Kuali Rice 2.4.0-M2-SNAPSHOT KNS to KRAD Conversion Guide

Released: 10-15-2013

Table of Contents

Introduction

This technical documentation assists with the conversion of existing Kuali Rice application to utilize the new KRAD framework. It does so by identifying the changes between the KNS and KRAD implementations of Kuali Rice application features. Overview and detailed documentation of features of the KRAD frame work can be found in the KRAD Guide and are omitted in this document.

The KNS2KRAD Conversion Script is a tool provided by Kuali Rice to automatize as much as reasonably possible. Throughout this document the following items are used to indicate if the utility supports the conversion:

script conversion Feature can be converted with the conversion script.

manual conversion Feature needs to be converted manually.

Data Dictionary

The Data Dictionary is a repository of metadata primarily describing data objects and their properties. Attribute Definitions provide the metadata about the attributes (i.e. fields) of a data object. The KNS2KRAD Conversion Script will automate part of the conversion for your data dictionary files. Some data dictionary attributes require no conversion, they are virtually identical between KNS and KRAD. These items will be passed through by the conversion script. Some items will be converted from KNS to KRAD by the script. Others will require manual conversion.

The subsections below 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:

```
<bean id="Book" parent="Book-parentBean"/>
<bean id="Book-parentBean" abstract="true" parent="DataObjectEntry">
  <property name="dataObjectClass" value="org.kuali.rice.knsapp.Book"/>
  <property name="titleAttribute" value="id"/>
  <property name="objectLabel" value="Book"/>
  <property name="attributes">
    <list>
      <ref bean="Book-id"/>
      <ref bean="Book-title"/>
      <ref bean="Book-price"/>
    </list>
  </property>
</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 and 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. 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:

```
<bean id="Country-code" parent="Country-code-parentBean"/>
<bean id="Country-code-parentBean" abstract="true" parent="AttributeDefinition">
  <property name="name" value="code"/>
  <property name="forceUppercase" value="true"/>
  <property name="label" value="Country Code"/>
  <property name="shortLabel" value="Country Code"/>
  <property name="maxLength" value="2"/>
  <property name="required" value="true"/>
  <property name="summary" value="Postal Country Code"/>
  <property name="description" value="The code uniquely identify a country."/>
  <property name="validationPattern">
    <bean parent="AlphaNumericValidationPattern"/>
  </property>
  <property name="control">
    <bean parent="TextControlDefinition" p:size="2"/>
  </property>
</bean>
```

KRAD Attribute Definition code example:

```
<bean id="Country-code" parent="Country-code-parentBean"/>
<bean id="Country-code-parentBean" abstract="true" parent="AttributeDefinition">
  <property name="name" value="code"/>
  <property name="forceUppercase" value="true"/>
  <property name="label" value="Country Code"/>
  <property name="shortLabel" value="Country Code"/>
  <property name="maxLength" value="2"/>
  <property name="required" value="true"/>
  <property name="summary" value="Postal Country Code"/>
  <property name="description" value="The code uniquely identify a country."/>
  <property name="validCharactersConstraint">
    <bean parent="AlphaNumericPatternConstraint"/>
  </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. The naming convention has changed for constraints and patterns, but the functionality is the same. KRAD also has several other validations available. See the KRAD Guide for more detail. The example below demonstrate the differences in defining validation constraints using a single validation pattern. Several such patterns are available

KRAD Attribute Definition code example:

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

KRAD Attribute Definition code 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, including:

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

See the KRAD User Guide for more detail.

Additional Constraint Patterns

KRAD has additional constraint patterns too

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

See the KRAD User 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>
4
```

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

KNS Button control example:

```
1 <property name="control">
2   <bean parent="ButtonControlDefinition"/>
3 </property>
4
```

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 User Guide for more detail

Checkbox Control

KNS Checkbox control example:

```
1 <property name="control">
```

```
2   <bean parent="CheckboxControlDefinition"/>
3 </property>
4
```

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 User Guide for more detail.

Currency Control

KNS Currency control example:

```
1 <property name="control">
2   <bean parent="CurrencyControlDefinition" p:formattedMaxLength="26" p:size="10"/>
3 </property>
4
```

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

KNS File control example:

