqld start
```

- You should see results similar to this:

```
Initializing MySQL database:  Installing MySQL system tables...
OK
Filling help tables...
OK

To start mysqld at boot time you have to copy
support-files/mysql.server to the right place for your system

PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
To do so, start the server, then issue the following commands:
/usr/bin/mysqladmin -u root password 'new-password'
/usr/bin/mysqladmin -u root -h krice password 'new-password'
See the manual for more instructions.
You can start the MySQL daemon with:
cd /usr ; /usr/bin/mysqld_safe &

You can test the MySQL daemon with mysql-test-run.pl
cd mysql-test ; perl mysql-test-run.pl

Please report any problems with the /usr/bin/mysqlbug script!

The latest information about MySQL is available on the web at
http://www.mysql.com
Support MySQL by buying support/licenses at http://shop.mysql.com
                                                [  OK  ]
Starting MySQL:                                 [  OK  ]
```

Steps to Install the Production Platform and Remote MySQL Server

These instructions assume that this is a fresh installation of Linux and NO MySQL server has been installed on the computer. Use the RHEL update manager or yum on CentOS and install this package:

- mysql
- mysql-server

If you did not install MySQL with the distribution, execute this command line (this assumes that you installed a CentOS 5.3 distribution):

- Check if MySQL is installed:

```
rpm -qa | grep mysql
```

- If the command has the following text in the results, then go down to the step where you check if MySQL is set to start at the appropriate run-levels:

```
mysql-server-5.1.xx-x.el5
mysql-5.1.xx-x.el5
```

- If the command returns no results, no MySQL packages are installed. In that case, do this:

```
yum -y install mysql
yum -y install mysql-server
```

- Next check that the MySQL server is set to start at the appropriate run-levels:

```
chkconfig --list | grep mysqld
mysqld          0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

- If the word “on” does not appear after the 3, 4, and 5, the MySQL server is set to be started manually. To set the MySQL server to start at the appropriate run-levels, execute this:

```
chkconfig --level 345 mysqld on
```

- Now, double check the run-levels for the MySQL server:

```
chkconfig --list | grep mysqld
mysqld          0:off   1:off   2:off   3:on   4:on   5:on   6:off
```

- Check if the MySQL server has been started automatically:

```
ps -ef | grep mysql
```

- If you get the following output:

```
root 4829 3577 0 22:57 pts/1 00:00:00 grep mysql
```

- Then, start the MySQL daemon with the following command:

```
/etc/init.d/mysqld start
```

- You should see results similar to this:

```
Initializing MySQL database:  Installing MySQL system tables...
OK
Filling help tables...
OK
```

To start mysqld at boot time you have to copy **support-files/mysql.server** to the right place for your system.

Warning

Remember to set a password for the MySQL root User!

To do so, start the server and then issue the following commands:

```
/usr/bin/mysqladmin -u root password 'new-password'
/usr/bin/mysqladmin -u root -h Rice password 'new-password'
```

```
See the manual for more instructions.
You can start the MySQL daemon with:
cd /usr ; /usr/bin/mysqld_safe &

You can test the MySQL daemon with mysql-test-run.pl
cd mysql-test ; perl mysql-test-run.pl

Please report any problems with the /usr/bin/mysqlbug script!

The latest information about MySQL is available on the web at
http://www.mysql.com
Support MySQL by buying support/licenses at http://shop.mysql.com
                                          [  OK  ]
Starting MySQL:                           [  OK  ]
```

This completes the MySQL server installation. Next, you install the MySQL client on the Rice computer.

MySQL Client Installation: For Production Platform and Remote MySQL Server

Log onto your MySQL/Rice computer as the root user. If you are installing MySQL after the initial operating system installation, use the RHEL update manager or yum on CentOS and install the MySQL package.

If you did not install MySQL with the distribution, execute this in the command line (this assumes that you installed a CentOS 5.3 distribution):

- Check if MySQL is installed:

  ```
  rpm –qa | grep mysql
  ```

- If the command has this text in the results, then go to the second step after this:

  ```
  mysql-5.1.xx-x.el5
  ```

- If the command returns no results, no MySQL packages are installed. In that case, do this:

  ```
  yum -y install mysql
  ```

- → Now, edit the **/etc/hosts** file and enter the IP address and the host name of the computer where the MySQL server is located. In the following example, kmysql is name of the MySQL server:

