Kuali Rice 2.4.0-M5 eDocLite Guide

Table of Contents

List of Figures

List of Tables

Chapter 1. Overview

eDocLite is a simple, form-based framework designed for rapid development and implementation within an existing Kuali Enterprise Workflow (KEW) infrastructure. It allows for the development of simple web applications, their forms and simple routing configurations using XML. Users only have to enter data into the form and then submit it. Rules can be constructed so that the form is then routed to a specific user or KIM Group based on the data entered. It can be integrated with larger applications using a database layer post-processor component.

Web pages called eDoc's are generated and are associated with a specific document type definition that provides the overall definition for how the document can be routed. Document types can also exist in hierarchies which provide storage of common information at various levels.

Key Ideas:

- Rapid implementation and development solution for simpler documents

- Easily re-configured

- Easily manageable

- Entirely web-based from design/development and user perspectives

- No java code required for developments; only XML with optional JavaScript for client side editing (workflow handles execution)

- Some validation JavaScript is automatically generated like regular expression editing and 'required field checking'.

The form uses an XSLT stylesheet to generate the html code. It uses XML to define form fields. Certain workflow classes make helper data available to the stylesheet programmer, and there are several features that can be 'plugged-in' to eDocLite to further enhance its usability in many situations. The actual form display is called an EDL. This diagram shows how these objects are related:

Figure 1.1. EDL Controller Chain
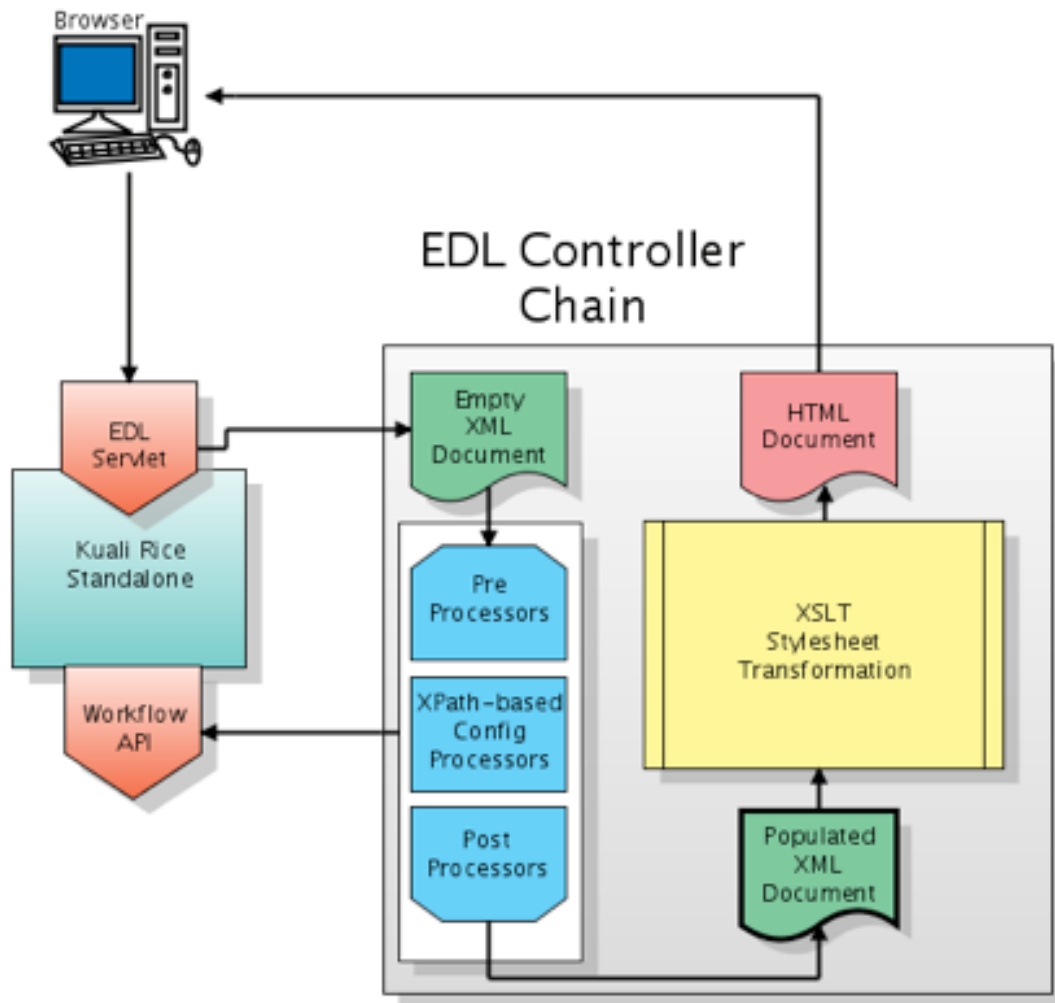
Chapter 2. Components

Field Definitions

You need to define eDocLite fields to capture data that is passed to the server for storage.