```
1 <property name="control">
2   <bean parent="FileControlDefinition"/>
3 </property>
4
```

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

KNS Hidden control example:

```
1 <property name="control">
2   <bean parent="HiddenControlDefinition"/>
3 </property>
4
```

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

KNS Kuali User control example:

```
1 <property name="control">
2   <bean parent="KualiUserControlDefinition"/>
3 </property>
4
```

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

KNS Link control example:

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

KRAD Link control example:

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

```
3           </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 **`jrefText`** 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

KNS TextArea control example:

```
1 <property name="control">
2   <bean parent="TextareaControlDefinition"/>
3 </property>
4
```

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

KNS Text control example:

```
1 <property name="control">
2   <bean parent="TextControlDefinition"/>
3 </property>
4
```

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. Including:

- **`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 User Guide for more detail.

Radio Control Group

KNS Radio Group control example:

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

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"
p:labelAttribute="para"/>
14   </property>
15 </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-DropdownContro, by setting property **`multiple`** to true. script conversion

Date Control

KNS Date control example:

```
1 <property name="control">
2   <bean parent="TextareaControlDefinition"/>
3 </property>
4
```

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

InquiryDefinition in KNS Data Dictionary to InquiryView in KRAD

KNS Code example:

```
<bean id="EntityType-inquiryDefinition" parent="InquiryDefinition">
  <property name="title" value="Entity Type Inquiry"/>
  <property name="inquirableClass" value="org.kuali.rice.kim.inquiry.EntityTypeInquirableImpl"/>
  <property name="inquirySections">
    <list>
      <bean parent="InquirySectionDefinition">
        <property name="title" value="Entity Type"/>
        <property name="defaultOpen" value="true"/>
        <property name="numberOfColumns" value="1"/>
        <property name="inquiryFields">
          <list>
            <bean parent="FieldDefinition" p:attributeName="code" p:noInquiry="true"/>
            <bean parent="FieldDefinition" p:attributeName="name"/>
            <bean parent="FieldDefinition" p:attributeName="active"/>
          </list>
        </property>
      </bean>
      <bean parent="InquirySectionDefinition">
        <property name="title" value="Rule Attributes"/>
        <property name="defaultOpen" value="false"/>
        <property name="inquiryFields">
          <list>
            <bean parent="InquiryCollectionDefinition">
              <property name="numberOfColumns" value="1"/>
              <property name="businessObjectClass"
value="org.kuali.rice.kim.impl.identity.EntityTypeDetailsBo"/>
              <property name="attributeName" value="entityTypeDetails"/>
                <property name="inquiryFields">
                  <list>
                    <bean parent="FieldDefinition" p:attributeName="name"/>
                    <bean parent="FieldDefinition" p:attributeName="value"/>
                  </list>
                </property>
              <property name="summaryTitle" value="Entity Type Details"/>
              <property name="summaryFields">
                <list>
                  <bean parent="FieldDefinition" p:attributeName="name"/>
                </list>
              </property>
            </bean>
          </list>
        </property>
      </bean>
    </list>
  </property>
</bean>
```

KRAD code example:

```
<bean id="EntityType-InquiryView" parent="Uif-InquiryView">
  <property name="headerText" value="Entity Type"/>
  <property name="dataObjectClassName" value="org.kuali.rice.kim.impl.identity.EntityTypeBo"/>
  <property name="viewHelperServiceClass" value="org.kuali.rice.kim.inquiry.EntityTypeInquirableImpl"/>
```

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

Sections

- The property that contains the inquiry sections was renamed from **`inquirySections`** to **`items`**. [KNS line: 4, KRAD line: 5] 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: 6, KRAD line: 7] script conversion

- The title of the section is specified via the **`headerText`** property instead of the **`title`** property. [KNS line: 7, KRAD line: 8] 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: 8, KRAD line: 9] 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: 9, KRAD line: 10] script conversion
- The org.kuali.rice.kns.inquiry.Inquirable.addAdditionalSections() has been removed in KRAD. To add additional sections dynamically to an inquiry override **`performFinalize`** 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`**.

```
 1 <bean id="Sample-InquiryView" parent="Uif-InquiryView">
 2                 ...
 3                 <property name="viewHelperServiceClass"
value="org.kuali.rice.krad.inquiry.CustomInquirableImpl/>
 4                 ...
 5                 </bean>
 6
```

