Kuali Rice 2.4.0-M3-SNAPSHOT KNS to KRAD Conversion Guide

Released: 12-16-2013

Table of Contents

Introduction

This technical documentation outlines what is needed for conversion of existing Kuali Rice applications using KNS to move to the new KRAD development framework. Here we identify the changes between the KNS and KRAD application features. Detailed documentation of KRAD features and how to use them are not included in this document, but can be found in the KRAD Guide.

The KNS2KRAD Conversion Script is a tool provided by the Kuali Rice team to automate as much as is reasonably possible. Throughout this document the following items are used to indicate if the item is converted via the script or must be done manually.

script conversion Feature can be converted with the conversion script.

manual conversion Feature conversion is not easily automated and needs to be converted manually.

Process Overview

There are four main steps involved in converting from KNS to KRAD:

1. Convert data dictionary into krad compliant format.
   - Convert validation patterns to constraint.
   - Convert control definitions into UIF controls.
2. Convert KNS definitions into KRAD UIF components.
   - Convert inquiry and lookup definitions into view.
   - Convert section definition into UIF groups
   - Convert field definitions into UIF input fields
3. Convert Struts to Spring MVC.
   - Convert Struts and their actions into UIF controllers.

- Convert action paths into request mapping annotations in the controller.

4. Convert JSP and tags into UIF components
   - Convert jsp tags into UIF components.
   - Convert jstl calls into spring el conditionals.
   - Convert document into view.
   - Convert kul:tabs into UIF disclosure.
   - Convert html tables into grid layouts.
   - Convert attributes/control tags into UIF input fields.
   - Convert image submit tags into action buttons

How to Run the Conversion Script

The automated KRAD conversion script is called KRADConversion.groovy and will process lookup, inquiry, and MaintenanceDocumentEntries into KRAD-compliant XML files. To run the script, complete the following steps:

1. Step 1
2. Step 2
3. ...

Data Dictionary Conversion

The Data Dictionary is a repository of metadata primarily describing data objects and their properties with attribute Definitions that provide the metadata about the attributes (i.e. fields) of a data object. The conversion script will automate part of the conversion for your data dictionary files, but some data dictionary attributes require no conversion since they are virtually identical between KNS and KRAD. The following sections identify the Business Object bean properties, Validation Patterns, and Control Definitions which are converted by the script and those which require manual conversion.

BusinessObjectEntry bean replaced by DataObjectEntry bean

KNS Code example:

```
 1 <bean id="Book" parent="Book-parentBean"/>
 2 <bean id="Book-parentBean" abstract="true" parent="BusinessObjectEntry">
 3   <property name="businessObjectClass" value="org.kuali.rice.knsapp.Book"/>
 4   <property name="inquiryDefinition">
 5     <ref bean="Book-inquiryDefinition"/>
 6   </property>
 7   <property name="lookupDefinition">
 8     <ref bean="Book-lookupDefinition"/>
 9   </property>
10   <property name="titleAttribute" value="id"/>
```

```
11   <property name="objectLabel" value="Book"/>
12   <property name="attributes">
13     <list>
14       <ref bean="Book-id"/>
15       <ref bean="Book-title"/>
16       <ref bean="Book-price"/>
17     </list>
18   </property>
19 </bean>
```

KRAD code example:

```
 1 <bean id="Book" parent="Book-parentBean"/>
 2 <bean id="Book-parentBean" abstract="true" parent="DataObjectEntry">
 3   <property name="dataObjectClass" value="org.kuali.rice.knsapp.Book"/>
 4   <property name="titleAttribute" value="id"/>
 5   <property name="objectLabel" value="Book"/>
 6   <property name="attributes">
 7     <list>
 8       <ref bean="Book-id"/>
 9       <ref bean="Book-title"/>
10       <ref bean="Book-price"/>
11     </list>
12   </property>
13 </bean>
```

- The parent bean for data dictionary beans in KRAD is **`DataObjectEntry`** instead of the KNS **`BusinessObjectEntry`**. [KNS line: 2, KRAD line: 2] script conversion
- The data object class is specified via the **`dataObjectClass`** property is no longer specified via the **`businessObjectClass`** property. A **`businessObject`** is no longer required and any object can be used. [KNS line: 3, KRAD line: 3] script conversion

Attribute Definitions

Attribute Definitions define the meta-data of an object and many of the Attribute Definition properties remain the same between KNS and KRAD. The major differences are in how validation constraints and controls are defined. The examples below demonstrate the differences for defining various validation and controls.

KNS Attribute Definition code example:

```
 1 <bean id="Country-code" parent="Country-code-parentBean"/>
 2 <bean id="Country-code-parentBean" abstract="true" parent="AttributeDefinition">
 3   <property name="name" value="code"/>
 4   <property name="forceUppercase" value="true"/>
 5   <property name="label" value="Country Code"/>
 6   <property name="shortLabel" value="Country Code"/>
 7   <property name="maxLength" value="2"/>
 8   <property name="required" value="true"/>
 9   <property name="summary" value="Postal Country Code"/>
10   <property name="description" value="The code uniquely identify a country."/>
11   <property name="validationPattern">
12     <bean parent="AlphaNumericValidationPattern"/>
13   </property>
14   <property name="control">
15     <bean parent="TextControlDefinition" p:size="2"/>
16   </property>
17 </bean>
```

KRAD Attribute Definition code example:

```
 1 <bean id="Country-code" parent="Country-code-parentBean"/>
 2 <bean id="Country-code-parentBean" abstract="true" parent="AttributeDefinition">
 3   <property name="name" value="code"/>
 4   <property name="forceUppercase" value="true"/>
 5   <property name="label" value="Country Code"/>
 6   <property name="shortLabel" value="Country Code"/>
 7   <property name="maxLength" value="2"/>
 8   <property name="required" value="true"/>
 9   <property name="summary" value="Postal Country Code"/>
10   <property name="description" value="The code uniquely identify a country."/>
11   <property name="validCharactersConstraint">
12     <bean parent="AlphaNumericPatternConstraint"/>
13   </property>
14   <property name="controlField">
15     <bean parent="Uif-TextControl" p:size="2"/>
16   </property>
17 </bean>
```

- The property **`validCharactersConstraint`** is used in KRAD in place of the KNS property **`validationPattern`**. [KNS line: 11, KRAD line: 11] script conversion
- The constraint **`AnyCharacterPatternConstraint`** replaces the KNS pattern **`AnyCharacterValidationPattern`**. [KNS line: 12, KRAD line: 12] script conversion
- The property **`controlField`** replaces the KNS **`control`** to define the control used to represent this field. [KNS line: 14, KRAD line: 14] script conversion
- The property **`Uif-TextControl`** replaces the KNS **`TextControlDefinition`**. [KNS line: 5, KRAD line: 15] script conversion
- The property **`forceUpperCase`** remains the same in KRAD, however you can also specify **`@ForceUppercase`** annotation for a property in KRAD.
- The size property and similar properties of TextControl are converted or carried over by the script. [KNS line: 15, KRAD line: 15] script conversion

Validation Patterns

The KNS validation patterns are supported in KRAD, but the naming convention has changed for constraints and patterns even though the functionality is the same. KRAD also has several other validations available which are described in more detail in the KRAD Guide. The example below demonstrates the differences in defining validation constraints using a single validation pattern and several such patterns are available.

KNS validationPattern property example:

```
1 <property name="validationPattern">
2   <bean  parent="AnyCharacterValidationPattern" p:allowWhitespace="true"/>
3 </property>
```

KRAD validCharactersConstraint property example:

```
1 <property name="validCharactersConstraint">
2   <bean parent="AnyCharacterPatternConstraint" p:allowWhitespace="true"/>
```

```
3 </property>
```

- The property **validCharactersConstraint** is used in KRAD in place of the KNS property **validationPattern**. script conversion
- The constraint **AnyCharacterPatternConstraint** replaces the KNS pattern **AnyCharacterValidationPattern**. script conversion
- The constraint **AlphaNumericPatternConstraint** replaces the KNS pattern **AlphaNumericValidationPattern**. script conversion
- The constraint **AlphaPatternConstraint** replaces the KNS pattern **AlphaValidationPattern**. script conversion
- The constraint **CharsetPatternConstraint** replaces the KNS pattern **CharsetValidationPattern**. script conversion
- The constraint **RegexPatternConstraint** replaces the KNS pattern **RegexValidationPattern**. script conversion
- The constraint **FixedPointPatternConstraint** replaces the KNS pattern **FixedPointValidationPattern**. script conversion
- The constraint **FloatingPointPatternConstraint** replaces the KNS pattern **FloatingPointValidationPattern**. script conversion
- The constraint **ZipcodePatternConstraint** replaces the KNS pattern **ZipcodeValidationPattern**. script conversion
- The constraint **YearPatternConstraint** replaces the KNS pattern **YearValidationPattern**. script conversion
- The constraint **TimestampPatternConstraint** replaces the KNS pattern **TimestampValidationPattern**. script conversion
- The constraint **PhoneUSPatternConstraint** replaces the KNS pattern **PhoneNumberValidationPattern**. script conversion
- The constraint **MonthPatternConstraint** replaces the KNS pattern **MonthValidationPattern**. script conversion
- The constraint **JavaClassPatternConstraint** replaces the KNS pattern **JavaClassValidationPattern**. script conversion
- The constraint **EmailPatternConstraint** replaces the KNS pattern **EmailAddressValidationPattern**. script conversion

- The constraint **`DatePatternConstraint`** replaces the KNS pattern **`DateValidationPattern`**. script conversion
- The constraint **`UTF8AnyCharacterValidationPattern`** replaces the KNS pattern **`UTF8AnyCharacterValidationPattern`**. script conversion
- Sub-properties for the validation pattern constraint beans, such as **`allowWhitespace="true"`** in the above example are either carried over or converted as appropriate. script conversion

Other Constraints

In addition to the **`validCharactersConstraint`**, KRAD has several other constraints available:

- **`AllowCharacterConstraint`**
- **`BaseConstraint`**
- **`CaseConstraint`**
- **`CollectionSizeConstraint`**
- **`DataTypeConstraint`**
- **`ExistenceConstraint`**
- **`LengthConstraint`**
- **`LookupConstraint`**
- **`MustOccurConstraint`**
- **`PrerequisiteConstraint`**
- **`RangeConstraint`**
- **`SimpleConstraint`**
- **`WhenConstraint`**

See the KRAD Guide for more detail.

Additional Constraint Patterns

KRAD has additional constraint patterns too such as:

- **`ConfigurationBasedRegexPatternConstraint`**
- **`IntegerPatternConstraint`**
- **`ValidCharactersPatternConstraint`**
- **`ValidDataPatternConstraint`**

See the KRAD Guide for more detail.

Controls

Control components are defined to determine which HTML element(s) are rendered to represent the input field. Here is an example of a Text control within an Attribute Definition.

KNS Control example:

```
1 <property name="control">
2   <bean parent="TextControlDefinition" p:size="10"/>
3 </property>
```

KRAD Control example:

```
1 <property name="controlField">
2   <bean parent="Uif-TextControl" p:size="10"/>
3 </property>
```

- The property **`controlField`** replaces the KNS **`control`** to define the control used to represent this field. [KNS line: 1, KRAD line: 1] script conversion
- The property **`control`** is currently deprecated.
- The property names of the various controls have changed. In the above example, **`Uif-TextControl`** replaces the KNS **`TextControlDefinition`**.[KNS line: 2, KRAD line: 2] script conversion
- The sub-properties for the various controls, ex: **`p:size="10"`** are converted or carried over by the script. [KNS line: 2, KRAD line: 2] script conversion

Button Control

An action component that is configured to render a button. The button element can include text (the label) along with an image.

KNS Button Control example:

```
1 <property name="control">
2   <bean parent="ButtonControlDefinition"/>
3 </property>
```

KRAD Button Control example:

```
1 <property name="controlField">
2   <bean parent="Uif-PrimaryActionButton" id="Submit" p:methodToCall="save"/>
3 </property>
```

- The property **`Uif-PrimaryActionButton`** replaces the KNS **`ButtonControlDefinition`**. script conversion
- There are actually several alternative action controls pre-configured with different styling. These include:
  - **`Uif-PrimaryActionButton-Small`**

- **`Uif-SecondaryActionButton`**
- **`Uif-SecondaryActionButton-Small`**
- **`Uif-ActionImage`**
- **`Uif-ActionLink`**

See the KRAD Guide. for more detail

Checkbox Control

The Checkbox control renders an HTML input tag with type of “checkbox”. This control is used to toggle the state of a property between two values (usually the Booleans true and false).

KNS Checkbox Control example:

```
1 <property name="control">
2   <bean parent="CheckboxControlDefinition"/>
3 </property>
```

KRAD Checkbox Control example:

```
1 <property name="controlField">
2   <bean parent="Uif-CheckboxControl"/>
3 </property>
```

- The property **`Uif-CheckboxControl`** replaces the KNS **`CheckboxControlDefinition`**.script conversion
- Also available in KRAD are the Checkbox Group Controls: **`Uif-VerticalCheckboxesControl`** and **`Uif-HorizontalCheckboxesControl`**. The CheckboxesGroup control is a multi-value control that presents each option as a checkbox. See the KRAD Guide. for more detail.

Currency Control

Same as a Text control, except has an added style class of 'uif-currencyControl' which adds a right aligh style to the control useful for diplaying currency.

KNS Currency control example:

```
1 <property name="control">
2   <bean parent="CurrencyControlDefinition" p:formattedMaxLength="26" p:size="10"/>
3 </property>
```

KRAD Currency Text control example:

```
1 <property name="controlField">
2   <bean parent="Uif-CurrencyTextControl" p:maxLength="26" p:size="10" />
3 </property>
```

- The property **`Uif-CurrencyTextControl`** replaces the KNS **`CurrencyControlDefinition`**.[KNS line: 2, KRAD line: 2]script conversion

- The sub-property **`maxlength`** replaces the KNS **`formattedMaxLength`** property.[KNS line: 2, KRAD line: 2] script conversion

File Control

The File control is used to allow the user to select a file from their file system whose contents will be submitted with the form. The server can then make use of the file contents or simply store the file on the server (for example a note attachment).