Key Information about eDocLite fields:

- Save eDocLite data fields as key value pairs in two columns of a single database table.

- Use the xml element name as the key.

- You do not need to make any database-related changes when building eDocLite web applications.

- Store documents by document number.

- Make all field names unique within a document type.

The code example below focuses on the EDL section of the eDocLite form definition. The file Edoclite.xsd found in source under the impl/src/main/resources/schema/ directory describes the xml rules for this section.

Note that the first few lines proceeding **<edl name="eDoc.Example1.Form"** relate to namespace definitions. These are common across all eDocLites, so this guide does not discuss them.

In this example, any XML markup that has no value shown or that is not explained offers options that are not important at this time.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="ns:workflow resource:WorkflowData">
 3     <edoclite xmlns="ns:workflow/eDocLite " xsi:schemaLocation="ns:workflow/eDocLite  resource:eDocLite ">
 4
 5     <edl name="eDoc.Example1.Form" title="Example 1">
 6         <security />
 7         <createInstructions>** Questions with an asterisk are required.</createInstructions>
 8         <instructions>** Questions with an asterisk are required.</instructions>
 9         <validations />
10         <attributes />
11         <fieldDef name="userName" title="Full Name">
12             <display>
13                 <type>text</type>
14                 <meta>
15                     <name>size</name>
16                     <value>40</value>
17                 </meta>
18             </display>
19             <validation required="true">
20                 <message>Please enter your full name</message>
21             </validation>
22         </fieldDef>
23         <fieldDef name="rqstDate" title="Requested Date of Implementation:">
24             <display>
25                 <type>text</type>
26             </display>
27             <validation required="true">
28                 <regex>^[0-1]?[0-9](/|-)[0-3]?[0-9](/|-)[1-2][0-9][0-9][0-9]$</regex>
29                 <message>Enter a valid date in the format mm/dd/yyyy.</message>
30             </validation>
31         </fieldDef>
32         <fieldDef name="requestType" title="Request Type:">
33             <display>
34                 <type>radio</type>
35                 <values title="New">New</values>
36                 <values title="Modification">Modification</values>
37             </display>
```

```
38              <validation required="true">
39                  <message>Please select a request type.</message>
40              </validation>
41          </fieldDef>
42          <fieldDef attributeName="EDL.Campus.Example" name="campus" title="Campus:">
43              <display>
44                  <type>select</type>
45                  <values title="IUB">IUB</values>
46                  <values title="IUPUI">IUPUI</values>
47              </display>
48              <validation required="true">
49                  <message>Please select a campus.</message>
50              </validation>
51          </fieldDef>
52          <fieldDef name="description" title="Description of Request:">
53              <display>
54                  <type>textarea</type>
55                  <meta>
56                      <name>rows</name>
57                      <value>5</value>
58                  </meta>
59                  <meta>
60                      <name>cols</name>
61                      <value>60</value>
62                  </meta>
63                  <meta>
64                      <name>wrap</name
65                      <value>hard</value>
66                  </meta>
67              </display>
68              <validation required="false" />
69          </fieldDef>
70          <fieldDef name="fundedBy" title="My research/sponsored program work is funded by NIH or NSF.">
71              <display>
72                  <type>checkbox</type>
73                  <values title="My research/sponsored program work is funded by NIH or NSF.">nihnsf</values
74              </display>
75          </fieldDef>
76          <fieldDef name="researchHumans" title="My research/sponsored program work involves human
subjects.">
77              <display>
78                  <type>checkbox</type>
79                  <values title="My research/sponsored program work involves human subjects.">humans</values>
80              </display>
81          </fieldDef>
82      </edl>
83      </eDocLite>
84 </data>
```

In the EDL XML file, field definition is embodied in the **edl** element. This element has a **name** attribute that is used to identify this file as a definition of an EDL form. It often has a **title** for display purposes.

Examination of this code shows that

- Individual fields have names, titles, and types. The types closely match html types.

- You can easily use simple validation attributes and sub-attributes to ensure that a field is entered if required and that an appropriate error message is presented if no value is provided by the web user.

- Regular expressions enhance the edit criteria without using custom JavaScript. (There are several ways that you can invoke custom JavaScript for a field, but they are not shown in this example.)

- An important field named campus has syntax that defines the value used to drive the routing destination. (In more complex documents, several fields are involved in making the routing decision.)

XSLT Style Sheet

The next section of the EDL XML file is the XSLT style sheet. It renders the EDL that the browser will present and contains logic to determine how data is rendered to the user.

A major workhorse of the XSLT code is contained in a style sheet library called **widgets.xml**. In the example below, it's included in the style sheet using an **xsl:include** directive.

Workflow Java classes have API's that offer methods that supply valuable information to the XSLT style sheet logic. XML allows you to interrogate the current value of **EDL**-defined fields, and it provides a variety of built-in functions.

Together, these helpers allow the eDocLite style sheet programmer to focus on rendering fields and titles using library (widget) calls and to perform necessary logic using the constructs built into the XML language(**if**, **choose…when**, etc.).

This is the area of eDocLite development that takes the longest and is the most tedious. Much of what the eDocLite style sheet programmer writes focuses on are: which fields and titles appear, in what order, to which users, and whether the fields are **readOnly**, **editable**, or **hidden**.

Below is the style sheet section of the EDL XML form for our example. It contains embedded comments.