- The org.kuali.rice.kns.inquiry.Inquirable.getSection() has been removed in KRAD. To modify existing sections dynamically in an inquiry override **`performFinalize`** method of ViewHelperService. The list of components can be obtained from the view object using **`getComponentsForLifecycle`**.
- The property that contains the inquiry display fields was renamed from **`inquiryFields`** to **`items`**. [KNS line: 10, KRAD line: 11] script conversion
- Data fields changed from **`FieldDefinition`** to **`Uif-DataField`**. [KNS line: 12-14, KRAD line: 13-15] script conversion
- The **`attributeName`** property on the field is now a **`propertyName`** property. [KNS line: 12-14, KRAD line: 13-15] 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: 12, KRAD line: 13] 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: 24, KRAD line: 19] script conversion

- The **businessObjectClass** property of the collection has been changed to **collectionObjectClass**. [KNS line: 25, KRAD line: 23] script conversion
- The **attributeName** property of the collection has been changed to **propertyName**. [KNS line: 26, KRAD line: 24] script conversion
- The property that contains the inquiry display fields was renamed from **inquiryFields** to **items**. [KNS line: 27, KRAD line: 25] script conversion
- The property **summaryTitle** has been moved to **layoutManager.summaryTitle**. [KNS line: 33, KRAD line: 31] script conversion
- The container **summaryFields** has been moved to **layoutManager.summaryFields** and simplified to just take a list of values. [KNS line: 34, KRAD line: 32] 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.buildInquiryLink** and be adapted for the new framework. Implementers have a choice between using the convenience method **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

KualiInquiryAction

KRAD removes **`KualiInquiryAction`** and replaces it with **`InquiryController`**. Normally, this class does not need to be extended unless advanced customizations are required, but any implementer already extending this class will have to make the switch manually. manual conversion

- Overrides of the methods **`downloadAttachment`**, **`downloadCustomBOAttachment`**, and **`downloadBOAttachment`** need to be moved to **`InquiryController`** and adapted to the new Spring MVC format. 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

LookupDefinition in KNS Data Dictionary to LookupView in KRAD

Note that 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:

```
 1 <bean id="EntityType-lookupDefinition" parent="LookupDefinition">
 2   <property name="title" value="Entity Type Lookup"/>
 3   <property name="menubar" value="&lt;a href=&quot;javascript:void(0)&quot;
 4               onclick=&quot;alert('JavaScript triggered action.')&quot;&gt;Custom Button&lt;/a&gt;"/>
 5   <property name="numOfColumns" value="2"/>
 6   <property name="extraButtonSource" value="images/tinybutton-createnew.gif"/>
 7   <property name="extraButtonParams" value="createNew"/>
 8   <property name="disableSearchButtons" value="true"/>
 9   <property name="lookupFields">
10     <list>
11       <bean parent="FieldDefinition" p:attributeName="code"/>
12       <bean parent="FieldDefinition" p:attributeName="name" p:noLookup="true"
13                                        p:treatWildcardsAndOperatorsAsLiteral="true"/>
14       <bean parent="FieldDefinition" p:attributeName="active" p:defaultValue="Y"/>
15     </list>
16   </property>
17   <property name="resultFields">
18     <list>
19       <bean parent="FieldDefinition" p:attributeName="code" p:triggerOnChange="true"/>
20       <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"/>
      <bean parent="Uif-DataField" p:propertyName="sortCode" p:enableAutoInquiry="false"/>
      <bean parent="Uif-DataField" p:propertyName="amount"/>
      <bean parent="Uif-DataField" p:propertyName="active"/>
    </list>
  </property>
  <property name="defaultSortAscending" value="false"/>
  <property name="defaultSortAttributeNames">
    <list>
      <value>code</value>
    </list>
  </property>
  <property name="resultsGroup.layoutManager.columnCalculations">
    <list>
      <bean parent="Uif-ColumnCalculationInfo-Sum" p:propertyName="amount" />
    </list>
  </property>
  <property name="translateCodesOnReadOnlyDisplay" value="true"/>
  <property name="multipleValuesSelectResultSetLimit" value="100"/>
  <property name="resultSetLimit" value="200"/>
</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_PER_PAGE application/system parameter. manual conversion
- **`resultSetLimit`** [KRAD line 52] overrides the application/system parameter RESULTS_LIMIT. 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 could 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">
2   <property name="dataObjectClassName" value="org.kuali.rice.SampleBo" />
3   ...
4 </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":