KNS File Control example:

```
1 <property name="control">
2   <bean parent="FileControlDefinition"/>
3 </property>
```

KRAD File Control example:

```
1 <property name="controlField">
2   <bean parent="Uif-FileControl"/>
3 </property>
```

- The property **`Uif-FileControl`** replaces the KNS **`FileControlDefinition`**. script conversion

Hidden Control

The Hidden control is used to render an HTML input of type hidden. A hidden control is not visible to the user, therefore its value can only be changed by script. These are often used to hold some state that is needed when the page is posted back, or to provide data for scripting purposes.

KNS Hidden control example:

```
1 <property name="control">
2   <bean parent="HiddenControlDefinition"/>
3 </property>
```

KRAD Hidden control example:

```
1 <property name="controlField">
2   <bean parent="Uif-HiddenControl"/>
3 </property>
```

- The property **`Uif-HiddenControl`** replaces the KNS **`HiddenControlDefinition`**. script conversion

Kuali User Control

The KIM User control is specifically tailored to represent a KIM user. This control does several things for us. First like the group control, it will configured a quickfinder for our field that is configured to invoke the KIM User lookup. The lookup will then return the principal id, principal name (username), and person

name (full name). Also like the group control it will automatically add the principal id as a hidden field for us. In addition, it sets up a field query (covered in later on in this chapter) that display the person name under the control on return from the lookup or when tabbing out of the control.

KNS Kuali User control example:

```
1 <property name="control">
2   <bean parent="KualiUserControlDefinition"/>
3 </property>
```

KRAD Kim Person control example:

```
1 <property name="controlField">
2   <bean parent="Uif-KimPersonControl"/>
3 </property>
```

- The property **`Uif-KimPersonControl`** replaces the KNS **`KualiUserControlDefinition`**.script conversion

Link Control

The Link control generates the HTML a (link) tag. The a tag is used to link to another document (the primary mechanism of navigation in the web). The link is presented to use by a label, which when clicked on will take the user to the linked page.

KNS Link control example:

```
1 <property name="control">
2   <bean parent="LinkControlDefinition" p:styleClass="globalLinks"
3         p:target="_blank" p:hrefText="click here" />
4 </property>
```

KRAD Link control example:

```
1 <property name="controlField">
2   <bean parent="Uif-LinkField" p:fieldLabel.cssClasses="globalLinks"
3         p:target="_blank" p:linkText="click here" href="@{#propertyName}"/>
4 </property>
```

- The property **`Uif-Link`** replaces the KNS **`LinkControlDefinition`**.[KNS line: 2, KRAD line: 2]script conversion
- The sub-property **`target`** is carried over during conversion.[KNS line: 2, KRAD line: 2]script conversion
- The sub-property **`linkText`** replaces the KNS **`hrefText`** property.[KNS line: 2, KRAD line: 2]script conversion
- The sub-property **`fieldLabel.cssClasses`** replaces the KNS **`styleClass`** property.[KNS line: 2, KRAD line: 2]script conversion

TextArea Control

The TextArea control is similar to the text control with the exception of providing multiple lines for input. This control is used for entering longer strings of data such as a description.

KNS TextArea control example:

```
1 <property name="control">
2   <bean parent="TextareaControlDefinition"/>
3 </property>
```

KRAD TextArea control example:

```
1 <property name="controlField">
2   <bean parent="Uif-TextAreaControl"/>
3 </property>
```

- The property `Uif-TextAreaControl` replaces the KNS `TextareaControlDefinition`. script conversion

Text Control

The Text control renders the HTML input element with type of “text”. This is a single-line box that allows the user to type the value.

KNS Text control example:

```
1 <property name="control">
2   <bean parent="TextControlDefinition"/>
3 </property>
```

KRAD Text control example:

```
1 <property name="controlField">
2   <bean parent="Uif-TextControl"/>
3 </property>
```

- The property `Uif-TextControl` replaces the KNS `TextControlDefinition`. script conversion
- Note: There are several other text controls to choose from with different default styling:
  - `Uif-SmallTextControl` – Similar to Uif-TextControl but sets the size to 10 and applies an additional style class of 'uif-smallTextControl'.
  - `Uif-MediumTextControl` – The same as Uif-TextControl except adds a style class of 'uif-mediumTextControl'.
  - `Uif-LargeTextControl` – Similar to Uif-TextControl but sets the size to 100 and applies an additional style class of 'uif-largeTextControl'.

  See the KRAD Guide for more detail.

Radio Control Group

A radio control allows a user to choose only one of a predefined set of options.

KNS Radio Group control example:

```
1 <property name="control">
2   <bean parent="RadioControlDefinition"
3         p:valuesFinderClass="org.kuali.rice.krad.keyvalues.DelegateRuleValuesFinder"/>
4 </property>
```

KRAD Radio Group control example:

```
1 <bean parent="Uif-InputField" p:propertyName="selectedOpt" p:label="Radio 1">
2   <property name="control">
3     <bean parent="Uif-VerticalRadioControl"/>
4   </property>
5   <property name="optionsFinder">
6     <bean class="org.kuali.rice.krad.keyvalues.DelegateRuleValuesFinder"/>
7   </property>
8 </bean>
```

- To define a Radio Control in KRAD, an **`Uif-InputField`** with a **`Uif-VerticalRadioControl`** control is used instead of the KNS **`RadioControlDefinition`** script conversion
- The property **`optionsFinder`** replaces the KNS **`valuesFinderClass`**. [KNS line: 2, KRAD lines: 5-7] script conversion
- **`Uif-HorizontalRadioControl`** is also available, the options are displayed horizontally instead of vertically.
- Note: **`options`** or **`optionsFinder`** may be used to define the choices in the Radio Group control. This also applies to Select controls.

Select Control

A select control provides a drop-down list of options. It is essentially an alternative to radio buttons. Or a checkbox group if set to allow multiple selected values

KNS Select control example:

```
 1 <bean id="BookOrder-bookId" parent="BookOrder-bookId-parentBean"/>
 2 <bean id="BookOrder-bookId-parentBean" abstract="true" parent="AttributeDefinition">
 3   <property name="name" value="bookId"/>
 4   <property name="label" value="Book Id"/>
 5   <property name="shortLabel" value="Book Id"/>
 6   <property name="maxLength" value="19"/>
 7   <property name="validationPattern">
 8     <bean parent="NumericValidationPattern"/>
 9   </property>
10   <property name="control">
11     <bean parent="SelectControlDefinition" p:businessObjectClass="edu.sampleu.bookstore.bo.Book"
12               p:valuesFinderClass="org.kuali.rice.krad.keyvalues.PersistableBusinessObjectValuesFinder"
13               p:includeKeyInLabel="false" p:includeBlankRow="true" p:keyAttribute="id"
14               p:labelAttribute="para"/>
15   </property>
```

```
16 </bean>
```

KRAD Select control example:

```
 1 <bean id="BookOrder-bookId" parent="BookOrder-bookId-parentBean"/>
 2 <bean id="BookOrder-bookId-parentBean" abstract="true" parent="AttributeDefinition">
 3   <property name="name" value="bookId"/>
 4   <property name="label" value="Book Id"/>
 5   <property name="shortLabel" value="Book Id"/>
 6   <property name="maxLength" value="19"/>
 7   <property name="validCharactersConstraint">
 8     <bean parent="NumericPatternConstraint"/>
 9   </property>
10   <property name="controlField">
11     <bean parent="Uif-DropdownControl"/>
12   </property>
13   <property name="optionsFinder">
14     <bean class="org.kuali.rice.krad.keyvalues.PersistableBusinessObjectValuesFinder"/>
15   </property>
16 </bean>
```

- The property **`Uif-DropdownControl`** replaces the KNS **`SelectControlDefinition`**. [KNS line: 11, KRAD line: 11] script conversion
- The property **`optionsFinder`** replaces the KNS **`valuesFinderClass`**. [KNS line: 12, KRAD lines: 13-15] script conversion
- The sub-properties for the SelectControlDefinition in KNS ex: **`p:includeBlankRow="true"`** carried over by the conversion script. [KNS line: 13] script conversion
- The property **`Uif-MultiSelectControl`** also replaces the KNS **`MultiselectControlDefinition`** You may also specify multi value select capability from a Uif-DropdownControl, by setting property **`multiple`** to true. script conversion

Date Control

A date control is an input element with a type attribute of "date" and represents a control for setting the element's value to a string representing a date.

KNS Date control example:

```
1 <property name="control">
2   <bean parent="TextareaControlDefinition"/>
3 </property>
```

KRAD TextArea control example:

```
1 <property name="controlField">
2   <bean parent="Uif-DateControl"/>
3 </property>
```

- The property **`Uif-DateControl`** replaces the KNS **`DateControlDefinition`**. script conversion

Workflow Workgroup Control

- No conversion for Workflow Workgroup Control yet. manual conversion

Inquiry Conversion

Inquiries allow users to quickly view a data entity (typically one created via a maintenance document) without having to start editing that entity. They provide a more compact view of these entities and can control which users see what displayed data. KRAD moves all inquiry definitions to a view and displays all inquiries in a lightbox by default. The conversion script will automate part of the conversion for your inquiries to the view format, but some inquiry attributes require no conversion since they are virtually identical between KNS and KRAD. The following sections provide a typical conversion sample and identify the Business Object bean properties, Sections, Collections, helper classes, and other features which are converted by the script and those which require manual conversion.

InquiryDefinition in KNS Data Dictionary to InquiryView in KRAD

KNS Code example:

```
 1 <bean id="EntityType-inquiryDefinition" parent="InquiryDefinition">
 2   <property name="title" value="Entity Type Inquiry"/>
 3   <property name="inquirableClass" value="org.kuali.rice.kim.inquiry.EntityTypeInquirableImpl"/>
 4   <property name="authorizerClass" value="org.kuali.rice.kim.authorization.EntityTypeAuthorizer"/>
 5   <property name="presentationControllerClass"
value="org.kuali.rice.kim.presentation.EntityTypePresentationController"/>
 6   <property name="inquirySections">
 7     <list>
 8       <bean parent="InquirySectionDefinition">
 9         <property name="title" value="Entity Type"/>
10         <property name="defaultOpen" value="true"/>
11         <property name="numberOfColumns" value="1"/>
12         <property name="inquiryFields">
13           <list>
14             <bean parent="FieldDefinition" p:attributeName="code" p:noInquiry="true"/>
15             <bean parent="FieldDefinition" p:attributeName="name"/>
16             <bean parent="FieldDefinition" p:attributeName="active"/>
17           </list>
18         </property>
19       </bean>
20       <bean parent="InquirySectionDefinition">
21         <property name="title" value="Entity Type Details"/>
22         <property name="defaultOpen" value="false"/>
23         <property name="inquiryFields">
24           <list>
25             <bean parent="InquiryCollectionDefinition">
26               <property name="numberOfColumns" value="1"/>
27               <property name="businessObjectClass"
28                         value="org.kuali.rice.kim.impl.identity.EntityTypeDetailsBo"/>
29               <property name="attributeName" value="entityTypeDetails"/>
30                 <property name="inquiryFields">
31                   <list>
32                     <bean parent="FieldDefinition" p:attributeName="name"/>
33                     <bean parent="FieldDefinition" p:attributeName="value"/>
34                   </list>
35                 </property>
36               <property name="summaryTitle" value="Entity Type Details"/>
37               <property name="summaryFields">
38                 <list>
39                   <bean parent="FieldDefinition" p:attributeName="name"/>
40                 </list>
41               </property>
42             </bean>
43           </list>
```

```
44         </property>
45       </bean>
46     </list>
47   </property>
48 </bean>
```

KRAD code example:

```
 1 <bean id="EntityType-InquiryView" parent="Uif-InquiryView">
 2   <property name="headerText" value="Entity Type"/>
 3   <property name="dataObjectClassName" value="org.kuali.rice.kim.impl.identity.EntityTypeBo"/>
 4   <property name="viewHelperServiceClass" value="org.kuali.rice.kim.inquiry.EntityTypeInquirableImpl"/>
 5   <property name="authorizerClass" value="org.kuali.rice.kim.authorization.EntityTypeAuthorizer"/>
 6   <property name="presentationControllerClass"
value="org.kuali.rice.kim.view.EntityTypeViewPresentationController"/>
 7   <property name="items">
 8     <list>
 9       <bean id="EntityType-InquiryView-General" parent="Uif-Disclosure-GridSection">
10         <property name="headerText" value="Entity Type"/>
11         <property name="disclosure.defaultOpen" value="true"/>
12         <property name="layoutManager.numberOfColumns" value="2"/>
13         <property name="items">
14           <list>
15             <bean parent="Uif-DataField" p:propertyName="code" p:inquiry.render="false"/>
16             <bean parent="Uif-DataField" p:propertyName="name"/>
17             <bean parent="Uif-DataField" p:propertyName="active"/>
18           </list>
19         </property>
20       </bean>
21       <bean id="EntityType-InquiryView-Details" parent="Uif-Disclosure-StackedCollectionSection">
22         <property name="headerText" value="Entity Type Details"/>
23         <property name="disclosure.defaultOpen" value="false"/>
24         <property name="layoutManager.numberOfColumns" value="2"/>
25         <property name="collectionObjectClass"
26                   value="org.kuali.rice.kim.impl.identity.EntityTypeDetailsBo"/>
27         <property name="propertyName" value="entityTypeDetails"/>
28         <property name="items">
29           <list>
30             <bean parent="Uif-DataField" p:attributeName="name" p:inquiry.render="false"/>
31             <bean parent="Uif-DataField" p:attributeName="value"/>
32           </list>
33         </property>
34         <property name="layoutManager.summaryTitle" value="Entity Type Details"/>
35         <property name="layoutManager.summaryFields">
36           <list>
37             <value>name</value>
38           </list>
39         </property>
40       </bean>
41     </list>
42   </property>
43 </bean>
```

Base Inquiry Bean

- Inquiries are now defined via **`Uif-InquiryView`** definitions instead of the KNS **`InquiryDefinition`** beans. [KNS line: 1, KRAD line: 1] script conversion
- The title of the view is specified via the **`headerText`** property instead of the **`title`** property. [KNS line: 2, KRAD line: 2] script conversion
- The data object class is specified via the **`dataObjectClassName`** property and is no longer specified via the **`BusinessObjectEntry`** as it doesn't exist in KRAD anymore. A **`businessObject`** is no longer required and any object can be used. [KNS line: n/a, KRAD line: 3] script conversion