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

MySQL Standalone and Production Platforms

Once you have installed MySQL, ensure that MySQL is running by performing this on the computer where the MySQL server is running:

```
mysqladmin -u root -p version
Enter password: [hit return, enter nothing]
```

Warning

If you see the following text, your MySQL server is NOT running:

```
mysqladmin: connect to server at ‘computername’ failed
error: ‘Can’t connect to local MySQL server through socket ‘/var/lib/mysql/mysq
Check that mysqld is running and that the socket ‘/var/lib/mysql/mysql.sock’ ex
```

If your MySQL server is NOT running, execute this on the computer as root where the MySQL server is installed:

```
/etc/init.d/mysqld start
```

If you see something similar to this, your MySQL server is running:

```
mysqladmin  Ver 8.41 Distrib 5.0.45, for redhat-linux-gnu on i686
Copyright (C) 2000-2006 MySQL AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license

Server version  5.0.45
Protocol version  10
Connection   Localhost via UNIX socket
UNIX socket   /var/lib/mysql/mysql.sock
Uptime:    34 sec

Threads: 1  Questions: 1  Slow queries: 0  Opens: 12  Flush tables: 1  Open tables
```

If this is a new MySQL install, you have to set the initial password. To do so after you have started the MySQL daemon, execute this command:

```
mysqladmin –u root password ‘new-password’
```

After initial installation, you must set the MySQL root password. This applies to all platforms: Standalone and Developer Platform and Production Platform. To set the MySQL root password, do this on the computer where the MySQL server is running (substitute the name of the machine for ‘computername’):

```
mysqladmin -u root password ‘kualirice’
mysql -u root --password=”kualirice”
mysql> use mysql
mysql> set password for ‘root’@’localhost’=password(‘kualirice’);
mysql> set password for ‘root’@’127.0.0.1’=password(‘kualirice’);
mysql> set password for ‘root’@’computername’=password(‘kualirice’);
mysql>
mysql> grant all on *.* to ‘root’@’localhost’ with grant option;
mysql> grant all on *.* to ‘root’@’127.0.0.1’ with grant option;
mysql> grant all on *.* to ‘root’@’computername’ with grant option;
mysql>
mysql> grant create user on *.* to ‘root’@’localhost’ with grant option;
mysql> grant create user on *.* to ‘root’@’127.0.0.1’ with grant option;
mysql> grant create user on *.* to ‘root’@’computername’ with grant option;
mysql>
mysql> quit
```

If your MySQL server is running remotely (this is usually true for the Production Platform), do this on the computer where the MySQL server is running:

```
mysql -u root --password=”kualirice”
mysql> use mysql
mysql> create user ‘root’@’client_computername’;
mysql>
```

```
mysql> set password for 'root'@'client_computername'=password('kualirice');
mysql>
mysql> grant all on *.* to 'root'@'client_computername' with grant option;
mysql>
mysql> grant create user on *.* to 'root'@'client_computername' with grant option;
mysql>
mysql> quit
```

Setting Up MySQL Configuration Parameters

1. When you install MySQL, you must decide what port number the database will use for communications. The default port is 3306. If you decide to have MySQL communicate on a port other than the default, please refer to the MySQL documentation to determine how to change the port. These instructions assume for Quick Start Recommended Best Practice sections that the MySQL communications port remains at the default value of 3306.

2. Please ensure your MySQL server has the following settings at a minimum. If not, set them and then restart your MySQL daemon after modifying the configuration file, **my.cnf**.

MySQL Configuration Parameters

Note

These settings reflect a CentOS 5.3 distribution using the distribution-installed MySQL packages. They are stored in **/etc/my.cnf**.

Only the root user can change these settings, and they should only be changed when the database server is not running.

```
[mysqld]
max_allowed_packet=20M
transaction-isolation=READ-COMMITTED
lower_case_table_names=1
max_connections=1000
innodb_locks_unsafe_for_binlog=1
```

Transaction Isolation

Warning

It is very important to verify that the default transaction isolation is set to READ-COMMITTED. KEW uses some 'SELECT ... FOR UPDATE' statements that do NOT function properly with the default MySQL isolation of REPEATABLE-READ.

Index Gaps Lock

innodb_locks_unsafe_for_binlog=1 is only necessary if you are running MySQL 5.0.x. This behavior has been changed in MySQL 5.1+ so that, in 5.1+, this command is NOT necessary as long as you specify READ-COMMITTED transaction isolation.

