Kuali Rice 2.0.0-b1-SNAPSHOT Release Notes

This document contains information on work completed
as part of the first beta release of Kuali Rice Version 2.0.

Released: 11-07-2011

Table of Contents

Overview 1
- Release Highlights 2
- Download 3
- Documentation 3
- Contact 3

Major Changes 3
- Middleware Version Compatibility 3
- Modularity 9
- PeopleFlows 9
- New Modules: KRMS and KRAD 10
- Configuration Changes 10
- Library Changes and Additions 10
- Other Changes 11

Rice 2.0 Work Remaining 12
- "Remote" and "Embedded" client application configurations 12
- Configurers 13
- Caching 13
- KEN and KCB Services 13
- Custom Action List Attributes 13
- Document Search 14
- createproject.groovy 14
- Documentation 14
- Upgrade Guides and Upgrade Scripts 14

Upgrade Guide 14
- Database Changes 14

Jiras 15

Overview

Welcome to the first beta release of Kuali Rice version 2.0!

This is the first major release of the Kuali Rice software since version 1.0 which was released in August of 2009. This release brings many new and exciting features and improvements to the Kuali Rice middlware and application development platforms. This document provides early information on the work that has been done as part of 2.0 development and how one can get started working with the software during the beta test phase of the project.

The primary architectural goal for this release of Rice was to position the platform for cross-version service compatibility moving forward. With the increase in the number of applications (both internal and external to Kuali) that are integrating with the Kuali Rice middleware platform, the ability to provide for

compatibility across different versions of Kuali Rice has risen to the forefront within the community as a critical need. The end result is intended to be a platform on which institutions can build applications without worrying about major forced upgrade paths for all integrated applications whenever they deploy a new release of Kuali Rice.

Another major effort that was undertaken in support of compatibility work was an effort to improve the modularity of the various components of Kuali Rice. In version 1.0.x of Rice, a high degree of coupling was present in numerous places. Work was done as part of this release to help improve upon that situation.

In addition to the architectural efforts supporting compatibility and modularity, Kuali Rice 2.0 sees the introduction of two new major modules.

The first of these is the Kuali Rapid Application Development (KRAD) module. KRAD is meant to provide the next generation of application development capabilities to those who are building applications based on Kuali software. KRAD provides a platform that can be utilized to build enterprise applications in a way that provides a more rich and flexible user experience than could be provided with the existing Kuali Nervous System (KNS). In fact, KRAD is intended to be the eventual replacement for the KNS which is the platform upon which much of the existing Kuali application software has been written. Note, however, that KNS is not being removed as of Rice 2.0. Work will continue on KRAD past the 2.0 release of Kuali Rice.

The final major piece of functionality that was undertaken for this release was the introduction of the Kuali Rule Management System (KRMS). KRMS is a business rule management system which can be used to author and maintain business rules that can be utilized during decision making processes within an application. This includes the ability to use business rules defined in KRMS to define rules for workflow routing, notification, validation, and other custom scenarios.

More details on each of the major efforts can be found in later sections of this document.

Release Highlights

Highlights of this release include:

- Refactoring of all remotely accessible services to position them for cross-version compatibility with future versions of Kuali Rice. This includes a migration from Java serialization-based services to SOAP-based web services.
- Reduction in the number of places where a Kuali Rice client application will interact directly with the Kuali Rice standalone server database (when using integration models like embedded KEW).
- Improvements to the Kuali Service Bus to allow it to support requirements for version compatibility.
- Introduction of a new approach to server and client-side caching use the Spring caching abstraction coupled with Ehcache. This includes a new user interface for managing caches.
- Breakdown of the codebase into smaller modules to increase modularity and flexibility of the software.
- Numerous improvements to the design and documentation of Kuali Rice apis.
- Integration of an out-of-the-box LDAP connector for Kuali Identity Management.
- Introduction of the Kuali Rules Management System (KRMS) module for creation and execution of business rules.
- Introduction of the concept of PeopleFlows in Kuali Enterprise Workflow.