- The **`inquirableClass`** has now changed to **`viewHelperServiceClass`**, along with a class parent change. See below for more information. [KNS line: 3, KRAD line: 4] script conversion
- The **`authorizerClass`** has now moved to the view. See below for more information. [KNS line: 4, KRAD line: 5] script conversion
- The **`presentationControllerClass`** has now moved to the view. See below for more information. [KNS line: 5, KRAD line: 6] script conversion

Sections

- The property that contains the inquiry sections was renamed from **`inquirySections`** to **`items`**. [KNS line: 6, KRAD line: 7] script conversion
- The bean **`InquirySectionDefinition`** was removed and can be replaced with any subclass of **`Group`**, although the default is a child of **`Uif-Disclosure-GridSection`**. [KNS line: 8, KRAD line: 9] script conversion
- The title of the section is specified via the **`headerText`** property instead of the **`title`** property. [KNS line: 9, KRAD line: 10] script conversion
- The property that controls whether the section defaults to open has moved from **`defaultOpen`** to the **`Disclosure`** object and is now accessed via **`disclosure.defaultOpen`**. [KNS line: 10, KRAD line: 11] script conversion
- To use multiple columns for the section use the **`layoutManager.numberOfColumns`** property to configure the layout manager instead of specifying the **`numberOfColumns`** property. Note that in the layout manager the field label and the field itself have their own columns and therefore the old **`numberOfColumns`** value needs to be doubled. [KNS line: 11, KRAD line: 12] script conversion
- The org.kuali.rice.kns.inquiry.Inquirable.addAdditionalSections() has been removed in KRAD. To add additional sections dynamically to an inquiry override **`performCustomInitialization`** method of ViewHelperService instead and add custom components. The custom Inquirable implementation can be specified using the **`viewHelperServiceClass`** property of the view which in turn would extend **`org.kuali.rice.krad.inquiry.InquirableImpl`**. manual conversion

```
1 <bean id="Sample-InquiryView" parent="Uif-InquiryView">
2   ...
3   <property name="viewHelperServiceClass"  value="org.kuali.rice.krad.inquiry.CustomInquirableImpl/>
4   ...
5 </bean>
```

- The org.kuali.rice.kns.inquiry.Inquirable.getSection() has been removed in KRAD. To modify existing sections dynamically in an inquiry override **`performCustomInitialization`** method of ViewHelperService. manual conversion

- The property that contains the inquiry display fields was renamed from **`inquiryFields`** to **`items`**. [KNS line: 12, KRAD line: 13] script conversion
- Data fields changed from **`FieldDefinition`** to **`Uif-DataField`**. [KNS line: 14-16, KRAD line: 15-17] script conversion
- The **`attributeName`** property on the field is now a **`propertyName`** property. [KNS line: 14-16, KRAD line: 15-17] script conversion
- The **`noInquiry`** property, which suppresses rendering the inquiry link on the fields, has moved to the **`Inquiry`** object and is now accessed via **`inquiry.render`**. Note here that **`noInquiry="true"`** is equivalent to **`inquiry.render="false"`**. [KNS line: 14, KRAD line: 15] script conversion

Collections

- The bean **`InquiryCollectionDefinition`** was removed and can be replaced with any subclass of **`CollectionGroup`**, although the default is a child of **`Uif-Disclosure-StackedCollectionSection`**. [KNS line: 25, KRAD line: 21] script conversion
- The **`businessObjectClass`** property of the collection has been changed to **`collectionObjectClass`**. [KNS line: 27, KRAD line: 25] script conversion
- The **`attributeName`** property of the collection has been changed to **`propertyName`**. [KNS line: 28, KRAD line: 26] script conversion
- The property that contains the inquiry display fields was renamed from **`inquiryFields`** to **`items`**. [KNS line: 29, KRAD line: 30] script conversion
- The property **`summaryTitle`** has been moved to **`layoutManager.summaryTitle`**. [KNS line: 35, KRAD line: 33] script conversion
- The container **`summaryFields`** has been moved to **`layoutManager.summaryFields`** and simplified to just take a list of values. [KNS line: 36, KRAD line: 34] script conversion
- In general, only simple nestings such as the ones shown above are supported by the conversion script. Any other combinations, such as **`InquiryCollectionDefinition`**s directly inside an **`inquiryFields`** property or inside another **`InquiryCollectionDefinition`** are not supported and will need to be converted manually. manual conversion

Inquirable / KualiInquirableImpl

KRAD adds **`InquirableImpl`** to eventually replace **`KualiInquirableImpl`**. Normally, this class does not need to be extended unless advanced customizations are required.

- In order to customize Inquiry links, the code originally in a subclass of **`org.kuali.rice.kns.inquiry.Inquirable.getInquiryUrl`** will need to move to **`org.kuali.rice.krad.inquiry.Inquirable.buildInquirableLink`** and be adapted

for the new framework. Implementers have a choice between using the convenience method **`org.kuali.rice.krad.uif.widget.Inquiry.buildInquiryLink`** to automatically build the link or define a completely new link by calling **`inquiry.getInquiryLink().setHref`**. manual conversion

InquiryAuthorizer

KRAD moves the functionality of **`InquiryAuthorizer`** to **`InquiryViewAuthorizer`**. While the data dictionary entry will be automatically converted, if it points to a custom implementation of **`InquiryAuthorizer`**, then the custom class will need to be reparented to **`InquiryViewAuthorizer`** and the methods will need to be migrated to **`canViewGroup`** (for sections) and **`canViewField`** (for fields). manual conversion

InquiryPresentationController

KRAD moves the functionality of **`InquiryPresentationController`** to **`InquiryViewPresentationController`**. While the data dictionary entry will be automatically converted, if it points to a custom implementation of **`InquiryPresentationController`**, then the custom class will need to be reparented to **`InquiryViewPresentationController`** and the methods will need to be migrated to **`canViewGroup`** (for sections) and **`canViewField`** (for fields).
manual conversion

ModuleService / RemoteModuleServiceBase / ModuleServiceBase

KRAD has both added and deprecated methods in this interface and its base classes.

- **`getExternalizableBusinessObjectInquiryUrl`** has been moved to **`getExternalizableDataObjectInquiryUrl`**. Helper methods of **`getExternalizableBusinessObjectInquiryUrl`** (such as **`getInquiryUrl`** and **`getUrlParameters`**) have also been deprecated. Any custom implementer configuration in these methods should be folded directly into **`getExternalizableDataObjectInquiryUrl`**.
  manual conversion

Other Inquiry Features

- **Inquiry Header Links**

  It is now possible to add links in the header of the Inquiry page. This was supposed to be possible in the KNS but never worked. Thus, any links desired in the header of the Inquiry will have to be added manually. manual conversion

```
<property name="page.header.lowerGroup.items">
  <list merge="true">
    <bean parent="Uif-Link" p:href="http://www.kuali.org" p:linkText="Kuali Site"/>
  </list>
</property>
```

- **Section Permission Checks**

The viewability of sections in Inquiries are controlled by a KIM permission only when a **`componentSecurity`** is configured in the section of the Uif-InquiryView and the appropriate KIM permission exists. The conversion requires that any sections listed in an override of the method **`InquiryAuthorizer.getSecurePotentiallyHiddenSectionIds()`** will need to have these configurations added, and that new KRAD permissions are created. manual conversion

Inquiry View:

```
<property name="componentSecurity">
  <bean parent="Uif-CollectionGroupSecurity" p:viewAuthz="true"/>
</property>
```

KIM Configuration:

A permission which extends the **`KR-KRAD : View Group`** template will have to be created.

- **Field Permission Checks**

  The viewability of fields in Inquiries are controlled by a KIM permission only when either

  - **`attributeSecurity`** is configured in the DD field, or
  - **`componentSecurity`** is configured in the field of the InquiryView

  and the appropriate KIM permission exists. The conversion requires that new KRAD permissions are created. manual conversion

  Data Dictionary:

```
<property name="attributeSecurity">
  <bean parent="AttributeSecurity" p:hide="true"/>
</property>
```

KIM Configuration:

A permission which extends the **`KR-KRAD : View Field`** template will have to be created.

Lookup Conversion

<add overview here>

LookupDefinition in KNS Data Dictionary to LookupView in KRAD

Note

The code example is very inclusive to show the configurations that have changed. As a result the lookups themselves have conflicting settings (i.e. why have custom search buttons when hiding them).

KNS Code example:

```
<bean id="EntityType-lookupDefinition" parent="LookupDefinition">
  <property name="title" value="Entity Type Lookup"/>
  <property name="menubar" value="&lt;a href=&quot;javascript:void(0)&quot;
              onclick=&quot;alert('JavaScript triggered action.')&quot;&gt;Custom Button&lt;/a&gt;"/>
  <property name="numOfColumns" value="2"/>
  <property name="extraButtonSource" value="images/tinybutton-createnew.gif"/>
  <property name="extraButtonParams" value="createNew"/>
  <property name="disableSearchButtons" value="true"/>
  <property name="lookupFields">
    <list>
      <bean parent="FieldDefinition" p:attributeName="code"/>
      <bean parent="FieldDefinition" p:attributeName="name" p:noLookup="true"
                                        p:treatWildcardsAndOperatorsAsLiteral="true"/>
      <bean parent="FieldDefinition" p:attributeName="active" p:defaultValue="Y"/>
    </list>
  </property>
  <property name="resultFields">
    <list>
      <bean parent="FieldDefinition" p:attributeName="code" p:triggerOnChange="true"/>
      <bean parent="FieldDefinition" p:attributeName="name" p:noLookup="true"/>
      <bean parent="FieldDefinition" p:attributeName="sortCode" p:forceInquiry="true"/>
      <bean parent="FieldDefinition" p:attributeName="amount" p:total="true"/>
      <bean parent="FieldDefinition" p:attributeName="active"/>
    </list>
  </property>
  <property name="defaultSort">
    <bean parent="SortDefinition">
      <property name="sortAscending" value="false"/>
      <property name="attributeNames">
        <list>
          <value>code</value>
        </list>
      </property>
    </bean>
  </property>
  <property name="translateCodes" value="true"/>
</bean>
```

KRAD code example:

```
<bean id="EntityTypeLookupView" parent="Uif-LookupView">
<property name="dataObjectClassName" value="org.kuali.rice.kim.impl.identity.EntityTypeBo"/>
  <property name="headerText" value="Entity Type Lookup" />
  <property name="page.header.lowerGroup.items">
    <list merge="true">
      <bean parent="Uif-SecondaryActionButton" p:actionLabel="Custom Button"
              p:actionScript="alert('JavaScript triggered action.')"/>
    </list>
  </property>
  <property name="renderLookupCriteria" value="false"/>
  <property name="criteriaGroup.layoutManager.numberOfColumns" value="4"/>
  <property name="criteriaGroup.footer">
    <bean parent="Uif-LookupCriteriaFooter">
      <property name="items">
        <list merge="true">
          <bean parent="Uif-PrimaryActionButton" p:methodToCall="createNew" p:actionLabel="create new"/>
        </list>
      </property>
    </bean>
  </property>
  <property name="renderCriteriaActions" value="false"/>
  <property name="criteriaFields">
    <list>
      <bean parent="Uif-LookupCriteriaInputField" p:propertyName="code"/>
      <bean parent="Uif-LookupCriteriaInputField" p:propertyName="name" p:enableAutoQuickfinder="false"
                                                    p:disableWildcardsAndOperators="true"/>
      <bean parent="Uif-LookupCriteriaInputField" p:propertyName="active" p:defaultValue="Y"/>
    </list>
  </property>
  <property name="resultFields">
    <list>
      <bean parent="Uif-DataField" p:propertyName="code"/>
      <bean parent="Uif-DataField" p:propertyName="name"/>
```

```
34        <bean parent="Uif-DataField" p:propertyName="sortCode" p:enableAutoInquiry="false"/>
35        <bean parent="Uif-DataField" p:propertyName="amount"/>
36        <bean parent="Uif-DataField" p:propertyName="active"/>
37      </list>
38    </property>
39    <property name="defaultSortAscending" value="false"/>
40    <property name="defaultSortAttributeNames">
41      <list>
42        <value>code</value>
43      </list>
44    </property>
45    <property name="resultsGroup.layoutManager.columnCalculations">
46      <list>
47        <bean parent="Uif-ColumnCalculationInfo-Sum" p:propertyName="amount" />
48      </list>
49    </property>
50    <property name="translateCodesOnReadOnlyDisplay" value="true"/>
51    <property name="multipleValuesSelectResultSetLimit" value="100"/>
52    <property name="resultSetLimit" value="200"/>
53  </bean>
```

Base Lookup Bean

- Lookups are now defined via **`Uif-LookupView`** definitions instead of the KNS **`LookupDefinition`** beans. [KNS line: 1, KRAD line: 1] script conversion
- The title of the view is specified via the **`headerText`** property instead of the **`title`** property. [KNS line: 2, KRAD line: 3] script conversion
- The data object class is specified via the **`dataObjectClassName`** property and is no longer specified via the **`BusinessObjectEntry`** as it doesn't exist in KRAD anymore. A **`businessObject`** is no longer required and any object can be used. [KNS line: n/a, KRAD line: 2] script conversion
- Instead of specifying the supplemental menu bar via the **`menubar`** property, the **`page.header.lowerGroup.items`** property is used. [KNS line: 3-4, KRAD line: 4-9] script conversion
- **`multipleValuesSelectResultSetLimit`** [KRAD line 51] overrides the MULTIPLE_VALUE_RESULTS_LIMIT application/system parameter. This parameter restricts the number of results returned from a lookup that can return multiple values. The parameter defaults to 100 results, but can either be changed in the database or be overridden by setting **`multipleValuesSelectResultSetLimit`**. manual conversion
- **`resultSetLimit`** [KRAD line 52] overrides the application/system parameter RESULTS_LIMIT. This parameter restricts the number of results returned from a lookup that can return one value. The parameter defaults to 100 results, but can either be changed in the database or be overridden by setting **`resultSetLimit`**. manual conversion

Criteria Fields

- The property that contains the lookup criteria fields was renamed from **`lookupFields`** to **`criteriaFields`**. [KNS line: 9, KRAD line: 22] script conversion
- Criteria fields changed from **`FieldDefinition`** to **`Uif-LookupCriteriaInputField`**. [KNS line: 11-14, KRAD line: 24-27] script conversion