```
 1 <!-- widgets is simply more xslt that contains common functionality that greatly simplifies html rendering
 2 It is somewhat complicated but does not require changes or full understanding unless enhancements are
required.  -->
 3 <xsl:include href="widgets" />
 4 <xsl:output indent="yes" method="html" omit-xml-declaration="yes" version="4.01" />
 5
 6 <!-- variables in the current version of xslt cannot be changed once set. Below they are set to various
values often fed by java classes or to
 7 values contained in workflow xml. Not all of these are used in this form but are shown because often they
can be useful
 8 The ones prefixed with my-class are methods that are exposed by workflow to eDocLite .-->
 9 <xsl:variable name="actionable" select="/documentContent/documentState/actionable" />
10 <xsl:variable name="docHeaderId" select="/documentContent/documentState/docId" />
11 <xsl:variable name="editable" select="/documentContent/documentState/editable" />
12 <xsl:variable name="globalReadOnly" select="/documentContent/documentState/editable != 'true'" />
13 <xsl:variable name="docStatus" select="//documentState/workflowDocumentState/status" />
14 <xsl:variable name="isAtNodeInitiated" select="my-class:isAtNode($docHeaderId, 'Initiated')" />
15 <xsl:variable name="isPastInitiated" select="my-class:isNodeInPreviousNodeList('Initiated',
$docHeaderId)" />
16 <xsl:variable name="isUserInitiator" select="my-class:isUserInitiator($docHeaderId)" />
17 <!-- <xsl:variable name="workflowUser" select="my-class:getWorkflowUser().authenticationUserId().id()" />
This has a unique implementation at IU -->
18 <xsl:param name="overrideMain" select="'true'" />
19
20 <!-- mainForm begins here. Execution of stylesheet begins here. It calls other templates which can call
other templates.
21 Position of templates beyond this point do not matter. -->
22 <xsl:template name="mainForm">
23     <html xmlns="">
24         <head>
25             <script language="javascript" />
26             <xsl:call-template name="htmlHead" />
27         </head>
28         <body onload="onPageLoad()">
29             <xsl:call-template name="errors" />
30             <!-- the header is useful because it tells the user whether they are in 'Editing' mode or 'Read
Only' mode. -->
31             <xsl:call-template name="header" />
32             <xsl:call-template name="instructions" />
33             <xsl:variable name="formTarget" select="'eDocLite '" />
34             <!-- validateOnSubmit is a javascript function (file: edoclite1.js) which supports edoclite
forms and can be somewhat complicated
35                 but does not
36  require modification unless enhancements are required. -->
37             <form action="{$formTarget}" enctype="multipart/form-data" id="edoclite" method="post"
onsubmit="return validateOnSubmit(this)">
38                 <xsl:call-template name="hidden-params" />
39                 <xsl:call-template name="mainBody" />
40                 <xsl:call-template name="notes" />
41                 <br />
42                 <xsl:call-template name="buttons" />
43                 <br />
44             </form>
45             <xsl:call-template name="footer" />
```