- Introduction of the Kuali Rapid Application Development (KRAD) module, the next revision of the KNS which includes rich UI features and is based on Spring MVC instead of Struts.
- Ability to configure Bitronix as a JTA alternative to JOTM and XAPool.
- Many of the Kuali Rice library dependencies have been updated.

Download

Kuali Rice 2.0.0-b1-SNAPSHOT can be downloaded from the Rice website at http://kuali.org/rice/download/beta.

There are three different distributions of Rice available: source, binary and server. Please read the Installation Guide for more details on each of these distributions.

Documentation

JavaDocs can be found at http://site.kuali.org/rice/2.0.0-b1-SNAPSHOT/apidocs/index.html

For the beta test, we have a wiki space to collect and maintain for beta testers.

Formal documentation for this release is under review. Other than KRMS and KRAD Chapters, the documentation found at http://site.kuali.org/rice/2.0.0-b1-SNAPSHOT/reference/html/portal.html has not yet been updated for Rice 2.0.

Contact

If you encounter any difficulty, please don't hesitate to contact the Rice team on our public collaboration mailing list at `<rice.collab@kuali.org>`. Please indicate that you are using the 2.0.0-b1-SNAPSHOT version of Rice.

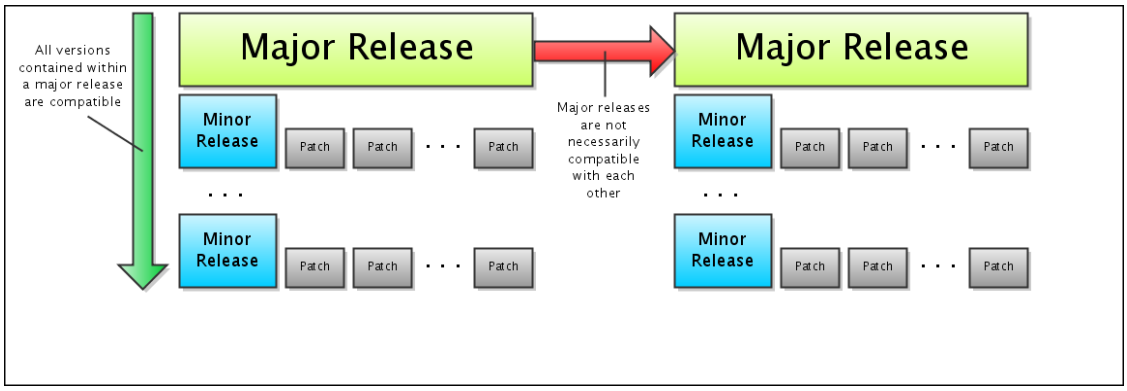
Major Changes

As mentioned in the overview and highlights, there were many major changes that were made in support of the goals of the Rice 2.0 release. During the implementation of modularity and version compatibility changes, there were many structural improvements and refactoring changes to Rice that are intended to help set a stronger foundation for future releases. Additionally, the KRMS and KRAD modules brought a large amount of new functionality to the Kuali Rice platform. These next sections dive into these changes in more detail.

Middleware Version Compatibility

In Rice 2.0, we are providing what we call "middleware" or "service" compatibility. This type of compatibility is mainly concerned with compatibility between a client application and the version of the Kuali Rice standalone server with which it is integrated. The basic requirement here is that if the Kuali Rice standalone server (which hosts all of the Kuali Rice services) is upgraded to a new version, that should not force any applications that integrate with those services to also upgrade (unless the server has been upgraded to a new "major" version such as Rice 3.0, 4.0, etc.). It should be able to continue to interoperate with the new set of services without having to pull in new Kuali Rice client libraries or (in the case of manual web-service integration) it should not have to modify the way in which it invokes and handles remote service calls.

The image below shows what the target is for version compatibility from the perspective of major, minor, and patch releases of Kuali Rice.



The type of compatibility being implemented for version 2.0 is in contrast to what we call "framework" compatibility, or the ability to upgrade a client application to a new version of the Kuali Rice client libraries or development framework without having to make any forced changes to that client application. Those who use Java are familiar with this type of compatibility because that is the level of compatibility that is generally provided between releases of the JDK (this is sometimes called a "drop-in" replacement). Working toward a goal of framework compatibility is on the roadmap for the Kuali Rice project, however the Rice 2.0 release makes no guarantees that it will provide such a level of compatibility moving forward.