- The **`attributeName`** property on the field is now a **`propertyName`** property. [KNS line: 11-14, KRAD line: 24-27] script conversion
- Not rendering a quickfinder on a lookup criteria field is specified via the **`enableAutoQuickfinder`** property instead of the **`noLookup`** property. [KNS line: 12, KRAD line 25] script conversion
- Adding additional buttons to the bottom of the search criteria is done by adding a **`Uif-PrimaryActionButton`** to the **`criteriaGroup.footer`** item list instead of using the **`extraButtonSource`** and **`extraButtonParms`** properties. [KNS line: 6-7, KRAD line: 12-20] script conversion
- The property that specifies whether or not to show the search buttons changed from **`disableSearchButtons`** to **`renderCriteriaActions`**. [KNS line: 8, KRAD line: 21] script conversion
- The rendering of search criteria was suppressed in KNS by adding the **`searchCriteriaEnabled=false`** parameter to the URL. In KRAD the URL parameter is renamed to **`renderLookupCriteria`**. Alternatively the **`renderLookupCriteria`** can be set on the Uif-LookupView. [KNS line: n/a, KRAD line: 10] manual conversion
- To use multiple columns for the criteria fields use the **`criteriaGroup.layoutManager.numberOfColumns`** property to configure the layout manager instead of specifying the **`numOfColumns`** property. Note that in the layout manager the field label and the field itself have their own columns and therefore the old **`numOfColumns`** value needs to be doubled. [KNS line: 5, KRAD line: 11] script conversion
- The **`treatWildcardsAndOperatorsAsLiteral`** property of **`FieldDefinition`** changed to **`disableWildcardsAndOperators`** on **`Uif-LookupCriteriaInputField`**. [KNS line: 13, KRAD line: 26] script conversion
- A default value can be specified in KNS by adding the **`<attribute-name>`** parameter to the URL. In KRAD the URL parameter naming convention changed to **`lookupCriteria[<attribute-name>]`**. Alternatively the defaultValue can be set on the Uif-LookupCriteriaInputField. [KNS line: n/a, KRAD line: 14] manual conversion

  KNS example:

```
1 http://demo.rice.kuali.org/kr/lookup.do?methodToCall=start
2   &businessObjectClassName=org.kuali.rice.location.impl.country.CountryBo&code=us
```

  KRAD example:

```
1 http://demo.rice.kuali.org/kr/lookup.do?methodToCall=start
2   &businessObjectClassName=org.kuali.rice.location.impl.country.CountryBo&lookupCriteria[code]=us
```

Results Fields

- Result fields changed from **`FieldDefinition`** to **`Uif-DataField`**. [KNS line: 19-23, KRAD line: 32-36] script conversion

- The **attributeName** property on the field is now a **propertyName** property. [KNS line: 19-23, KRAD line: 32-36] script conversion
- The **forceInquiry** property does not need to be set as inquiry links are rendered automatically. Set the **enableAutoInquiry** property on the **Uif-DataField** to false to suppress the rendering of the inquiry link. [KNS line: 21, KRAD line: 34] manual conversion
- Custom actions in the result rows are no longer specified by overriding the **getCustomActionUrls** method of **LookupableHelperService**. Instead the **resultsGroup.lineActions** property list is extended or overridden. Look in **UifLookupDefinition.xml** to see how the edit, copy and delete actions are defined. All types of **Components** (not just **Actions**) may be defined to appear as results group line actions. manual conversion
- The **SortDefinition** bean with the **sortAscending** and **attributeName** properties has been replaced. The sort order is specified via the **defaultSortAscending** property and the sort fields via the **defaultSortAttributeName** property list on the **Uif-Lookup-View**. [KNS line: 26-35, KRAD line: 39-44] script conversion
- Column totaling is now specified with the **Uif-ColumnCalculationInfo-Sum** property on the layout manager instead via the **total** property on the field definition. [KNS line: 22, KRAD line: 45-49] script conversion
- To display the maintenance links (create new, edit, copy, delete) on non maintenance documents the **renderMaintenanceLinks** on the LookupForm needs to be set. Not however the framework has logic based on whether a quickfinder is used to determine whether to show the return links or maintenance links. manual conversion
- For automatic translation of code fields in lookup the **translateCodes** of the **LookupDefinition** was set. In KRAD the automatic translation is enabled by setting the **translateCodesOnReadOnlyDisplay** of the **Uif-LookupView**. [KNS line: 36, KRAD line: 50] script conversion
- All types of **Components** (not just **Actions**) may be defined to appear in the collection add line actions, via the **resultsGroup.addLineActions** property list manual conversion

Lookupable / LookupableHelper Service

KRAD combines the **Lookupable** and **LookupableHelperService** from KNS. The **LookupableImpl** does not need to be extended unless advanced customizations are required.

Lookupable

- **set/getBusinessObjectClass** have been renamed to set/getDataObjectClass. Since now all data objects are supported the requirement that the object is a BusinessObject has been removed. The rename of these methods reflect this change. However, don't use this method. In KRAD the data object is specified via the dataObjectClassName property of the extended Uif-LookupView bean.

```
1 <bean id="Sample-LookupView" parent="Uif-LookupView">
```

```
  <property name="dataObjectClassName" value="org.kuali.rice.SampleBo" />
  ...
</bean>
```

The KNS2KRAD conversion script adds this property based on the dataObjectClass property of the MaintenanceDocumentEntry in KNS. script conversion

- **`getHtmlMenuBar/getSupplementalMenuBar`** have been removed. Instead these menu bars are configured via Uif. The HtmlMenuBar used to add additional HTML content to the right of the "Create New" button, while the SupplementalMenuBar replaces the "Create New" button with the specified HTML content. Note that with KRAD the "Create New" has been moved out of the lookup header area and instead is right below the header, still at the right side. The following sample displays how to add content after the "Create New". In this example a Message is used but any Uif component could be used. Append a custom button after the "Create New":

```
<bean id="Sample-LookupView" parent="Uif-LookupView">
  ...
  <property name="page.header.lowerGroup.items">
    <list>
      <bean parent="Uif-CreateNewLink" />
      <bean parent="Uif-SecondaryActionButton" p:actionLabel="Custom Button"
            p:actionScript="alert('JavaScript triggered action.')"/>
    </list>
  </property>
  ...
</bean>
```

Append a custom message to the right in the header

```
<bean id="Sample-LookupView" parent="Uif-LookupView">
  ...
  <property name="header.rightGroup">
    <bean parent="Uif-HeaderRightGroup">
      <property name="items">
        <list>
          <bean parent="Uif-Message" p:messageText="Right Group of headerText"/>
        </list>
      </property
    </bean>
  </property>
  ...
</bean>
```

Instead of "rightGroup" the "upperGroup" and "lowerGroup" can be used to position components above and below the header.

- **`getRows`** has been removed. Criteria fields can be conditionally displayed and configured via Uif.
- **`getColumns`** has been removed. The result columns are now specified via the Uif-LookupView (see resultFields property).

```
<bean id="Sample-LookupView" parent="Uif-LookupView">
  ...
  <property name="resultFields">
    <list>
      <bean parent="Uif-DataField" p:propertyName="cd" />
      <bean parent="Uif-DataField" p:propertyName="description" />
    </list>
  </property>
  ...
</bean>
```

- **`validateSearchParameters`** takes the LookupForm as an additional parameter and returns a boolean value indicating that no validation error message has occurred (true = no error). Warning and informational messages do not affect this return indicator. A ValidationException is no longer thrown. Try using Uif configurations for more complex validation that the default KRAD validation can't handle (e.g. Constraints)

- **`performLookup`** has been renamed to performSearch. The resultTable parameter has been removed and takes searchCriteria as an additional parameter.

- **`getSearchResults`** and **`getSearchResultsUnbounded`** have been removed. Override *executeSearch* for implementing a custom search routine.

- **`performClear`** takes the **`searchCriteria`** map as an additional parameter since it is no longer stored in the Lookupable. The **`searchCriteria`** map is returned after clearing the criteria and setting their default values.

- **`getReturnUrl`** has been renamed to *buildReturnUrlForResult* which accepts the Link that returns the result from the lookup and the model. Configuration via the Uif-LookupView is also possible:

```
1 <bean id="Sample-LookupView" parent="Uif-LookupView">
2   ...
3   <property name="resultsReturnField">
4     <bean parent="Uif-LinkField" p:href="http://www.kuali.org" p:linkText="Kuali"/>
5   </property>
6   ...
7 </bean>
```

- **`getCreateNewUrl`** has been removed. The create new url is now specified via the Uif-LookupView:

```
1 <bean id="Sample-LookupView" parent="Uif-LookupView">
2   ...
3   <property name="page.header.lowerGroup.items">
4     <list>
5       <bean parent="Uif-Link" p:linkText="Create New" p:href="http://www.kuali.org">
6         <property name="cssClasses">
7           <list merge="true">
8             <value>uif-createNewLink</value>
9           </list>
10        </property>
11      </bean>
12    </list>
13  </property>
14  ...
15 </bean>
```

- **`getTitle`** has been removed. The title is now specified via the Uif-LookupView (see headerText property).

```
1 <bean id="Sample-LookupView" parent="Uif-LookupView">
2   <property name="headerText" value="Sample Lookup" />
3   ...
4 </bean>
```

- **`getReturnKeys`** takes the **`LookupView, LookupForm`** and the data object as an additional parameter.

- **`getReturnLocation`** has been removed. The return location is now stored on the form. Use **`LookupForm.getReturnLocation.`**

- **`getExtraButtonSource`** has been removed. Buttons are configured via **`Uif-LookupCriteriaGroup`** (see footer property).
- **`getExtraButtonParams`** has been removed. Buttons are configured via **`Uif-LookupCriteriaGroup`** (see footer property).
- **`checkForAdditionalFields`** has been removed. Use progressive disclosure of the Uif.
- **`getDefaultSortColumns`** has been removed. Sort columns are configured via Uif-LookupView (see defaultSortAttributeNames property).

```
<bean id="Sample-LookupView" parent="Uif-LookupView">
  ...
  <property name="defaultSortAttributeNames">
    <list>
      <value>cd</value>
      <value>description</value>
    </list>
  </property>
  ...
</bean>
```

- **`set/getDocFormKey`** has been removed. The document form key is stored on the form. Use **`LookupForm.getFormKey`**.
- **`setFieldConversions`** has been removed. The field conversion is specified via Uif (see quickfinder.fieldConversions property).

```
<bean parent="Uif-InputField">
  ...
  <property name="quickfinder.fieldConversions">
    <map>
      <entry key="cd" value="sampleCd" />
      <entry key="description" value="sample.description" />
    </map>
  </property>
  ...
</bean>
```

- **`setReadOnlyFieldsList`** has been removed. Read only criteria fields are specified via Uif (see **`readOnly`** property).

```
<bean id="Sample-LookupView" parent="Uif-LookupView">
  ...
  <property name="criteriaFields">
    <list>
      <bean parent="Uif-LookupCriteriaInputField" p:propertyName="namespace"
            p:readOnly="true" />
      <bean parent="Uif-LookupCriteriaInputField" p:propertyName="cd" />
    </list>
  </property>
  ...
</bean>
```

- **`set/getLookupableHelperService`** has been removed. Lookupable helper services are specified via **`Uif-LookupView`**:

```
<bean id="Sample-LookupView" parent="Uif-LookupView">
  <property name="viewHelperServiceClass"
            value="org.kuali.rice.SampleLookupableHelperServiceImpl" />
```

```
4   ...
5 </bean>
```

- **`isSearchUsingOnlyPrimaryKeyValues`** has been removed.
- **`getPrimaryKeyFieldLabels`** has been removed.
- **`shouldDisplayHeaderNonMaintActions`** has been removed.
- **`shouldDisplayLookupCriteria`** has been removed.
- **`performCustomAction`** has been removed. Create **`methodToCall`** methods for the actions in the controller for the data object.
- **`getExtraField`** has been removed (see getHtmlMenuBar/getSupplementalMenuBar above).
- **`get/setExtraOnLoad`** has been removed. OnLoad scripts can be specified via the Uif-LookupView:

```
1 <bean id="Sample-LookupView" parent="Uif-LookupView">
2   <property name="onLoadScript" value="alert('Hi!')" />
3   ...
4 </bean>
```

- **`applyFieldAuthorizationsFromNestedLookups`** has been removed. Conditional masking of result field is done through Uif configuration.

```
1 <bean id="Sample-LookupView" parent="Uif-LookupView">
2   ...
3   <property name="resultFields">
4     <list>
5       <bean parent="Uif-DataField" p:propertyName="cd">
6         <property name="componentSecurity">
7           <bean parent="Uif-DataFieldSecurity">
8             <property name="attributeSecurity">
9               <bean parent="AttributeSecurity">
10                <property name="mask" value="true"/>
11                <property name="maskFormatter">
12                  <bean parent="MaskFormatterLiteral" p:literal="*********"/>
13                </property>
14              </bean>
15            </property>
16          </bean>
17        </property>
18      </bean>
19      ...
20    </list>
21  </property>
22  ...
23 </bean>
```

- **`applyConditionalLogicForFieldDisplay`** has been removed. Conditional displaying of criteria field is done through Uif configuration.

```
1 <bean id="Sample-LookupView" parent="Uif-LookupView">
2   ...
3   <property name="criteriaFields">
4     <list>
5       <bean parent="Uif-LookupCriteriaInputField" p:propertyName="cd"
6             p:readOnly="@{!#empty(#dp.lookupCriteria['cd'])}" />
7       <bean parent="Uif-DataField" p:propertyName="description"
8             p:required="@{#dp.lookupCriteria['CD'] == 'A_CD'}" />
```

```
 9       <bean parent="Uif-LookupCriteriaInputField" p:propertyName="namespace"
10             p:render="@{#dp.lookupCriteria['CD'] == 'A_CD'}" />
11     </list>
12   </property>
13   ...
14 </bean>
```

LookupableHelper

The functionality of **`LookupableHelper`** can now be found in the **`LookupableImpl`** class.

- The method **`allowsMaintenanceNewOrCopyAction`**, **`allowsMaintenanceEditAction`**, and **`allowsMaintenanceDeleteAction`** remain the same.
- The functionality of **`getActionUrl`** and **`getMaintenanceUrl`** is handled by **`getMaintenanceActionLink`**.
- The method **`getSearchResults`** now accepts the form, search criteria and the unbounded indicator.