3. If you make the changes to your MySQL configuration specified above, you will have to restart your MySQL server for these changes to take effect. You can restart your MySQL daemon by executing this command:

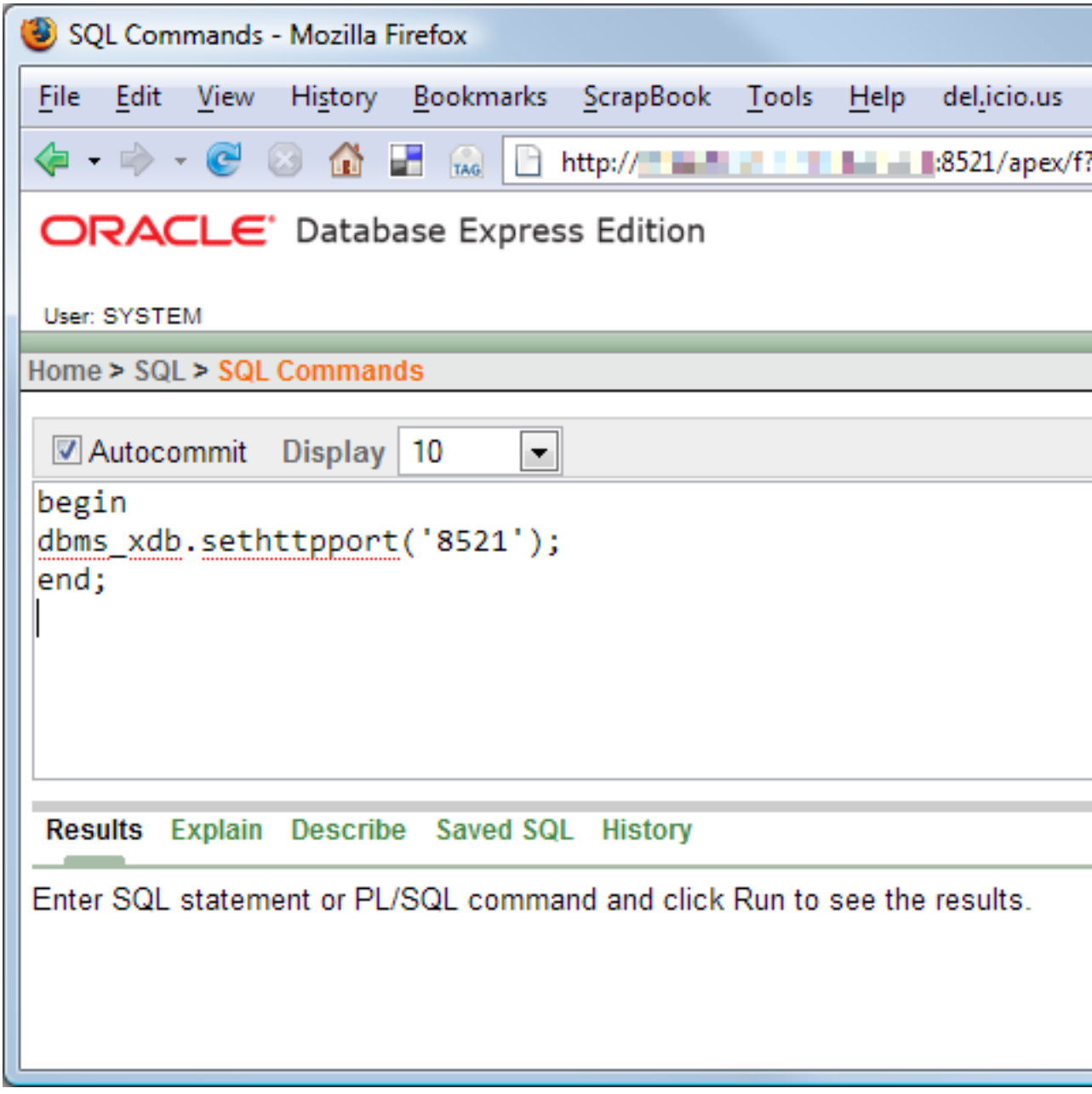
```
/etc/init.d/mysqld restart
```

Oracle Database Preparation

1. *Optional*: To run the database completely on your local machine, we recommend installing Oracle 10g Express (XE). Please refer to the Sources for Required Software section of this Installation Guide to find the download location for this software.

   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 A.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

2. Optional: To connect to the supporting Oracle database (i.e., run scripts, view database tables, etc.), we recommend installing the Squirrel SQL client. Please refer to the Sources for Required Software section to find the download location for this software.