More info on this and also Framework Version Compatibility can be found in our Kuali Rice Version Compatibility Statement.

The following sub-sections will delve into some specific work that was done in support of version compatibility.

Migration to SOAP web services

In Kuali Rice 1.0.x, services based on Java serialization were the primary services used for integration between Kuali Rice and client applications. The mechanism by which serialization was performed used Spring's HTTP invoker remoting framework. Java serialization is convenient for Java-to-Java application interaction. However, a SOAP web service approach allows for a greater level of interoperability between applications, including those not written in Java. Additionally, serialization can sometimes inherently cause compatibility issues when attempting to communicate between applications which might be running different versions of Java or different versions of libraries.

To this end, the services in Rice 2.0 have all been reimplemented as SOAP-based web services. There will be no Java-serialization based service distributed with Kuali Rice 2.0. However, the Kuali Service Bus will still support publishing of services that use Java. As part of this effort, the team has taken care to carefully design the message formats such that:

- The "required" portions of the message format are well-defined
- The proper format of data in the messages in well-defined
- Error and exception cases are well-defined
- The services are designed to allow for safe evolution over time
- The services are namespaced and named appropriately
- The services are versioned appropriately
- There is consistency in how services are designed and documented

Much of the above information can be found in the WSDL's which are generated for each of the services. The WSDL defines the message formats, operations, and error conditions of the numerous Kuali Rice services. This effort of service refactoring is at the heart of what we are calling "version compatibility" for the Kuali Rice 2.0 release.

Note that while SOAP is being used as the primary way in which services are exposed in Rice 2.0, there will likely be work in the near future to make services available using the REST architectural style.

Kuali Service Bus

In order to support requirements for version compatibility, there were numerous changes that needed to be made to the Kuali Service Bus. This ultimately resulted in a reworking of much of the internals of the KSB (with the exception of it's messaging functionality). The major changes made to the KSB for Kuali Rice 2.0 were:

1. Client applications now connect to the service registry through a SOAP web service, rather than through direct connections to the KRSB_SVC_DEF_T and related tables.

2. There is a new API for the ServiceBus which is well documented and acts as the client application interface point with the KSB.

3. The practice of storing a java serialized form of the service configuration in the database was altered to store an XML-serialized service descriptor. This allows for more portability and greater access for non-Java applications to interact with the service bus.

4. Removed the faulty "dead service" indicator from the registry.

5. Added the concept of a service status to the registry.

6. Added service version number information to the registry.

7. Replaced the concept of the "service namespace" with the concept of an "application id".

8. Introduced the concept of an "instance id" which identifies an instance of an application. This allows the KSB to better handle applications that are clustered with multiple deployments all publishing the same set of services.

9. Much more flexibility in how a client application can configure and use the KSB. For example, a client application could just configure the KSB client to act as a publisher of a service, or it could just connect to the KSB as a consumer of services. As well as additional configuration permutations related to support for reliable messaging and "development" mode (where services are not actually published).

10. A general refactoring of the internal architecture of the KSB related to handle it handles the publishing lifecycle of services as well as synchronization with the shared service registry.

As can be seen from the list above, much work was done to the service bus for Kuali Rice version 2.0. For more details, read the following page https://wiki.kuali.org/x/T3dyEg

Parameter Service, Namespace Service, and Component Service

In Rice 1.0.x, parameters, namespaces, and components were part of the KNS module of Rice. Client applications would access tables in the Kuali Rice server database directory in order to read and write values from these tables. As part of the general effort to modularize the Rice platform, it was decided that these three concepts were core concepts since they were used outside of the pure application development domain of the KNS (parameters in particular are used all over the Kuali Rice codebase). To this end they

were moved to the newly established "core" module of Rice (see more information on modularity work in later sections).