QuickFinder Specifics

- The **`overrideLookupClass`** and **`overrideFieldConversions`** don't exist in KRAD since the quickfinder can be directly configured via the **`quickfinder.dataObjectClassName`** and **`quickfinder.fieldConversions properties`** script conversion

  KNS code example:

```
1 <bean parent="MaintainableFieldDefinition" p:name="entityTypeCode"/>
2   <property name="overrideLookupClass"
3             value="org.kuali.rice.kim.impl.identity.EntityTypeBo"/>
4   <property name="overrideFieldConversions">
5     <map>
6       <entry key="code" value="entityTypeCode"/>
7     </map>
8   </property>
9 </bean>
```

  KRAD code example:

```
1 <bean parent="Uif-InputField" p:propertyName="entityTypeCode">
2   <property name="quickfinder.dataObjectClassName"
3             value="org.kuali.rice.kim.impl.identity.EntityTypeBo"/>
4   <property name="quickfinder.fieldConversions">
5     <map>
6       <entry key="code" value="entityTypeCode"/>
7     </map>
8   </property>
9 </bean>
```

- The **`searchIconOverride`** doesn't exist in KRAD since the quickfinder can be directly configured via the **`quickfinder.quickfinderAction.actionImage.source`**. Consequently, it was really not intended to be defined in the XML but rather in the JSP code for a transactional document. KRAD allows for much more flexibility and thus it can be specified as well in the lookup. manual conversion

  KRAD code example:

```
1 <bean parent="Uif-LookupCriteriaInputField" p:propertyName="entityTypeCode">
2   <property name="quickfinder.quickfinderAction.actionImage.source" value="icon.png"/>
3 </bean>
```

Other Lookup Features

- **Custom Forms**

  In KNS custom forms are specified in the struts-config.xml file.

```
<form-bean name="TravelAuthorizationForm"
           type="org.kuali.rice.kns.demo.travel.authorization.TravelAuthorizationForm"/>
```

  With KRAD the forms are specified as a property of the view.

```
<bean parent="Uif-TransactionalDocumentView">
    <property name="formClass"
              value="edu.sampleu.travel.TravelAuthorizationForm"/>
    ...
</bean>
```

- **Help Items**

  Help items can be displayed in the same manner in KRAD as in the KNS. script conversion

  KNS code example:

```
1 <property name="helpDefinition">
2   <bean parent="HelpDefinition" p:parameterNamespace="KR-KRAD" p:parameterDetailType="Lookup"
3                                 p:parameterName="DEFAULT_HELP_URL"/>
4 </property>
```

  or

```
1 <property name="helpUrl" value="link.html"/>
```

  KRAD code example:

```
1 <property name="help">
2   <bean parent="Uif-Help">
3     <property name="helpDefinition">
4       <bean parent="HelpDefinition" p:parameterNamespace="KR-KRAD" p:parameterDetailType="Lookup"
5                                     p:parameterName="DEFAULT_HELP_URL"/>
6     </property>
7   </bean>
8 </property>
```

  or

```
1 <property name="help">
2   <bean parent="Uif-Help" p:externalHelpUrl="link.html"/>
3 </property>
```

Maintenance Conversion

<add overview here>

MaintenanceDefinition in KNS Data Dictionary to MaintenanceView in KRAD

KNS Code example:

```
<bean id="EntityTypeMaintenanceDocument" parent="MaintenanceDocumentEntry">
  <property name="dataObjectClass" value="org.kuali.rice.kim.impl.identity.EntityTypeBo"/>
  <property name="documentTypeName" value="EntityTypeMaintenanceDocument"/>
  <property name="maintainableClass" value="org.kuali.rice.kim.maintenance.EntityTypeMaintainableImpl"/>

  <property name="webScriptFiles">
    <list>
      <value>entityType.js</value>
    </list>
  </property>

  <property name="maintainableSections">
    <list>
      <ref bean="EntityTypeMaintenanceDocument-General"/>
      <ref bean="EntityTypeMaintenanceDocument-Details"/>
    </list>
   </property>
</bean>

<bean id="EntityTypeMaintenanceDocument-General" parent="MaintainableSectionDefinition">
  <property name="title" value="Entity Type"/>
  <property name="defaultOpen" value="true"/>
  <property name="helpUrl" value="docs/entityType.html"/>
  <property name="maintainableItems">
    <list>
      <bean parent="MaintainableFieldDefinition" p:name="code"/>
      <bean parent="MaintainableFieldDefinition" p:required="true" p:name="name"/>
      <bean parent="MaintainableFieldDefinition" p:required="true" p:name="active"/>
    </list>
  </property>
</bean>

<bean id="EntityTypeMaintenanceDocument-Details" parent="MaintainableSectionDefinition">
  <property name="title" value="Entity Type Details"/>
  <property name="defaultOpen" value="false"/>
  <property name="helpUrl" value="docs/entityTypeDetails.html"/>
  <property name="maintainableItems">
    <list>
      <bean parent="MaintainableCollectionDefinition">
        <property name="summaryTitle" value="Entity Type Detail"/>
        <property name="businessObjectClass" value="org.kuali.rice.kim.impl.identity.EntityTypeDetailsBo"/>
        <property name="sourceClassName" value="org.kuali.rice.kim.impl.identity.EntityTypeDetailsBo"/>
        <property name="name" value="entityTypeDetails"/>
        <property name="maintainableFields">
          <list>
            <bean parent="MaintainableFieldDefinition" p:name="name"/>
            <bean parent="MaintainableFieldDefinition" p:name="value"/>
          </list>
        </property>
      </bean>
    </list>
  </property>
</bean>
```

KRAD code example:

```
<bean id="EntityTypeMaintenanceDocument" parent="uifMaintenanceDocumentEntry">
  <property name="dataObjectClass" value="org.kuali.rice.krad.demo.travel.dataobject.TravelAccount"/>
```

```
 3   <property name="documentTypeName" value="TravelAccountMaintenanceDocument"/>
 4   <property name="maintainableClass" value="org.kuali.rice.kim.maintenance.EntityTypeMaintainableImpl"/>
 5 </bean>
 6
 7 <bean id="EntityType-MaintenanceView" parent="Uif-MaintenanceView">
 8   <property name="dataObjectClassName" value="org.kuali.rice.kim.impl.identity.EntityTypeBo"/>
 9   <property name="additionalScriptFiles">
10     <list>
11       <value>entityType.js</value>
12     </list>
13   </property>
14
15   <property name="items">
16     <list>
17       <ref bean="EntityType-MaintenanceView-General"/>
18       <ref bean="EntityType-MaintenanceView-Details"/>
19     </list>
20   </property>
21 </bean>
22
23 <bean id="EntityType-MaintenanceView-General" parent="Uif-MaintenanceGridSection">
24   <property name="headerText" value="Entity Type"/>
25   <property name="disclosure.defaultOpen" value="true"/>
26   <property name="help">
27     <bean parent="Uif-Help">
28       <property name="externalHelpUrl" value="docs/entityType.html"/>
29     </bean>
30   </property>
31   <property name="items">
32     <list>
33       <bean parent="Uif-InputField" p:propertyName="code"/>
34       <bean parent="Uif-InputField" p:propertyName="name"/>
35       <bean parent="Uif-InputField" p:propertyName="active"/>
36     </list>
37   </property>
38 </bean>
39
40 <bean id="EntityType-MaintenanceView-Details" parent="Uif-Disclosure-StackedCollectionSection">
41   <property name="headerText" value="Entity Type Details"/>
42   <property name="disclosure.defaultOpen" value="false"/>
43   <property name="collectionObjectClass" value="org.kuali.rice.kim.impl.identity.EntityTypeDetailsBo"/>
44   <property name="propertyName" value="entityTypeDetails"/>
45   <property name="help">
46     <bean parent="Uif-Help">
47       <property name="externalHelpUrl" value="docs/entityTypeDetails.html"/>
48     </bean>
49   </property>
50   <property name="items">
51     <list>
52       <bean parent="Uif-DataField" p:attributeName="name"/>
53       <bean parent="Uif-DataField" p:attributeName="value"/>
54     </list>
55   </property>
56 </bean>
```

Base Maintenance Bean

- The parent bean of the maintenance document has changed from **`MaintenanceDocumentEntry`** to **`uifMaintenanceDocumentEntry`**. [KNS line: 1, KRAD line: 1] script conversion
- The maintainable class for a given document is specified in the same way in KRAD as was done in KNS using the **maintainableClass** property of MaintenanceDocumentEntry. The maintainable class should extend **org.kuali.rice.krad.maintenance.MaintainableImpl.** [KNS line: 4, KRAD line: 4] script conversion
- Maintenance sections are now referenced in a completely new bean parented by **`Uif-MaintenanceView`** instead of the KNS **`MaintainableSectionDefinition`** beans. Note that the sections are no longer referenced inside the **`uifMaintenanceDocumentEntry`** and instead

are referenced in the view and are linked back to the **`uifMaintenanceDocumentEntry`** via **`dataObjectClassName`** [KNS line: 12, KRAD line: 7] script conversion

- For including extra Javascript files, **`webScriptFiles`** has changed to **`additionalScriptFiles`**. [KNS line: 6-10, KRAD line: 9-13] script conversion

Sections

- The title of the view is specified via the **`headerText`** property instead of the **`title`** property. [KNS line: 21, KRAD line: 24] script conversion
- The property that controls whether the section defaults to open has moved from **`defaultOpen`** to the **`Disclosure`** object and is now accessed via **`disclosure.defaultOpen`**. [KNS line: 22, KRAD line: 25] script conversion
- Help links specified by **`helpUrl`** have now been changed to a full **`Uif-Help`** which contains a property **`externalHelpUrl`** where the link is now stored. [KNS line: 23, KRAD line: 26-30] script conversion

Data Dictionary Features

- In order to customize how the backing form populated after a lookup or refresh, the KNS provided **`DerivedValuesSetter.setDerivedValues`** which could update other objects on the form when one object was updated via the request. KRAD provides a Uif property to do this: **`p:refreshWhenChangedPropertyNames="field1, field2"`**. so that when an **`InputField`** specifying this property is updated, **`field1`** and **`field2`** are refreshed as well. manual conversion
- The KNS **`PromptBeforeValidation`** class was used to prompt the user with a question before continuing on with routing. This has been deprecated in KRAD in favor of the new Dialog framework. To implement this, first override the base Maintenance controller, call the dialog right before routing, and get the dialog response from **`getBooleanDialogResponse`**. Next in the Uif, add the dialog to the **`MaintenanceView.dialogs`** property.

```
 1 <property name="dialogs">
 2   <list>
 3     <bean id="routeConfirmationDialog" parent="Uif-OK-Cancel-DialogGroup" p:promptText="Would you like to
route?"/>
 4   </list>
 5 </property>
```

manual conversion

- The KNS **`LookupReadonlyControlDefinition`** which only had a quickfinder icon and no text field has been replaced with other fields on the generic **`InputField`**.

```
 1 <bean parent="Uif-InputField" p:propertyName="field" p:widgetInputOnly="true">
 2   <property name="quickfinder">
 3     <bean parent="Uif-QuickFinder"/>
 4   </property>
 5 </bean>
```

script conversion

- While default values can be specified via finders, they can also be specified directly on the Uif with complete expression support.

```
1 <bean parent="Uif-InputField" p:propertyName="field" p:defaultValue="@{field2}"/>
```

- Several fields have been removed in favor of the new conditional **`readOnly`**. The **`p:unconditionallyReadOnly="true"`** can be replaced directly by **`p:readOnly="true"`**, while **`p:readOnlyAfterAdd="true"`** can be replaced with **`p:readOnly="@{!#isAddLine}"`**. script conversion

- When the document is in read only mode, an alternate property can be displayed for a field. The KNS property **`p:alternateDisplayAttributeName="field"`** has been replaced with the KRAD equivalent **`p:readOnlyDisplayReplacementPropertyName="field"`**. script conversion

- When the document is in read only mode, an additional property can be displayed with a field. The KNS property **`p:additionalDisplayAttributeName="field"`** has been replaced with the KRAD equivalent **`p:readOnlyDisplaySuffixPropertyName="field"`**. script conversion

- In order to control whether the add line in a collection should be shown, the KNS provided **`<property name="includeAddLine" value="false"/>`**. The setup in KRAD is a bit more complex but a lot more flexible as it allows implementers to control the exactly when the line should be shown or not depending on an expression.

```
1 <bean parent="Uif-MaintenanceStackedCollectionSection">
2   <property name="addLineActions">
3     <list>
4       <bean parent="Uif-SecondaryActionButton-Small" p:methodToCall="addLine" p:actionLabel="add"
p:hidden="true"/>
5     </list>
6   </property>
7 </bean>
```

script conversion

- To control whether or not to have a multiple value lookup, the KNS provided the property **`p:includeMultipleLookupLine="false"`**. This has been replaced with the KRAD equivalent **`p:includeLineSelectionField="false"`**. script conversion

- To specify whether items can be deleted from a collection, the KNS provided the property **`p:alwaysAllowCollectionDeletion="true"`**. This has been removed in KRAD but has an equivalent replacement:

```
1 <property name="lineActions">
2   <list>
3     <bean parent="Uif-DeleteLineAction" p:render="@{isAddedCollectionItem(#line)}"/>
4     <bean parent="Uif-SaveLineAction"/>
5   </list>
6 </property>
```

script conversion

Maintenance Multiple Value Lookups

Multiple value lookups allow for a collection on a maintenance document to be populated with multiple results at once, rather than forcing the user to iteratively add results one after another. For a maintenance document to use a multiple value lookup, the maintenance document must contain a collection, and a collection lookup must be configured for that collection.