```
1 <bean id="Sample-LookupView" parent="Uif-LookupView">
2   ...
3   <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:

```
<bean id="Sample-LookupView" parent="Uif-LookupView">
  ...
  <property name="resultsReturnField">
    <bean parent="Uif-LinkField" p:href="http://www.kuali.org" p:linkText="Kuali"/>
  </property>
  ...
</bean>
```

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

```
<bean id="Sample-LookupView" parent="Uif-LookupView">
  ...
  <property name="page.header.lowerGroup.items">
    <list>
      <bean parent="Uif-Link" p:linkText="Create New" p:href="http://www.kuali.org">
        <property name="cssClasses">
          <list merge="true">
            <value>uif-createNewLink</value>
          </list>
        </property>
      </bean>
    </list>
  </property>
  ...
</bean>
```

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

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

```
11
```

- **`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).

```
 1 <bean parent="Uif-InputField">
 2   ...
 3   <property name="quickfinder.fieldConversions">
 4     <map>
 5       <entry key="cd" value="sampleCd" />
 6       <entry key="description" value="sample.description" />
 7     </map>
 8   </property>
 9   ...
10 </bean>
```

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

```
 1 <bean id="Sample-LookupView" parent="Uif-LookupView">
 2   ...
 3   <property name="criteriaFields">
 4     <list>
 5       <bean parent="Uif-LookupCriteriaInputField" p:propertyName="namespace"
 6             p:readOnly="true" />
 7       <bean parent="Uif-LookupCriteriaInputField" p:propertyName="cd" />
 8     </list>
 9   </property>
10   ...
11 </bean>
```

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

```
1 <bean id="Sample-LookupView" parent="Uif-LookupView">
2   <property name="viewHelperServiceClass"
3             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">
```

```
  <property name="onLoadScript" value="alert('Hi!')" />
  ...
</bean>
```

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

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

```
<bean parent="MaintainableFieldDefinition" p:name="entityTypeCode"/>
  <property name="overrideLookupClass" value="org.kuali.rice.kim.impl.identity.EntityTypeBo"/>
  <property name="overrideFieldConversions">
    <map>
      <entry key="code" value="entityTypeCode"/>
    </map>
  </property>
</bean>
```

  KRAD code example:

```
<bean parent="Uif-InputField" p:propertyName="entityTypeCode">
  <property name="quickfinder.dataObjectClassName" value="org.kuali.rice.kim.impl.identity.EntityTypeBo"/>
  <property name="quickfinder.fieldConversions">
    <map>
```

```
5       <entry key="code" value="entityTypeCode"/>
6     </map>
7   </property>
8 </bean>
```

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]

Other Lookup Features

- 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="org.kuali.rice.krad.demo.travel.authorization.TravelAuthorizationForm"/>
    ...
</bean>
```

Maintenance Documents

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

Specifying a maintainable class

- 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.**

```
 1
 2                <property name="maintainableClass"
value="org.kuali.rice.krad.maintenance.SampleMaintainableImpl" />
 3
```

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.

```
1
2            public void prepareBusinessObject(BusinessObject businessObject);
3
```

In KRAD this has been replaced by using MaintenanceDocumentService.setupMaintenanceObject

```
1
2            public void setupMaintenanceObject(MaintenanceDocument document, String maintenanceAction,
3            Map<String, String[]> requestParameters);
4
```

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.

- 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.

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 the 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.

Transactional Documents

Conversion Tools

Introduction

Lookup

Inquiry

Maintenance

Transactional

KNS to KRAD Conversion Script

KNS to KRAD conversion is setup into four steps:

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

The existing script for KRAD conversion is included in KRADConversion.groovy. The script will process lookup, inquiry, and MaintenanceDocumentEntries into KRAD-compliant XML files.

OJB to JPA Conversion

coming soon...