Additionally, SOAP services were created for each of these to eliminate the need for a client application to directly communicate with the related database tables, instead using a service-oriented approach to interacting with these different information sources. This resulted in the creation of the following services:

- **org.kuali.rice.core.api.parameter.ParameterRepositoryService** - remote service which provides operations related to system parameters
- **org.kuali.rice.core.api.namespace.NamespaceService** - remote service which provides operations related to namespaces
- **org.kuali.rice.core.api.component.ComponentService** - remote service which provides operations related to components, as well as implementing a new model for publication of "derived" components

For working with parameters, there is still a ParameterService this is provided as part of the client-side API. This service is configured with the application id of the application and delegates calls down to the remote ParameterRepositoryService.

For the ComponentService, it provides a new mechanism for publication of "derived" components, which can include components that are derived from an application's data dictionary configuration. This is a replacement for the "getNonDatabaseComponents" method which was part of the RiceApplicationConfigurationService in Rice 1.0.x. It uses a push model in contrast to the pull model which was used in Rice 1.0.x.

More information about these individual services and their available operations can be found in the javadocs. Also see the following page for more details on the work that was done with these services, including a description of the new model used for publication of derived components: https://wiki.kuali.org/x/tg5yEg

RiceApplicationConfigurationService

In Rice 1.0.x, the RiceApplicationConfigurationService was a service published by all KNS applications which provided callbacks into the application from the Kuali Rice standalone server. This service was removed in Rice 2.0 for various reasons. The reasons for it's removal and information on it's effective replacements can be found on the following page: https://wiki.kuali.org/x/vA6tEg

General Purpose Lookup APIs

In Rice 1.0.x, may of the services (particularly the KIM services) had general-purpose lookup methods which took a `Map<String, String>`. In this map, the key was intended to represent the property name on the data element being looked up and the value was intended to be the value to search for. The value itself could contain various wild card characters and other operators to allow for the construction of more complex searches. However, there were a few inadequacies with this approach:

1. If an institution wanted to implement their own version of the service, they would effectively need to implement a complex interpreter which would take the incoming String value and determine how to implement the lookup according to the values passed in.
2. There was no way to preform cross-property "ors" with this model.
3. There was no support for "paging" of query results.
4. The contract itself tended to be weakly defined, resulting in service methods that would be hard to maintain from a service compatibility perspective.

To this end, some work was done to design a general purpose lookup API that could be applied in a standard fashion across the various Rice services which needed it. The following page describes more about the various possibilities for implementing this as well as the solution which was chosen: https://wiki.kuali.org/x/PR9yEg.

The implementation of this api can be found on the core-api module under the package `org.kuali.rice.core.api.criteria`.

Kuali Enterprise Workflow Services

The Kuali Enterprise Workflow services have been refactored for version compatibility in a similar fashion to the rest of the services in Rice. As part of this effort, a bit of a rearchitecture of the service layer and APIs for workflow were performed as well. The major changes made to the workflow services and apis are as follows:

- The `WorkflowDocument` class is now an interface and is created using `WorkflowDocumentFactory.createDocument` or `WorkflowDocumentFactory.loadDocument` instead of creating new instances through the `WorkflowDocument` constructor.
- `SimpleDocumentActionsWebService` was combined with `WorkflowDocumentActions` to create the new `WorkflowDocumentActionsService` remotable service.
- `WorkflowInfo` and `WorkflowUtility` was removed and replaced with the following services:
  - `WorkflowDocumentService`
  - `ActionListService`
  - `DocumentTypeService`
  - `NoteService`
  - `RuleService`
- The various KEW framework services were refactored to conform to service version compatibility standards. This includes services for document search customization, action list customization, workflow notes, document security, rule validation, and workflow rule attributes.
- The various KEW message queues were refactored to conform to service version compatibility standards. This includes queues for document routing, blanket approval, immediate email reminders, searchable attribute indexing, document requeuing, etc.
- In the KNS, both `KualiWorkflowInfo` and `KualiWorkflowDocument` were removed as it was deemed this duplicate level of abstraction was unnecessary. In the case of `KualiWorkflowDocument`, existing references were changed to just `WorkflowDocument`. In the case of `KualiWorkflowInfo`, code was changed to call one of the new KEW services that was added in Rice 2.0.