KNS code example:

```
1 <bean parent="MaintainableCollectionDefinition">
2   <property name="name" value="categories"/>
3   <property name="businessObjectClass" value="org.kuali.rice.krms.impl.repository.CategoryBo"/>
4   <property name="includeMultipleLookupLine" value="true"/>
5   ...
6 </bean>
```

KRAD code example:

```
 1 <bean parent="Uif-TableCollectionSection">
 2   <property name="collectionObjectClass" value="org.kuali.rice.krms.impl.repository.CategoryBo"/>
 3   <property name="propertyName" value="categories"/>
 4   <property name="collectionLookup">
 5     <bean parent="Uif-CollectionQuickFinder"
 6               p:dataObjectClassName="org.kuali.rice.krms.impl.repository.CategoryBo"
 7               p:fieldConversions="id:id,name:name,namespace:namespace"/>
 8   </property>
 9   ...
10 </bean>
```

- The in KNS the **`MaintainableCollectionDefinition`** is used to specify collections on a maintenance document. With KRAD one of the Uif collection groups is used. [KNS line: 1, KRAD line: 1]
- In KNS the **`includeMultipleLookupLine`** property was set to true to enable multiple value lookups on the collection. In KRAD this property does no longer exist, instead the **`collectionLookup`** property is initialized with the **`Uif-CollectionQuickfinder`** bean. **`Uif-CollectionQuickfinder`** has the following properties but do not need to be specified if the collectionObjectClass of the collection group is the same as the dataObjectClassName of the multi value lookup:
  - dataObjectClassName - the data object that should be looked up
  - fieldConversions - from-to mapping of the returned data (<lookup-field>:<maintenance-collection-field>)
  - viewName - name of the lookup view (only needed if multible views for the dataObjectClassName exists)

  [KNS line: , KRAD line: 4-8]

Base Maintenance Class

Custom documents that extended **`org.kuali.rice.kns.document.MaintenanceDocumentBase`** will need to

reparent to **`org.kuali.rice.krad.maintenance.MaintenanceDocumentBase`**. manual conversion

Section Customization

The KNS provided the hook **`Maintainable.getSections`** to allow tweaking the section objects to be read only or hidden. Now that KRAD provides dynamic fields on the Uif, this functionality is deprecated in favor of the dynamic fields. These should be used to configure when a given section is read only or hidden. An example of this would be to specify **`p:readOnly="@{#dp.propertyName eq null}"`**. manual conversion

Collection Customization

The KNS provided the container **`Maintainable.newCollectionLines`** for lines being added to a collection to prevent them from being persisted until added. KRAD provides a similar **`UifFormBase.newCollectionLines`** but also allows customizing the property names and **`BindingInfo`** via **`CollectionGroup.addLinePropertyName`** and **`CollectionGroup.addLineBindingInfo`**.

Rule Customization

The KNS provided methods for setting up convenience objects to process the rules. These methods, **`org.kuali.rice.kns.rules.MaintenanceDocumentRule.setupBaseConvenienceObjects`** and **`org.kuali.rice.kns.rules.MaintenanceDocumentRule.setupConvenienceObjects`** haved been reparented and can now be overridden in **`org.kuali.rice.krad.rules.MaintenanceDocumentRule.setupBaseConvenienceObjects`** and **`org.kuali.rice.krad.rules.MaintenanceDocumentRule.setupConvenienceObjects`**, respectively. manual conversion

MaintenanceDocumentPresentationController

KRAD splits the **`MaintenanceDocumentPresentationController`** interface into one for the maintenance entry and one for the maintenance view. The one for the maintenance entry is also named **`MaintenanceDocumentPresentationController`** but has been reparented. The one for the maintenance view is **`ViewPresentationController`**.

- Any references to **`getConditionallyHiddenSectionIds`** and **`getConditionallyReadOnlySectionIds`** in **`MaintenanceDocumentPresentationController`** should be moved to **`ViewPresentationController`** and translated to **`canViewGroup`** and **`canEditGroup`** respectively. manual conversion
- Sections can contain an ID attribute in KRAD just like they could in the KNS. However, if this ID was not specified in the KNS, it defaulted to the tab name. In KRAD, this defaults to the bean name, so implementers using this defaulting capability in the **`getConditionallyHiddenSectionIds`** and **`getConditionallyReadOnlySectionIds`** methods will need to change any references from the tab name to the bean name. manual conversion

Specifying a javascript function to call on tab out from a field

In KNS this was achieved using **MaintainableFieldDefinition.webUILeaveFieldFunction**.

```
 1 <bean parent="MaintainableFieldDefinition" p:name="name" p:required="false"
 2   p:webUILeaveFieldFunction="alert('you are out')"/>
```

However in KRAD Component classes include properties for events that are applicable to the generate HTML elements. The same functionality can be obtained using the **ComponentBase.onBlurScript**

```
<bean parent="Uif-TextAreaControl" p:rows="5" p:cols="40">
  <property name="onBlurScript" value="jq(this).stop().animate({'height':'-=50px'},100);"/>
</bean>
```

Specifying help URL on a per-field basis

In KNS **System parameter ENABLE_FIELD_LEVEL_HELP_IND enables help on all fields** enables help on all fields. You could set help at the field\BO\document page level. In KRAD the **Help widget** is used to render tooltip help and/or external help. Help can be configured at the view, page, section or sub-section, or at the field level. Tooltip help content is defined in the data dictionary.

```
 1 <bean parent="Uif-TextControl">
 2   <property name="help">
 3     <bean parent="Uif-Help" p:tooltipHelpContent="This is my help text"/>
 4   </property>
 5 </bean>
```

External Help can be specified via the data dictionary, or through a system parameter.

```
 1 <bean parent="Uif-View">
 2   <property name="help">
 3     <bean parent="Uif-Help" p:externalHelpUrl="http://www.kuali.org/"/>
 4   </property>
 5 </bean>
```

```
 1 <bean parent="Uif-View">
 2   <property name="help">
 3     <bean parent="Uif-Help">
 4       <property name="helpDefinition">
 5         <bean parent="HelpDefinition" p:parameterNamespace="KR-SAP"
 6                 p:parameterName="TEST_PARAM"
 7                 p:parameterDetailType="TEST_COMPONENT"/>
 8       </property>
 9     </bean>
10   </property>
11 </bean>
```

Additional display value

In KNS **LookupDefinition.translateCode FieldUtils.setAdditionalDisplayPropertyForCodes** gave the ability to indicate that any property which is the code on a relationship for a BO class that implements **KualiCode** should display itself as a combination of Code + Name in read-only mode. In KRAD this is done using **View.translateCodesOnReadOnlyDisplay**

and **DataField.setAlternateAndAdditionalDisplayValue translateCodesOnReadOnlyDisplay** is a boolean that indicates whether code properties should automatically be translated to their name property for read only display. **setAlternateAndAdditionalDisplayValue** get the relationship configured in the datadictionary file and set the name additional display value which will be displayed along with the code

Attachments

The KNS provided a way to download an attachment from a Maintenance Document. Typically this was specified both in the business object and the data dictionary as follows:

```
1 private String fileName;
2 private String contentType;
3 private byte[] attachmentContent;
4 private transient FormFile attachmentFile;
```

```
1 <bean id ="AttachmentSample-attachmentFile" parent="AttributeDefinition">
2   <property name="name" value="attachmentFile" />
3   <property name="label" value="Attachment File" />
4   <property name="control">
5     <bean parent="FileControlDefinition" p:size="50" />
6   </property>
7 </bean>
```

In KRAD, because of the transition to Spring, this configuration needs to be changed with some additional setup. The Struts **FormFile** should change to **MultipartFile** but should include some additional setup to make sure that all of the rest of the variables are set on the form. KRAD also provides an intelligent way to display the attachments depending on whether the attachment is null or not. When the attachment exists, then this setup allows downloading the attachment via a button. When the attachment is null, the **Uif-ConditionalBeanPropertyReplacer** replaces the button with a simple **Uif-FileControl**.

```
 1 private String fileName;
 2 private String contentType;
 3 private byte[] attachmentContent;
 4 private transient MultipartFile attachmentFile;
 5
 6 public void setAttachmentFile(MultipartFile attachmentFile) {
 7   if (attachmentFile != null) {
 8     setContentType(attachmentFile.getContentType());
 9     setFileName(attachmentFile.getOriginalFilename());
10     try {
11       setAttachmentContent(attachmentFile.getBytes());
12     } catch (Exception e) {
13       throw new RuntimeException(e);
14     }
15   }
16 }
```

```
 1 <bean parent="Uif-VerticalFieldGroup" p:label="Attachment File">
 2   <property name="items">
 3     <list>
 4       <bean parent="Uif-DataField" p:fieldLabel.render="false" p:propertyName="fileName" />
 5       <bean parent="Uif-PrimaryActionButton-Small" p:methodToCall="downloadBOAttachment"
 6               p:actionLabel="download attachment" p:title="download attachment"
 7               p:ajaxSubmit="false" p:disableBlocking="true"
 8               p:onClickScript="writeHiddenToForm("ajaxReturnType", "update-none");">
 9         <property name="actionParameters">
10           <map>
11             <entry key="selectedLineIndex" value="@{#index}"/>
12           </map>
13         </property>
14       </bean>
15     </list>
```

```
16   </property>
17   <property name="propertyReplacers">
18     <list>
19       <bean parent="Uif-ConditionalBeanPropertyReplacer" p:propertyName="items" p:condition="@{#dp.fileName
eq null}">
20         <property name="replacement">
21           <list>
22             <bean parent="Uif-InputField" p:propertyName="attachmentFile" p:fieldLabel.render="false">
23               <property name="control">
24                 <bean parent="Uif-FileControl" p:size="50"/>
25               </property>
26             </bean>
27           </list>
28         </property>
29       </bean>
30     </list>
31   </property>
32 </bean>
```

Overriding default values

In KNS **Maintainable.setGenerateDefaultValues** was used to set the default values for fields. However in KRAD this functionality is provided by **ViewHelperServiceImpl.applyDefaultValues**. This is called during the finalize phase of the view cycle before the view is rendered.

Customization Hooks

Customization hooks have been provided for the various actions that can be performed on a maintenance document. It involves overriding the specific methods related to that action (new,copy etc). KNS used to provide the following methods:

- MaintenanceDocument.getNewMaintainableObject
- Maintainable.setupNewFromExisting
- Maintainable.processBeforeAddLine
- Maintainable.processAfterPost

Similar functionality can be obtained in KRAD using the following methods:

- MaintenanceDocumentController.setupMaintenance
- Maintainable.setupNewFromExisting
- Maintainable.retrieveObjectForEditOrCopy
- Maintainable.processAfterPost
- ViewHelperServiceImpl.processBeforeAddLine
- ViewHelperServiceImpl.processAfterAddLine

Preparing the BO as part of document setup

In KNS the business object was set up as part document setup using the Maintainable.prepareBusinessObject. In KRAD this has been replaced by using MaintenanceDocumentService.setupMaintenanceObject.

KNS BO Preparation:

```
1 public void prepareBusinessObject(BusinessObject businessObject);
```

KRAD BO Preparation:

```
1 public void setupMaintenanceObject(MaintenanceDocument document,
2         String maintenanceAction, Map<String, String[]> requestParameters);
```

Preparing the BO from an external source

In KNS an external business object was set up using **org.kuali.rice.kns.maintenance.Maintainable.prepareBusinessObject**. In KRAD this functionality has been moved to **org.kuali.rice.krad.maintenance.Maintainable** and replaced. Different methods should be overridden depending on whether the external BO being loaded is from the database or not.

KNS BO Preparation:

```
1 public void prepareBusinessObject(BusinessObject businessObject);
```

KRAD Database BO Preparation:

```
1 public boolean isExternalBusinessObject();
2 public void prepareExternalBusinessObject(BusinessObject businessObject);
```

KRAD Non-Database BO Preparation:

```
1 public void retrieveObjectForEditOrCopy(MaintenanceDocument document, Map<String, String> dataObjectKeys);
```

Refreshing references

Specifying which reference objects to refresh on a document can be configured by using the *referencesToRefresh* property. It specifies primary keys of reference objects, which are then used to pull those objects from where they are persisted.

In KNS this was done using the AbstractLookupableHelperServiceImpl.referencesToRefresh and KualiMaintainableImpl.refreshReferences. In KRAD this has been replaced by LookupForm.referencesToRefresh and ViewHelperService.refreshReferences.

Permission Checks

Permission checks for maintenance documents in KRAD can be setup the same way as specified in the Other Lookup Features. Note that if any implementers are extending a KNS base class, they need to move that logic over to a KRAD base class. Examples below.

- Partial Unmask Field checks if a security-masked field can be partially displayed. In KNS this was checked using the *BusinessObjectAuthorizationServiceImpl.canPartiallyUnmaskField.* In KRAD the method name has changed to *ViewAuthorizer.canPartialUnmaskField.* Enabled by setting

attributeSecurity.partialMask to true on the data field security object of the data. Implementers will have manually do the conversion and move any custom logic in the new method.

- Full Unmask Field checks if a security-masked field can be displayed in the clear. In KNS this was checked using the *BusinessObjectAuthorizationServiceImpl.canFullyUnmaskField.* In KRAD the method name has changed to *ViewAuthorizer.canUnmaskField.* Enabled by setting *attributeSecurity.mask* to true on the data field security object of the data. Implementers will have manually do the conversion and move any custom logic in the new method.

- Modify Maintenance Document Section - In KNS this was accomplished using the *BusinessObjectAuthorizationServiceImpl.considerMaintenanceDocumentAuthorizer* and a permission extending the KR-NS *Modify Maintenance Document Section.* In KRAD the method has changed to *ViewAuthorizer.canEditGroup.* Any custom logic will need to be moved to the new method. A new permission which extends the KR-KRAD *Edit Group template* will have to be created.

- View Inquiry or Maintenance Document Section - In KNS this was accomplished using the *BusinessObjectAuthorizationServiceImpl.considerInquiryOrMaintenanceDocumentAuthorizer* and a permission extending the KR-NS *View Inquiry or Maintenance Document Section.* In KRAD the method has changed to *ViewAuthorizer.canViewGroup.* Any custom logic will need to be moved to the new method. A new permission which extends the KR-KRAD *View Group template* will have to be created.

- Modify Maintenance Document Field - In KNS this was accomplished using the *BusinessObjectAuthorizationServiceImpl.considerBusinessObjectFieldModifyAuthorization* and a permission extending the KR-NS *Modify Maintenance Document Field.* In KRAD the method has changed to *ViewAuthorizer.canEditField.* Any custom logic will need to be moved to the new method. A new permission which extends the KR-KRAD *Edit Field template* will have to be created.

- View Inquiry or Maintenance Document Field - In KNS this was accomplished using the *BusinessObjectAuthorizationServiceImpl.considerBusinessObjectFieldViewAuthorization* and a permission extending the KR-NS *View Inquiry or Maintenance Document Field.* In KRAD the method has changed to *ViewAuthorizer.canViewField.* Any custom logic will need to be moved to the new method. A new permission which extends the KR-KRAD *View Field template* will have to be created.