   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 SQuirreL, you do this in Session->Session Properties->SQL->Statement Separator.

3. 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
   ```

Appendix B. 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

Appendix C. Building Rice from Source

Installing Java

Warning

When you install Java on the server to run Kuali Rice 2.0.0-b3, please make a note of the installation directory. You must have this information to configure the other Rice products.

Note

Version 1.0.0 of Rice was compiled with Java 5 and maintained source code compatibility with Java 5 as well. Starting with the 1.0.1.1 release, Rice is now being compiled using Java 6, although the source code still remains compatible with Java 5. It is recommended that client applications begin using Java 6 for compiling and running Rice.

1. In a prior step, you should have uploaded the Java installation file, **jdk-1_5_0_18-linux-i586-rpm.bin**, to the directory, **/opt/software/java**. Change your current directory to that directory.

   ```
   cd /opt/software/java
   ```

2. Change the file to have executable attributes.

   ```
   chmod 777 *
   ```

3. Run the file with this command:

   ```
   ./jdk-1_5_0_18-linux-i586-rpm.bin
   ```

This puts your Java JDK software in **/usr/java/jdk1.6.0_16/**.

Note

This directory is used throughout the rest of the Quick Start Recommended Best Practices sections in this Installation Guide as the Java home directory

Warning

Using a default IBM JDK will result in the failure of Rice to startup do to incompatibilities between cryptography packages. In order to make the IBM JDK work with Rice, the bcprov-*.jar needs to be removed from the classpath.

Install Software Tools

Note

This step should not be necessary for binary and server distributions.

Install Apache Ant and Maven

- Rice is sensitive to the versions of Ant and Maven that you use.
- Please check the requirements for Rice and then verify, install, and use the required versions.

- These software packages can be installed in any directory, as long as their bin directories are specified in the path first.

Install Ant and Maven to these directories:

1. Ant: **/usr/local/apache-ant-1.7.1**

   a. Change your current directory to the directory where the Ant software zip is located, **/opt/software/ant**.

   b. Uncompress the Ant zip file.

   c. Create a symbolic link to **/usr/local/ apache-ant-1.7.1** in **/usr/local**.

   For example:

   ```
   cd /opt/software/ant
   tar xvfz apache-ant-1.7.1-bin.tar.gz -C /usr/local
   ln -s /usr/local/apache-ant-1.7.1 /usr/local/ant
   ```

2. Maven: **/usr/local/maven**

   a. Change your current directory to the directory where the Maven software zip is located, **/opt/software/maven**.

   b. Uncompress the Maven zip file.

   c. Create a symbolic link to **/usr/local/apache-maven-2.0.9** in **/usr/local**.

   For example:

   ```
   cd /opt/software/maven
   tar xvfz apache-maven-2.0.9-bin.tar.gz -C /usr/local
   ln -s /usr/local/apache-maven-2.0.9 /usr/local/maven
   ```

Source Code Retrieval From Subversion

Please verify that you have installed the subversion client. As long as your Linux distribution has a subversion client that meets the required version for Rice, this should be sufficient. Please refer to the Sources for Required Software section of this Installation Guide for the download location and the required subversion client software version.

You can verify that you have the subversion client with this command:

```
rpm -qa | grep sub
```

If you have the subversion client installed, you will see something similar to this:

```
subversion-1.4.2-4.el5
```

Configuring the ImpEx Tool from the Rice Subversion Repository

After you have downloaded the ImpEx tool from the source code repository, log in as the non-authoritative user that you use to run the Tomcat server:

1. Go to the sub-directory, **<downloaded dir>/trunk/impex**, from which you downloaded the **kul-cfg-dbs** module.

2. Copy the **impex-build.properties.sample** file to your home directory, renaming the file to **impex-build.properties**.

3. Then, configure the file, **impex-build.properties**, for the Ant utilities to set up the database.

Database location setup

MySQL users

For retrieving the most current Rice database, verify that these parameters are set to:

```
svnroot=https://test.kuali.org/svn/
svn.module =rice-cfg-dbs
svn.base= branches/rice-release-1-0-2-br

torque.schema.dir=<root or where you wish to download dev dbs>/${svn.module}
torque.sql.dir=${torque.schema.dir}/sql