Some rough notes with more details on the work that was done in KEW can be found here: https://wiki.kuali.org/x/fwuREg

Document Search

One of the major KEW framework refactorings that was undertaken for Rice 2.0 was a reimplementation of the various components that support Document Search customization. Full documentation on these changes and how to utilize them is still in progress, but a summary is as follows:

- `SearchableAttribute` - class still named the same but moved to a different package, method names changed, returns `RemotableAttributeField` instead of `List<Row>`
- Client applications no longer allowed to override or use these attributes:
  - `DocumentSearchCriteriaProcessorAttribute`
  - `DocumentSearchGeneratorAttribute`
  - `DocumentSearchResultProcessorAttribute`
  - `DocumentSearchXMLResultProcessorAttribute`
- The only valid document search-related attributes are now:
  - `SearchableAttribute`
  - `DocumentSearchCustomizer`
  - `DocumentSecurityAttribute`
- The `DocumentSearchCustomizer` replaces most of the original document search-related attributes. It is a single attribute which allows for customization of criteria, results, and various other aspects of the Document Search operation.
- The customizer itself is not remoted, instead a `DocumentSearchCustomizationHandlerService` is published by each KEW client application. This allows for a client application to implement as many customizers as they want and map them to the document types they should apply to via `DocumentType` configuration. The Document Search functionality itself will then callback into the client application when it needs to via the published `DocumentSearchCustomizationHandlerService`.
- The new document search framework uses the `RemotableAttributeField` abstraction for defining fields that should be displayed as extra search criteria. This is in contrast to Rice 1.0.x where KNS `Row` and `Field` objects were transmitted using Java serialization. This old model did not translate well to SOAP-based services and had numerous other problems as well (namely that not all of `Field` and `Row` was properly serializable or could be interpreted correctly by the Rice standalone service, such as custom formatters, values finders, etc.).

Further documentation on how to migrate from the old document search framework to the new document search framework is forthcoming. For the time-being, the best resource is to look at the javadocs for the `org.kuali.rice.kew.framework.document.search` package. All of the framework apis should have fairly thorough javadocs. Additional, some rough notes on the document search work and architectural service approach for the customization handler service can be found near the bottom of this page: https://wiki.kuali.org/x/fwuREg

Caching Architecture

In Rice 1.0.x, caching was implemented using a framework called OSCache along with distributed caching flushing behavior across the KSB. This worked well in general, however this setup was plagued by issues of far too many cache-related message events getting generated and sent. Additionally, OSCache is a long-dead project which has not had any activity in a long time. In order to ensure that Kuali Rice was positioned for service compatibility moving forward, an effort was undertaken to revamp and update the caching architecture utilized by the Rice platform and find a better and more sustainable solution.

The result is a solution based on the caching abstraction provided with the core Spring framework version 3.1. The backend caching implementation being used is Ehcache (which is the default recommended by

Spring). There is a also integration with the Kuali Service Bus for sending cache eviction messages out to interested listeners. This allows us to keep our caching consistent across different applications in an enterprise architecture which has been integrated with Kuali Rice.

More details on the specific work that was done to rearchitect the Kuali Rice caching infrastructure can be found here: https://wiki.kuali.org/x/XgmWEg

Modularity

In version 1.0.x of Kuali Rice, the software was divided into 3 main modules: api, impl, and web. While this was convenient from the development perspective, it resulted in a high level of coupling between the various "logical" modules of Rice (the core, kew, kim, ksb, etc.). Attempts were made to rectify some of these issues in Kuali Rice 2.0. Since the full vision for modularity in Kuali Rice was not in scope for the 2.0 release, as much work was done on modularity as could be permitted in order help reach version compatibility goals for the release.

Primarily, this included the creation of numerous "api" and "framework" modules and the partitioning of Kuali Rice into various sub-project modules. This structure can be seen when looking at the source code for Rice 2.0. The overall vision for modularity in Kuali Rice as well as some notes on specific changes that were made in pursuit of it can be found at the following links:

- Modularity Use Case Analysis
- Modularity Design
- the running list of modularity changes.
- KEW Refactoring Notes

PeopleFlows

Core PeopleFlow concept

A new concept was added to Kuali Enterprise Workflow as part of workflow routing requirements for Kuali Coeus. The requirement was essentially to replicate the functionality of "Maps" in the existing Coeus system upon which Kuali Coeus is based. These maps are simply lists of people or roles who can approve in a particular order. This order is defined by a series of "stops". At each stop there can be a number of possible approvers which may have delegates.