- Perform Custom Maintenance Document Function - In KNS this was accomplished using the *BusinessObjectAuthorizationServiceImpl.considerCustomButtonFieldAuthorization* and a permission extending the KR-NS *Perform Custom Maintenance Document Function.* In KRAD the logic has moved to the Uif under *Uif-ActionSecurity.* Any custom logic will need to be moved to the new definition below. A new permission which extends the KR-KRAD *Perform Action template* will have to be created.

```
1 <bean parent="Uif-ActionField" p:label="Button">
2   <property name="action">
3     <bean parent="Uif-PrimaryActionButton" p:id="button" p:actionLabel="Button" p:methodToCall="Button">
4       <property name="componentSecurity">
5         <bean parent="Uif-ActionSecurity" p:performActionAuthz="true" />
6       </property>
7     </bean>
8   </property>
9 </bean>
```

Document Error Handling

Document errors with custom paths, typically specified in a subclass of **MaintenanceDocumentRuleBase**, will not appear on the document unless they are converted to the new KRAD format. For example, if you were throwing an additional error against text entered for notes, in KNS you would write a validation like:

```
1 public boolean isValid(Note note) {
2   boolean isValid = true;
3   if (!StringUtils.isAllUpperCase(note.getNoteText())) {
4     isValid = false;
5     GlobalVariables.getMessageMap().putError("newNote.noteText", "error.note.not.uppercase");
6   }
7   return isValid;
8 }
```

but in KRAD, the beginning of the property path has changed to **newCollectionLines['document.notes']** so the code would have to be modified to

```
1 public boolean isValid(Note note) {
2   boolean isValid = true;
3   if (!StringUtils.isAllUpperCase(note.getNoteText())) {
4     isValid = false;
5     GlobalVariables.getMessageMap().putError("newCollectionLines['document.notes'].noteText",
"error.note.not.uppercase");
6   }
7   return isValid;
8 }
```

Validation of all of these customizations is left to the implementer.

Other Maintenance Features

- Where it used to be possible to specify additional non-standard sections via the Data Dictionary property **`additionalSectionsFile`**, this is no longer possible since KRAD does not use JSP, preferring instead to use Freemarker. The conversion from JSP files to the KRAD format will vary from use case to use case, and thus it is not possible to provide a straight conversion from JSP to Freemarker. KRAD does however address natively many of the typical non-standard layouts previously addressed by this feature, and implementers can always choose to create custom components to address any other requirements. manual conversion
- To enable Externalizable Business Objects, users of the KNS would often override the indicator **`org.kuali.rice.kns.maintenance.Maintainable.isExternalBusinessObject`**. In KRAD, this has been moved to **`org.kuali.rice.krad.maintenance.Maintainable.isExternalBusinessObject`**. If this has been implemented in an old KNS class, then the logic must be moved to the new KRAD class. manual conversion

Transactional Conversion

<introduction here>

KIM Permission Conversion

Note

Currently there is no conversion script to automatically convert or create new KRAD KIM permissions from KNS permissions. This section of the guide documents the KIM structures involved and the KNS / KRAD equivalents. To illustrate the various KIM entities, examples from the Rice KIM management screens are shown.

KRAD uses KIM permissions for checking the ability to open views, view and edit groups, lines and fields, as well as the ability to perform actions. In several instances the KRAD permissions are similar to their KNS counterparts, but use different permissions templates and qualifiers. But there are some differences:

- In KNS the permission was checked against the business object. In KRAD, the permission is checked against the view name. And you may have multiple views on the same object with different permissions.
- In KRAD a view is viewable by all authenticated users by default. In this case, no KIM permissions need to be defined. In KNS, a default KIM permission of the appropriate type is required, with the derived role USER assigned to it.

KIM permissions are stored in Rice database tables. They are created either by SQL script or using the Rice KIM maintenance screens.

Inquiry Permissions

In KNS, there are three types of KIM permissions related to Inquiries

- Inquire Into Records - may the user inquire on objects of that type.
- Full Unmask Field - may a security masked field be displayed in the clear.
- Partial Unmask Field - may a security masked field be partially displayed.

Inquire Into Records

Checks the user's ability to inquiry objects of that type. In KRAD the **`Open View`** permission is used in place of the KNS **`Inquire Into Records`** permission.

The KRAD Can View permission uses different KIM entities than the KRAD Inquire Into Records permission.

- KIM Permission

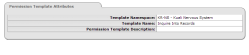
  KNS Inquire Into Records Permission

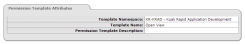
  The following example illustrates the differences between a KNS Inquiry Into Records permission and a KRAD Open VIew permission. 

  KRAD Open View Permission 

  - In KRAD **`Open View`** permissions are used in place of the KNS **`Inquire Into Records`** permissions to limit access to an inquiry view.
  - In KRAD the **`Open View`** permission has a single Detail attribute: **`viewId`**. This is in contrast to the two permission detail attributes **`namespaceCode`** and **`componentName`** used by the KNS **`Inquire Into Records`** permissions to limit access to an inquiry view.

- KIM Permission Template: Inquire Into Records

  KNS Namespace or Component KimType: 

  KRAD View KimType: 

In KRAD the **`Open View`** permission template is used in place of the KNS **`Inquire Into Records`** permission template.

- KIM Type: Namespace or Component

KNS Inquire Into Records Permission Template: 

KRAD Open View Permission Template:

- In KRAD the **`Open View`** permission template is used in place of the KNS **`Inquire Into Records`** permission template.

- KIM Roles - the same roles can be applied to either KNS or KRAD permissions. It is not necessary to create separate KRAD equivalent roles.

To Restrict an Inquiry to users assigned a supervisor role in KRAD:

1. Create a new KIM Permission Using the KR-KRAD Open View Template

 The Open View permission has one Permission Detail attribute: **`viewID`**. Set the viewId to be the name of the Inquiry View defined for the data object. [Line 1 below]

```
 1 <bean id="AgendaBo-InquiryView" parent="Uif-InquiryView">
 2                 <property name="headerText" value="Agenda Inquiry"/>
 3                 <property name="dataObjectClassName" value="org.kuali.rice.krms.impl.repository.AgendaBo"/>
 4                 <property name="viewHelperServiceClass"
value="org.kuali.rice.krms.impl.repository.AgendaInquiryHelperServiceImpl" />
 5                 <property name="additionalScriptFiles">
```

2. Assign the newly created permission to the appropriate roles

   Edit the role(s) being given permission to open the view. Add the permission to the role.

manual conversion

To make an inquiry available to all users in KRAD:

- Do Nothing!

script conversion

Full Ummask Field

The Full Unmask Field permission is used in both KNS and KRAD. The existing KNS permissions will work as is in KRAD. No conversion or duplication is necessary.

Partial Ummask Field

The Full Unmask Field permission is used in both KNS and KRAD. The existing KNS permissions will work as is in KRAD. No conversion or duplication is necessary.

Lookup Permissions

Maintenance Permissions

OJB to JPA Conversion

Conversion from OJB to JPA should be done a module at a time if possible.

In some cases there are clusters of tables that are related and will need to be converted together, but often there are sets of tables (even within a given module) which are independent from each other and can be converted independently to allow for some amount of testing.

JPA Configuration

Create a new spring file for your JPA configuration. Add a uniqueJpaPersistenceUnitName that is unique for the module(location in example). To include classes or whole packages add managedClassNames listed out or add the package prefix.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
                           http://www.springframework.org/schema/util
                           http://www.springframework.org/schema/util/spring-util-3.2.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.2.xsd">

 <alias
  alias="kradApplicationDataSource"
  name="locationDataSource" />
 <bean
  id="jpaPersistenceUnitName"
  class="java.lang.String">
  <constructor-arg value="location" />
 </bean>

 <util:list id="jpaPackagesToScan">
     <value>org.kuali.rice.location.impl</value>
 </util:list>

 <util:list id="managedClassNames" />

 <util:list id="additionalMetadataProviders" />
 <util:list id="springMetadataFileLocations" />

 <import resource="classpath:org/kuali/rice/krad/config/KRADSpringBeans-jpa-common.xml" />

</beans>
```

Add the highlighted below section to your ModuleConfigurationBean

```
 <bean id="locationModuleConfiguration" class="org.kuali.rice.krad.bo.ModuleConfiguration">
 <property name="namespaceCode" value="KR-NS"/>
   <property name="dataSourceName" value="locationDataSource"/>
   <property name="initializeDataDictionary" value="true"/>
   <property name="dataDictionaryService" ref="dataDictionaryService"/>
   <property name="persistenceService" ref="persistenceServiceOjb"/>
   <property name="dataDictionaryPackages">
     <list>
       <value>classpath:org/kuali/rice/location/web/campus/Campus.xml</value>
       <value>classpath:org/kuali/rice/location/web/campus/CampusType.xml</value>
       <value>classpath:org/kuali/rice/location/web/country/Country.xml</value>
       <value>classpath:org/kuali/rice/location/web/county/County.xml</value>
       <value>classpath:org/kuali/rice/location/web/postalcode/PostalCode.xml</value>
       <value>classpath:org/kuali/rice/location/web/state/State.xml</value>
       <value>classpath:org/kuali/rice/location/web/campus/CampusMaintenanceDocument.xml</value>
       <value>classpath:org/kuali/rice/location/web/campus/CampusTypeMaintenanceDocument.xml</value>
       <value>classpath:org/kuali/rice/location/web/country/CountryMaintenanceDocument.xml</value>
       <value>classpath:org/kuali/rice/location/web/county/CountyMaintenanceDocument.xml</value>
       <value>classpath:org/kuali/rice/location/web/postalcode/PostalCodeMaintenanceDocument.xml</value>
       <value>classpath:org/kuali/rice/location/web/state/StateMaintenanceDocument.xml</value>
     </list>
   </property>
   <property name="packagePrefixes">
     <list>
       <value>org.kuali.rice.location.</value>
     </list>
   </property>
   <property name="externalizableBusinessObjectImplementations">
     <map>
       <entry key="org.kuali.rice.location.framework.country.CountryEbo"
              value="org.kuali.rice.location.impl.country.CountryBo"/>
       <entry key="org.kuali.rice.location.framework.state.StateEbo"
              value="org.kuali.rice.location.impl.state.StateBo"/>
       <entry key="org.kuali.rice.location.framework.postalcode.PostalCodeEbo"
              value="org.kuali.rice.location.impl.postalcode.PostalCodeBo"/>
       <entry key="org.kuali.rice.location.framework.county.CountyEbo"
              value="org.kuali.rice.location.impl.county.CountyBo"/>
       <entry key="org.kuali.rice.location.framework.campus.CampusEbo"
              value="org.kuali.rice.location.impl.campus.CampusBo"/>
       <entry key="org.kuali.rice.location.framework.campus.CampusTypeEbo"
              value="org.kuali.rice.location.impl.campus.CampusTypeBo"/>
     </map>
   </property>

   <property name="entityManager" ref="sharedEntityManager" />
   <property name="providers">
     <list>
       <ref bean="jpaPersistenceProvider"/>
       <ref bean="metadataProvider"/>
     </list>
   </property>
   </bean>
```

Entity Conversion

To convert your entity from OJB to JPA you have two options. One is to use the script that will autoconvert for you or to manually convert based on the OJB-repository.xml file for the module. To start here is a sample OJB descripotor for Postal Code in Rice.

```
<class-descriptor class="org.kuali.rice.location.impl.postalcode.PostalCodeBo" table="KRLC_PSTL_CD_T">
 <field-descriptor name="countryCode" column="POSTAL_CNTRY_CD" jdbc-type="VARCHAR" primarykey="true"
 indexed="true" />
    <ield-descriptor name="code" column="POSTAL_CD" jdbc-type="VARCHAR" primarykey="true" indexed="true" />
    <field-descriptor name="stateCode" column="POSTAL_STATE_CD" jdbc-type="VARCHAR" />
    <field-descriptor name="cityName" column="POSTAL_CITY_NM" jdbc-type="VARCHAR" />
    <field-descriptor name="objectId" column="OBJ_ID" jdbc-type="VARCHAR" indexed="true" />
    <field-descriptor name="versionNumber" column="VER_NBR" jdbc-type="BIGINT" locking="true" />
```

```
 <field-descriptor name="active" column="ACTV_IND" jdbc-type="VARCHAR"
 conversion="org.kuali.rice.core.framework.persistence.ojb.conversion.OjbCharBooleanConversion"/>
 <field-descriptor name="countyCode" column="COUNTY_CD" jdbc-type="VARCHAR" />

 <reference-descriptor name="country" class-ref="org.kuali.rice.location.impl.country.CountryBo" auto-
retrieve="true" auto-update="none" auto-delete="none">
        <foreignkey field-ref="countryCode" target-field-ref="code" />
    </reference-descriptor>

    <reference-descriptor name="state" class-ref="org.kuali.rice.location.impl.state.StateBo" auto-
retrieve="true" auto-update="none" auto-delete="none">
        <foreignkey field-ref="countryCode" target-field-ref="countryCode" />
        <foreignkey field-ref="stateCode" target-field-ref="code"/>
   </reference-descriptor>

 <reference-descriptor name="county" class-ref="org.kuali.rice.location.impl.county.CountyBo" auto-
retrieve="true" auto-update="none" auto-delete="none">
        <foreignkey field-ref="countryCode" target-field-ref="countryCode" />
        <foreignkey field-ref="countyCode" target-field-ref="code" />
        <foreignkey field-ref="stateCode" target-field-ref="stateCode" />
    </reference-descriptor>
  </class-descriptor>
```

First to declare this class in JPA using @Entity and the table it maps to.

```
@Entity
@Table(name = "KRLC_PSTL_CD_T")
class PostalCodeBo extends PersistableBusinessObjectBase implements PostalCodeEbo {
```

Next is to declare each of the columns and what property it maps to.