# then, to overlay a KFS/KC/KS database on the base rice database, use the paramet
# If these parameters are commented out, the impex process will only use the infor
#svnroot.2=https://test.kuali.org/svn/
#svn.module.2=kfs-cfg-dbs
#svn.base.2=trunk
#torque.schema.dir.2=../../${svn.module.2}
#torque.sql.dir.2=${torque.schema.dir.2}/sql
```

and verify that the second set of svn parameters (svnroot.2, svn.module.2) are commented out or deleted for a Rice only/non-Kuali Financials, Kuali Student or Kuali Coeus installation.

Oracle Users

For retrieving the most current Rice database, verify that these parameters are set to:

```
svnroot=https://test.kuali.org/svn/
svn.module =rice-cfg-dbs
svn.base= branches/rice-release-1-0-2-br

torque.schema.dir=<root or where you wish to download dev dbs>/${svn.module}
torque.sql.dir=${torque.schema.dir}/sql

# then, to overlay a KFS/KC/KS database on the base rice database, use the paramet
# If these parameters are commented out, the impex process will only use the infor
#svnroot.2=https://test.kuali.org/svn/
#svn.module.2=kfs-cfg-dbs
#svn.base.2=trunk
#torque.schema.dir.2=../../${svn.module.2}
#torque.sql.dir.2=${torque.schema.dir.2}/sql
```

Verify that the second set of svn parameters (svnroot.2, svn.module.2) are commented out or deleted for a Rice only/non-Kuali Financials, Kuali Student or Kuali Coeus installation.

Compiling the Source Code

Tool Requirements:

1. Ant must be installed. (It should be already.)
2. Maven must be installed. (It should be already.)
3. Subversion must be installed.

Compilation steps:

1. Retrieve the source code from Kuali: Download Kuali source zip distribution from the Kuali website or retrieve the source code from the Subversion repository. You can find both of these in the Recommended Software Sources section.
2. To begin working with the source code:
   a. To work with one of the Kuali distributions: i. Unzip the software into **/opt/software/kuali/src** OR
   b. Check out the source code from the Subversion repository.
      - To retrieve and compile the source code from the source code repository, log in as root and enter this:

```
cd /opt/software/kuali
mkdir src
cd src
svn co https://test.kuali.org/svn/rice/branches/rice-release-1-0-2-br/
cd ..
pwd
/opt/software/kuali
chmod -R 777 /opt/software/kuali/src
su - rice
cd ~rice # JUST TO VERIFY YOU ARE IN RICE'S HOME DIRECTORY
```

      - To compile the source code from the Binary distribution, first uncompress the software:

```
cd /opt/software/distribution/
mkdir src
unzip rice-1.0.3-src.zip -d src
chmod -R 777 /opt/software

su - rice
cd ~rice # JUST TO VERIFY YOU ARE IN RICE'S HOME DIRECTORY
```

   c. Create a file with VI named **kuali-build.properties**. This file should be in the root directory of the non-authoritative user that runs the Tomcat server.

      The contents should be:

```
maven.home.directory=/root/of/maven/installation…
maven.home.directory=/usr/local/maven
```

   d. Change directory:

i. For the source code from the Subversion repository:

```
cd /opt/software/kuali/src/rice-release-1-0-2-br
```

ii. For the source code from the Binary distribution:

```
cd /opt/software/distribution/src
```

e. Install the Oracle JDBC driver into the Maven repository.

Warning

You must run the Ant command to install the Oracle JDBC into the Maven repository from the root of the source code directory. In the Quick Start Recommended Best Practices sections, use these directories:

- For the source code repository, checkout: /opt/software/kuali/src/rice-release-1-0-2-br
- For the source code distribution: /opt/software/distribution/src

An Apache Ant script called install-oracle-jar installs this dependency; however, due to licensing restrictions you must download the driver from the Oracle website. Please refer to the Sources for Required Software section to find the download location for this software.

Once you have downloaded the JDBC driver, copy it to the /java/drivers directory. The Apache Ant script, located in the source code directory, will look for ojdbc14.jar in {java root}/drivers and install the necessary file.

To install the Oracle JDBC driver in the Maven repository, run this command:

```
ant install-oracle-jar
```

Default directory - **/java/drivers** – Ant looks there as a default. This can be overridden by executing this command:

```
ant -Ddrivers.directory=/my/better/directory install-oracle-jar
```

f. To build the WAR file from source, enter:

```
ant dist-war
```

g. At this point, you have built the WAR file. It is in a subdirectory called target. The WAR file is named **rice-dev.war**.

To verify that the WAR file was built:

```
cd /opt/software/kuali/src/rice-release-1-0-2-br/target
ls -la
total 44972
drwxrwxr-x  3 rice rice     4096 Jun  1 00:11 .
drwxrwxrwx 19 root root     4096 May 31 23:59 ..
drwxrwxr-x  3 rice rice     4096 Jun  1 00:11 ant-build
-rw-rw-r--  1 rice rice 45987663 Jun  1 00:11 kr-dev.war
```

h. Copy the WAR file from the target subdirectory to the Tomcat webapps directory.