There was not really an equivalent concept in KEW today, this is essentially like a mini-workflow itself. So the concept of PeopleFlows were added along with a new activation type called "Priority-Parallel" activation. This priority-parallel activation allows for action requests to be activated at a node in parallel if they have the same priority, but in sequence if they have different priority values.

Further details on the implementation of PeopleFlows, their lineage in Coeus, as well as some screenshots of the maintenance user interface can be found here: https://wiki.kuali.org/x/3R6REg

PeopleFlow integration with KRMS

The previous section introduced the concept of PeopleFlows. The KRMS module will have integration with PeopleFlows built in through the use of a custom type of "Action". This action will allow a rule author to build a business rule and then indicate that the action that should be executed if the rule succeeds would be to route to the specified PeopleFlow.

Details on the way this integration is implemented can be found here: https://wiki.kuali.org/x/3R6REg

New Modules: KRMS and KRAD

Kuali Rapid Application Development Framework (KRAD)

KRAD (Kuali Rapid Application Development) is a framework that eases the development of enterprise web applications by providing reusable solutions and tooling that enable developers to build in a rapid and agile fashion. KRAD is a complete framework for web development that provides infrastructure in all major areas of an application (client, business, and data) and integrates with other modules of the Rice middleware project. KRAD is the next generation of the Rice Kuali Nervous System (KNS) and Kuali Student development frameworks. It expands on the ability to rapidly create user interfaces with XML while providing more flexibility with layouts and user experience. In addition, KRAD allows for the creation of modern, rich user interfaces using the jQuery and Fluid frameworks.

How does KRAD improve on the KNS? See Benefits of KRAD.

To learn more about KRAD, please visit the following pages:

- KRAD in a nutshell
- UIF Architecture
- View Tutorial
- Creating a component

In addition, see various analysis pages here (Note: some of the information might be incomplete or out of date): KRAD Analysis

Kuali Rule Management System (KRMS)

KRMS (Kuali Rule Management System) is a new module in Rice that enables the externalization of pieces of business logic (rules) from an application such that it is customizable without need for additional client programming. Agendas containing custom business rules can be defined, edited and stored in the repository, and then executed by the application against some set of information (facts).

There is also integration between KRMS and KEW's new PeopleFlow concept that enables invocation of the rule execution engine from within a workflow. Rules can be configured with PeopleFlow actions that specify PeopleFlows to route to or to notify.

View the following analysis page for some additional information on KRMS:KRMS Analysis

Configuration Changes

- XAPool/Bitronix Configuration Changes (KRDOC-382)
- New configuration Parameters related to component publishing (KRDOC-386)
- Drop support of Tomcat 5.5, servlet spec 2.4,jsp 2.0) - KRDOC-223
- Changes to configurers and run modes (KRDOC-389)

Library Changes and Additions

There were many changes made to the various direct and transitive dependencies for Kuali Rice. Additional documentation is forthcoming on what these specific changes were, but the following is a summary of some of the highlights:

- Spring 3.1
  - currently on RC1 of this library, but it is expected that the generally available release of Spring 3.1 will be available prior to the generally available release of Rice 2.0
- Apache CXF 2.3.6
- OJB 1.0.4-patch7
  - includes various patches from the Kuali community
- Direct Web Remoting (DWR) 3.0.RC2
- Ehcache 2.4.6
- Groovy 1.8.1
- joda-time 2.0
- JOTM 2.1.9
- Bitronix 2.1.1
- Quartz 1.8.4
- Spring Security 3.0.7
- Struts 1.3.10
- Tiles 2.2.2
- jQuery 1.6.3

Other Changes

There were many other changes made in Kuali Rice 2.0 which are not included in the previous section. The following list is a highlight of some of these. For a comprehensive list of changes that were performed, the Jira issues are the best place to look.