```
@Entity
@Table(name = "KRLC_PSTL_CD_T")
class PostalCodeBo extends PersistableBusinessObjectBase implements PostalCodeEbo {

    @Column(name = "POSTAL_CD")
    String code;

    @Column(name = "POSTAL_CNTRY_CD")
    String countryCode;

    @Column(name = "POSTAL_CITY_NM")
    String cityName;

    @Column(name = "POSTAL_STATE_CD")
    String stateCode;

    @Column(name = "COUNTY_CD")
    String countyCode;

    @Column(name = "ACTV_IND")
    boolean active;
```

Next is to declare the ID columns. This class has a compound primary key so it uses an more advanced feature of JPA called an IdClass.

```
@IdClass(PostalCodeId.class)
@Entity
@Table(name = "KRLC_PSTL_CD_T")
class PostalCodeBo extends PersistableBusinessObjectBase implements PostalCodeEbo {

    @Id
    @Column(name = "POSTAL_CD")
    String code;

    @Id
    @Column(name = "POSTAL_CNTRY_CD")
```

```
    String countryCode;

    @Column(name = "POSTAL_CITY_NM")
    String cityName;

    @Column(name = "POSTAL_STATE_CD")
    String stateCode;

    @Column(name = "COUNTY_CD")
    String countyCode;

    @Column(name = "ACTV_IND")
    boolean active;
```

The active indicator used an OJB converter to handle the Y/N conversion into boolean. This is done with a customized EclipseLink Converter now.

```
@IdClass(PostalCodeId.class)
@Entity
@Table(name = "KRLC_PSTL_CD_T")
class PostalCodeBo extends PersistableBusinessObjectBase implements PostalCodeEbo {
   ...

    @Column(name = "ACTV_IND")
    @javax.persistence.Convert(converter=BooleanYNConverter.class)
    boolean active;


    @Converter(
  autoApply = true)
public class BooleanYNConverter implements AttributeConverter<Boolean, String> {

 protected static final Set<String> YES_VALUES = new HashSet<String>();
 static {
  YES_VALUES.add("Y");
  YES_VALUES.add("y");
  YES_VALUES.add("true");
  YES_VALUES.add("TRUE");
 }

 @Override
 public String convertToDatabaseColumn(Boolean objectValue) {
  if (objectValue == null) {
   return "N";
  }
  return objectValue ? "Y" : "N";
 }

 @Override
 public Boolean convertToEntityAttribute(String dataValue) {
  if (dataValue == null) {
   return false;
  }
  return YES_VALUES.contains(dataValue);
 }
}
```

The next part is to map the relationships that are defined by the reference-descriptor in OJB. The CountyBo has 3 columns as its primary key so it is described as follows.

```
@IdClass(PostalCodeId.class)
@Entity
@Table(name = "KRLC_PSTL_CD_T")
class PostalCodeBo extends PersistableBusinessObjectBase implements PostalCodeEbo {
    ...
    @ManyToOne(targetEntity = CountyBo.class, fetch = FetchType.EAGER)
    @JoinColumns(value=[@JoinColumn(name = "COUNTY_CD", referencedColumnName="COUNTY_CD", insertable = false,
 updatable = false),
```

```
                        @JoinColumn(name="POSTAL_STATE_CD", referencedColumnName="STATE_CD", insertable =
 false, updatable = false),
                        @JoinColumn(name="POSTAL_CNTRY_CD", referencedColumnName="POSTAL_CNTRY_CD", insertable
 = false, updatable = false)])
    CountyBo county;
```

Migrate Service methods and Data Access layer to JPA

During the JPA conversion it is a good opportunity to remove any code that is not necessary due to enhancements
in the core data fetching layer in the KRAD Data Module. The following are the best practices that should be used
when using JPA in Kuali applications. In general the order of consideration for a query should go as the following:

1. Use DataObjectService methods in service methods.
2. Create custom DAOs method to use NamedQuery in JPA.
3. Use Rice Criteria API if query is too complicated and would require dynamic generation(String concatenation).
4. Use JPA Criteria API if query requires functions not supported in Rice Criteria API.

Simple DataObjectService fetch by Primary Key

```
CountryBo countryBo = getDataObjectService().find(CountryBo.class,code);

//Fetch by Compound Primary Key
final Map<String, Object> map = new HashMap<String, Object>();
map.put("countryCode", countryCode);
map.put("code", code);
StateBo stateBo = getDataObjectService().find(StateBo.class,new CompoundKey(map)));
```

DataObjectService query for matching results

```
//Fetch all matching results by countryCode and that have active equivalent to true
final Map<String, Object> map = new HashMap<String, Object>();
map.put("countryCode", countryCode);
map.put("active", Boolean.TRUE);

QueryResults<PostalCodeBo> postalCodeBoQueryResults = getDataObjectService().
    findMatching(PostalCodeBo.class,QueryByCriteria.Builder.andAttributes(map).build());

//Fetch all Countries that have alternateCountryCode equal to value passed in
QueryByCriteria qbc = QueryByCriteria.Builder.forAttribute(KRADPropertyConstants.ALTERNATE_POSTAL_COUNTRY_CODE,
                alternateCode).build();
QueryResults<CountryBo> countryBoQueryResults = getDataObjectService().findMatching(CountryBo.class,qbc);
List<CountryBo> countryList = countryBoQueryResults.getResults();
```

DataObjectService query returning the count based on Criteria

```
//Fetch count based on document id and principal id and current indicator being true
QueryByCriteria.Builder criteria = QueryByCriteria.Builder.create().setPredicates(
                                       equal(DOCUMENT_ID, documentId),
                                   equal(PRINCIPAL_ID, principalId),
                                   equal(CURRENT_INDICATOR, Boolean.TRUE)
                               );
criteria.setCountFlag(CountFlag.ONLY);
return getDataObjectService().findMatching(ActionTakenValue.class, criteria.build()).getTotalRowCount();
```

Injecting the Shared Entity Manager

```
//Add the following to your Spring DAO implementation to assign the appropriate Persistence
//Unit to your DAO

public class DocumentTypeDAOJpa implements DocumentTypeDAO {

@PersistenceContext(unitName="kew")
private EntityManager entityManager;

}
```

Simple example of Named Query in Rice

```
//Fetch Application Document ID by Document ID
//Define constants for named query in DAO - In this case DocumentRouteHeaderDAOJpa
//Name your queries such that they start with the Entity name
//like @NamedQuery(name="ParameterBo.findAll", query="SELECT p FROM ParameterBo")

public static final String GET_APP_DOC_STATUS_NAME = "DocumentRouteHeaderValue.GetAppDocStatus";
public static final String GET_APP_DOC_STATUS_QUERY = "SELECT d.appDocStatus from "
            + "DocumentRouteHeaderValue as d where d.documentId = :documentId";

//Definition of NamedQuery on Queried Entity(DocumentRouteHeaderValue)
    @NamedQuery(name=DocumentRouteHeaderDAOJpa.GET_APP_DOC_STATUS_NAME, query=
            DocumentRouteHeaderDAOJpa.GET_APP_DOC_STATUS_QUERY)

//Code to call NamedQuery
TypedQuery<String> query = getEntityManager().createNamedQuery(
                            "DocumentRouteHeaderValue.GetAppDocId",String.class);
query.setParameter("documentId",documentId);

String applicationDocId = null;
if(query.getResultList() != null && !query.getResultList().isEmpty()){
    applicationDocId = query.getResultList().get(0);
}
return applicationDocId;
```

More Complex example of NamedQuery

```
//Fetch all distinct document IDs by document type and application document ID
public static final String GET_DOCUMENT_ID_BY_DOC_TYPE_APP_ID_NAME =
            "DocumentRouteHeaderValue.GetDocumentIdByDocTypeAndAppId";
public static final String GET_DOCUMENT_ID_BY_DOC_TYPE_APP_ID_QUERY = "SELECT "
            + "DISTINCT(DH.documentId) FROM DocumentRouteHeaderValue DH, DocumentType DT "
            + "WHERE DH.appDocId = :appDocId AND DH.documentTypeId = DT.documentTypeId  AND DT.name = :name";

@NamedQuery(name=DocumentRouteHeaderDAOJpa.GET_DOCUMENT_ID_BY_DOC_TYPE_APP_ID_NAME, query =
            DocumentRouteHeaderDAOJpa.GET_DOCUMENT_ID_BY_DOC_TYPE_APP_ID_QUERY)

TypedQuery<String> query = getEntityManager().createNamedQuery(GET_DOCUMENT_ID_BY_DOC_TYPE_APP_ID_NAME,
                String.class);
query.setParameter("appDocId",appId);
query.setParameter("name",documentTypeName);
return query.getResultList();
```

Tip

When building named queries you must use an alias the object "select **r** from KUL_RICE_T **r**". If you get the below error you are probably missing an alias.

```
Caused by: java.lang.ClassCastException:
                    org.eclipse.persistence.jpa.jpql.parser.NullExpression
                    cannot be cast to org.eclipse.persistence.jpa.jpql.parser.IdentificationVariable
```

Using Rice Criteria API

```
//Example of Dynamic query, this query needs to add date checks dates
//if effectiveDate parameter is not null

//This should be in a DAO class - RuleDAOJpa in this case

public List<RuleBaseValues> fetchAllCurrentRulesForTemplateDocCombination(String ruleTemplateId, List
 documentTypes, Timestamp effectiveDate) {
       QueryByCriteria.Builder builder = QueryByCriteria.Builder.create();
        List<Predicate> predicates = new ArrayList<Predicate>();
        predicates.add(equal("ruleTemplateId",ruleTemplateId));
        predicates.add(in("docTypeName", documentTypes));
        predicates.add(equal("active", Boolean.TRUE));
        predicates.add(equal("delegateRule",Boolean.FALSE));
        predicates.add(equal("templateRuleInd",Boolean.FALSE));

        if(effectiveDate != null){
            predicates.add(lessThanOrEqual("activationDate",effectiveDate));
            predicates.add(greaterThanOrEqual("deactivationDate", effectiveDate));
        }
        List<Predicate> datePredicateList = generateFromToDatePredicate(new Date());
        Predicate[] datePreds = generateFromToDatePredicate(new Date()).
                toArray(new Predicate[datePredicateList.size()]);
        predicates.add(and(datePreds));
        Predicate[] preds = predicates.toArray(new Predicate[predicates.size()]);
        builder.setPredicates(preds);
        QueryResults<RuleBaseValues> results = getDataObjectService().findMatching(RuleBaseValues.class,
                                            builder.build());
        return results.getResults();
 }

    public List<Predicate> generateFromToDatePredicate(Date date){
        List<Predicate> datePredicates = new ArrayList<Predicate>();

        Predicate orFromDateValue = or(lessThanOrEqual("fromDateValue",new Timestamp(date.getTime())),
              isNull("fromDateValue"));
        Predicate orToDateValue = or(greaterThanOrEqual("toDateValue",new Timestamp(date.getTime())),
              isNull("toDateValue"));

        datePredicates.add(orFromDateValue);
        datePredicates.add(orToDateValue);

        return datePredicates;
    }
```

Using JPA Criteria API

```
//Using JPA Criteria Builder

 public List<RuleBaseValues> search(String docTypeName, String ruleId, String ruleTemplateId, String
 ruleDescription,
 String groupId, String principalId, Boolean delegateRule, Boolean activeInd, Map extensionValues, String
 workflowIdDirective) {
        CriteriaBuilder cb = getEntityManager().getCriteriaBuilder();
        CriteriaQuery<RuleBaseValues> cq = cb.createQuery(RuleBaseValues.class);
        Root<RuleBaseValues> root = cq.from(RuleBaseValues.class);
        List<javax.persistence.criteria.Predicate> predicates = getSearchCriteria(root,cq,docTypeName,
 ruleTemplateId, ruleDescription, delegateRule, activeInd, extensionValues);

        if (ruleId != null) {
            predicates.add(cb.equal(root.get("id"),ruleId));
        }
        if (groupId != null) {
            predicates.add(cb.in(root.get("id")).value(getRuleResponsibilitySubQuery(
                    groupId, cq)));
        }
        Collection<String> kimGroupIds = new HashSet<String>();
        Boolean searchUser = Boolean.FALSE;
        Boolean searchUserInWorkgroups = Boolean.FALSE;

        if ("group".equals(workflowIdDirective)) {
```

```
            searchUserInWorkgroups = Boolean.TRUE;
        } else if (StringUtils.isBlank(workflowIdDirective)) {
            searchUser = Boolean.TRUE;
            searchUserInWorkgroups = Boolean.TRUE;
        } else {
            searchUser = Boolean.TRUE;
        }

        if (!org.apache.commons.lang.StringUtils.isEmpty(principalId) && searchUserInWorkgroups) {
            Principal principal = null;

            principal = KimApiServiceLocator.getIdentityService().getPrincipal(principalId);

            if (principal == null)
            {
             throw new RiceRuntimeException("Failed to locate user for the given principal id: " +
principalId);
            }
            kimGroupIds = KimApiServiceLocator.getGroupService().getGroupIdsByPrincipalId(principalId);
        }
        Subquery<RuleResponsibilityBo> subquery = addResponsibilityCriteria(cq,kimGroupIds, principalId,
searchUser, searchUserInWorkgroups);

        if(subquery != null){
            predicates.add(cb.in(root.get("id")).value(subquery));
        }
        cq.distinct(true);
        javax.persistence.criteria.Predicate[] preds = predicates.toArray(
                new javax.persistence.criteria.Predicate[predicates.size()]);
        cq.where(preds);
        TypedQuery<RuleBaseValues> q = getEntityManager().createQuery(cq);

        return q.getResultList();
}

   private Subquery<RuleResponsibilityBo> getRuleResponsibilitySubQuery(String ruleRespName,
                       CriteriaQuery<RuleBaseValues> query){
        CriteriaBuilder cb = getEntityManager().getCriteriaBuilder();
        Subquery<RuleResponsibilityBo> subquery = query.subquery(RuleResponsibilityBo.class);
        Root fromResp = subquery.from(RuleResponsibilityBo.class);
        subquery.where(cb.equal(fromResp.get("ruleResponsibilityName"),ruleRespName));
        subquery.select(fromResp.get("ruleBaseValuesId"));

        return subquery;
   }
```

Miscellaneous JPA information

Tip

```
//Relationship foreign key updating can go wrong if missing
//"nullable" on JoinColumn.  It can insert null into the column instead of
//the actual value of the foreign entity key

public class RouteNodeInstance implements Serializable {
    @ManyToOne
    @JoinColumn(name="RTE_NODE_ID", nullable = false)
    private RouteNode routeNode;
}
```