```
cp -p kr-dev.war /usr/local/tomcat/webapps/kualirice.war
```

Appendix D. Setting Up a Load-Balanced Clustered Production Environment

Note

These details are repeated from the previous section on running a productionplatform

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

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
```

   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.

Appendix E. 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}</p
```

- 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</par
```

- 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 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/fo
      [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/fo
      [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-${rice-prd-instance-na



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


   <param name="notification.basewebappurl">${application.url}/ken</param>
<param name="workflow.url">${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.ojb.platform">MySQL</param>

 <param name="datasource.platform">org.kuali.rice.core.database.platform.MySQLDat
 <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.size">5</param>

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



<!-- Oracle example

    <param name="datasource.ojb.platform">Oracle9i</param>
    <param name="datasource.platform">org.kuali.rice.core.database.platform.Orac
 <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.size">5</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.ojb.platform">MySQL</param>
 <param name="datasource.platform">org.kuali.rice.core.database.platform.MySQLDat
 <param name="datasource.url">jdbc:mysql://mysql:3306/riceprd2</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.size">5</param>
 <param name="datasource.pool.maxWait">10000</param>
 <param name="datasource.pool.validationQuery">select 1</param>

    <!-- Oracle example
    <param name="datasource.ojb.platform">Oracle9i</param>
    <param name="datasource.platform">org.kuali.rice.core.database.platform.Orac
 <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.size">5</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.

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: • 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 transaction into the processed state if neccessary. • 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: • 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: • 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 being 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.

	• 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 is 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: • 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) authenticaton implementation in Rice does not require a password as it is intended for testing puposes 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

The standard fields that are defined and collected for every *Routing Rule* These include:

- 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

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.

Blanket Approve Workgroup

A workgroup that has the authority to Blanket Approve a document.

Branch

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.

Business Rule

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

Identifies the different fiscal and physical operating entities of an institution.

Campus Type

Designates a campus as physical only, fiscal only or both.

Cancel

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.

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: • 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

1. An acronym for Educational Community License.
2. All Kuali 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 interfact 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 - Kuali Coeus	TODO
KCA - Kuali Commercial Affiliates	A designation provided to commercial affiliates who become part of the Kuali Partners Program to provide for-fee guidance, support, implementation, and integration services related to the Kuali software. Affiliates hold no ownership of Kuali intellectual property, but are full KPP participants. Affiliates may provide packaged versions of Kuali that provide value for installation or integration beyond the basic Kuali software. Affiliates may also offer other types of training, documentation, or hosting services.
KCB – Kuali Communications Broker	KCB is logically related to KEN. It handles dispatching messages based on user preferences (email, SMS, etc.).
KEN - Kuali 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 – Kuali Enterprise Workflow	Kuali 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 Kuali Enterprise Workflow to automate and regulate the approval process for the transactions or documents they create.
KFS – Kuali 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 – Kuali Identity Management	A Kuali Rice module, Kuali Identity Management provides a standard API for persons, groups, roles and permissions that can be implemented by an institution. It also provdies an out of the box reference implementation that allows for a university to use Kuali 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: • 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	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 Kuali Partners program.
Kuali Rice	Provides an enterprise-class middleware suite of integrated products that allow both Kuali and non-Kuali 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 “KUALI”. 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	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. In other words, an Entity is for identity while a Principal is for access management. 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.
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 Kuali 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	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. • 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	The status of a document in the course of its routing: • 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. No changes are allowed to a document that is in Final status.

- 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 it's 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	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*. Technical considerations for a Searchable Attributes are: • 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	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. Secondary Delegation is often configured for a range of dates and it must be registered in KEW or KIM to be in effect.
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	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. This field is equivalent to the "Subject" field in an email.

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 Kuali Enterprise Workflow (KEW). The Kuali 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 Ingester*. 1. An acronym for Extensible Markup Language. 2. Used for data import/export.
XML Ingester	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