- Removal of DateUtils and NumberUtils (KRDOC-232)
- Changes in how unit tests and integration tests are run (KRDOC-367)
- Addition of Priority parallel activation type (KRDOC-380)
- Use of immutable objects and the builder pattern for various APIs
- Removal of "object remoting" functionality from the KSB
- Conversion of identifiers from numeric to character-based data types in KEW
- Changes to PersistableBusinessObjectBase and notes (KULRICE-4862)
- Normalization on a single concept of "application id"
- Unique constraint on kim permission namespace:name and responsibility namespace:name
- Clean up of multiple "key-value-pair" style classes (KULRICE-4854)
- Transition from custom-written email services to standardizing on the spring mailer (KULRICE-4863)

- Move to Maven version 3
- DateTimeService converted to use configuration parameters instead of system parameters, allowing for it to be used through more of the system without the need for database dependencies
- Kuali Communication Broker (KCB) integration turned off by default
- Removal of IdentityManagementService, replaced with individual service calls in KIM
- Replacement of AttributeSet in KIM service apis with simple Map<String, String>
- Renaming of various data elements in KIM to more closely align with PESC standards (KULRICE-5360)
- Ability for a RouteModule to be called back into multiple times in order to generate more requests in support of new PeopleFlow feature
- Hours of availability awareness (KULRICE-5695)
- Change from WorkflowFunctions to EDLFunctions (and package name changes) in eDocLites
- Ability to define finer-grained Action List notification preferences for KEW (KULRICE-5848)

Rice 2.0 Work Remaining

Rice 2.0 is the first beta release and the goal of this release was to ensure that approximately 95% of the functionality was complete prior to the public beta test. The following sections summarize major work items that still need to be completed prior to the generally available release of Kuali Rice 2.0. Work will continue to try and resolve these issues as soon as possible for future beta releases.

"Remote" and "Embedded" client application configurations

There was much work done in Rice 2.0 to refactor the layout and organization of the project and it's services. This was primarily driven from the perspective of work being done both on improving modularity for the 2.0 release as well as requirements for version compatibility. Unfortunately, as of the first beta release for Rice 2.0, there is still cleanup work to do to allow for proper configuration and operation of non-bundled client applications. For those unfamiliar with the concept of "bundled" vs. "non-bundled", options for integration with the Kuali Rice middleware is essentially as follows:

- **bundled** - in this model, a single application is bundled together with the Kuali Rice services and web application. This allows for that application to be deployed without the need to set up a separate application deployment for Kuali Rice. This model is not intended for "enterprise" application deployments because it prohibits proper reuse of the Kuali Rice middleware as a shared set of services.
- **standalone** - in this model, a Kuali Rice "standalone server" is deployed which hosts the Kuali Rice web application along with all of it's services. Client applications can then integrate with the standalone server instance through various "run modes" which are supported by the client-side pieces for the different Kuali Rice components (KEW, KIM, KSB, etc.). This integration model makes more sense for an enterprise deployment of Kuali Rice because it allows the standalone server instance and the services it provides to be shared and leveraged by multiple applications within the enterprise.

For applications that are integrating with a standalone server, there are three primary possibilities for integration (generally, it's possible to integrate with different modules of Rice in different ways):

- **remote** - in this mode, the client application will be configured to connect to the KSB service registry and will invoke remote rice services using the server endpoint information located within the registry

- **embedded** - in this mode, some or all of the module will be "embedded" into the client application (in Rice 2.0, this mode is only supported by KEW and KIM). When using this client-integration mode, the client application will connect directly to the Kuali Rice database.

- **manual** - all of the Kuali Rice services are available as SOAP web services, so any application can build a manual web-service client using information found in the WSDL's for the services published by the Kuali Rice standalone server. In these cases, there is generally no Kuali Rice code that need to be configured or used within the client application.

Because of the issues mentioned previously, not all of the modules in Rice are configured to fully support the remote and embedded configurations (however, manual web-service integration should be possible). For this reason, we are recommending those participating in the beta test start by working on upgrades to any Rice standalone server instances they have as well as running Rice 2.0 in "bundled" mode with any existing applications they wish to convert to Rice 2.0.

Configurers

Related to the previous item of remote and embedded client application configuration, part of the issue here lies with the fact that there is still outstanding work on how to set up client "configurers". Those familiar with Rice 1.0.x will recall the `RiceConfigurer` class which was used to load and configure the various Rice modules. In Rice 2.0, this class no longer exists and it's replacement is `org.kuali.rice.core.impl.config.module.CoreConfigurer`. However, the core configurer only configures the basic "core" of Kuali Rice. The other modules (such as KEW, KSB, etc.) have their own configurers as well.

The best place to look at for an example of how to configure Rice 2.0 is in the file `impl/src/main/resources/org/kuali/rice/config/RiceSpringBeans.xml`. This file imports other spring files which configure the various Rice modules. These files can also be imported into an application's Spring file in order to configure that particular module (unless an alternative or customized configuration is needed). Note, however, that the way in which Rice is configured will likely be adjusted in future beta releases, but information and documentation on such changes will be communicated along with those releases.

Caching

There is still work outstanding on the caching infrastructure for Kuali Rice. Remaining work primarily revolves around how client-side caching is managed and performed. Much of this work relates to the outstanding issues resulting in the inability of configuring remote and embedded client-side integration patterns.

KEN and KCB Services

The KEN and KCB services have not yet gone through the refactoring to align them with Kuali Rice 2.0 service compatibility standards. The services that are published for these modules will likely change in subsequent beta releases. However, they should be finalized prior to the final release of Rice 2.0.

Custom Action List Attributes

Work is still being undertaken to reimplement the Action List customization framework so that it adheres to Rice 2.0 version compatibility standards.

Document Search

As a result of heaving refactoring of the document search frameworks for version compatibility in Rice 2.0, there is still work remaining before this framework will have parity with the implementation that was in place in Kuali Rice 1.0.x. Specific outstanding work that remains here includes:

- Support for range-based searches on search attribute values
- Application document status search support
- Route Node Name and Route Node Logic search support
- Proper data formatting in result set
- Ability to indicate column visibility on searchable attribute fields
- Ability to customize help link on document search
- Ability to pass query parameters via the document search URL
- General cleanup and documentation

createproject.groovy

The createproject.groovy script has not yet been converted over to produce a valid Rice 2.0-based project.

Documentation

Document is still in a rudimentary state. During the course of the beta releases, it will be reviewed and augmented.

Upgrade Guides and Upgrade Scripts

These guides and scripts are not yet fully complete. However, final documentation and scripts will be provided prior to the final release. To access rough versions of scripts and some contributed by other members of the community, please visit the beta test site.

Upgrade Guide

Database Changes

The scripts to upgrade from a 1.0.3.3 version of Kuali Rice to version 2.0.0-b1-SNAPSHOT can be found in the following directory inside of the source distribution:

```
scripts/upgrades/1.0.3 to 2.0.0
```

Within this directory, are the following sub-directories:

- **db-updates** - contains the SQL needed to upgrade from a Kuali Rice 1.0.3.3 standalone server service database to 2.0.0-b1-SNAPSHOT. Each of these scripts is dated and that indicates the order in which they should be run. SQL scripts for MySql are prefixed by "mysql". Scripts for oracle have no prefix on their name.

- **db-updates-client** - contains the SQL needed to upgrade a Kuali Rice client application's Rice-related tables to 2.0.0-b1-SNAPSHOT. At this point in time, the only update required is to the KRNS_SESN_DOC_T table.
- **demo-data** - this directory contains updates to the various pieces of Kuali Rice demo data. Chances are that most people performing an upgrade will not have demo data in their local database and will, therefore, not need to execute the scripts located within this directory.
- **support** - this directory contains a groovy script which helps deal with the fact that unique constraints were added to the database for the combination of namespace code and name in both the KIM permission and responsibility tables
- **xml-ingest-updates** - contains XML that needs to be ingested after the database upgrade is completed

Jiras

There were nearly 800 Jira issues addressed in Rice 2.0. Below are links to Jira filters for the items addressed.

- All Rice 2.0 Jiras
- Bug Fixes
- Improvements
- Tasks & SubTasks