```
46              </body>
47         </html>
48  </xsl:template>
49
50  <!-- mainBody template begins here. It calls other templates which can call other templates. Position of
templates do not matter. -->
51  <xsl:template name="mainBody">
52      <!-- to debug, or see values of previously created variables, one can use the following format.
53         for example, uncomment the following line to see value of $docStatus. It will be rendered at the
top of the main body form. -->
54      <!-- $docStatus=<xsl:value-of select="$docStatus" /> -->
55      <!-- rest of this all is within the form table -->
56      <table xmlns="" align="center" border="0" cellpadding="0" cellspacing="0" class="bord-r-t" width="80%">
57          <tr>
58              <td align="left" border="3" class="thnormal" colspan="1">
59  <br />
60  <h3>
61  My Page
62  <br />
63  EDL EDoclite Example
64  </h3>
65  <br />
66  </td>
67              <td align="center" border="3" class="thnormal" colspan="2">
68  <br />
69  <h2>eDocLite Example 1 Form</h2></td>
70          </tr>
71          <tr>
72              <td class="headercell5" colspan="100%">
73  <b>User Information</b>
74  </td>
75          </tr>
76          <tr>
77              <td class="thnormal">
78                  <xsl:call-template name="widget_render">
79                      <xsl:with-param name="fieldName" select="'userName'" />
80                      <xsl:with-param name="renderCmd" select="'title'" />
81                  </xsl:call-template>
82                  <font color="#ff0000">*</font>
83              </td>
84              <td class="datacell">
85                  <xsl:call-template name="widget_render">
86                      <xsl:with-param name="fieldName" select="'userName'" />
87                      <xsl:with-param name="renderCmd" select="'input'" />
88                      <xsl:with-param name="readOnly" select="$isPastInitiated" />
89                  </xsl:call-template>
90              </td>
91          </tr>
92          <tr>
93              <td class="headercell5" colspan="100%">
94  <b>Other Information</b>
95  </td>
96          </tr>
97          <tr>
98              <td class="thnormal">
99                  <xsl:call-template name="widget_render">
100                     <xsl:with-param name="fieldName" select="'rqstDate'" />
101                     <xsl:with-param name="renderCmd" select="'title'" />
102                 </xsl:call-template>
103                 <font color="#ff0000">*</font>
104             </td>
105             <td class="datacell">
106                 <xsl:call-template name="widget_render">
107                     <xsl:with-param name="fieldName" select="'rqstDate'" />
108                     <xsl:with-param name="renderCmd" select="'input'" />
109                     <xsl:with-param name="readOnly" select="$isPastInitiated" />
110                 </xsl:call-template>
111             </td>
112         </tr>
113         <tr>
114             <td class="thnormal">
115                 <xsl:call-template name="widget_render">
116                     <xsl:with-param name="fieldName" select="'campus'" />
117                     <xsl:with-param name="renderCmd" select="'title'" />
118                 </xsl:call-template>
119                 <font color="#ff0000">*</font>
120             </td>
```

```
121            <td class="datacell">
122                <xsl:call-template name="widget_render">
123                    <xsl:with-param name="fieldName" select="'campus'" />
124                    <xsl:with-param name="renderCmd" select="'input'" />
125                    <xsl:with-param name="readOnly" select="$isPastInitiated" />
126                </xsl:call-template>
127            </td>
128        </tr>
129        <tr>
130            <td class="thnormal">
131                <xsl:call-template name="widget_render">
132                    <xsl:with-param name="fieldName" select="'description'" />
133                    <xsl:with-param name="renderCmd" select="'title'" />
134                </xsl:call-template>
135            </td>
136            <td class="datacell">
137                <xsl:call-template name="widget_render">
138                    <xsl:with-param name="fieldName" select="'description'" />
139                    <xsl:with-param name="renderCmd" select="'input'" />
140                    <xsl:with-param name="readOnly" select="$isPastInitiated" />
141                </xsl:call-template>
142            </td>
143        </tr>
144        <tr>
145            <td class="thnormal" colspan="2">
146 <b>(Check all that apply)</b>
147 </td>
148        </tr>
149        <tr>
150            <td class="datacell" colspan="2">
151                <xsl:call-template name="widget_render">
152                    <xsl:with-param name="fieldName" select="'fundedBy'" />
153                    <xsl:with-param name="renderCmd" select="'input'" />
154                    <xsl:with-param name="readOnly" select="$isPastInitiated" />
155                </xsl:call-template>
156                <br />
157                <xsl:call-template name="widget_render">
158                    <xsl:with-param name="fieldName" select="'researchHumans'" />
159                    <xsl:with-param name="renderCmd" select="'input'" />
160                    <xsl:with-param name="readOnly" select="$isPastInitiated" />
161                </xsl:call-template>
162                <br />
163            </td>
164        </tr>
165        <tr>
166            <td class="headercell1" colspan="100%">
167 <b>Supporting Materials</b></td>
168        </tr>
169        <tr>
170            <td class="thnormal" colspan="100%">Use the Create Note box below to attach supporting
 materials to your request. Notes may be added with or without attachments. Click the red 'save' button on the
 right.</td>
171        </tr>
172    </table>
173    <br xmlns="" />
174 </xsl:template>
175 <xsl:template name="nbsp">
176    <xsl:text disable-output-escaping="yes">&amp;nbsp;</xsl:text>
177 </xsl:template>
178 </xsl:stylesheet>
179 </style>
180
```

The beginning portion of this style sheet defines some XSL variables that are often useful to drive logic choices. For simplicity, this example uses very little logic.

The **isPastInitiated** variable drives whether a user-defined EDL field renders readOnly or not.

The **mainform** often serves to call some common widget templates that add canned functionality. The **mainform** then calls the **mainBody** template, which creates the html to render the EDL-defined fields. The **mainform** then (optionally) calls the notes, buttons, and footer templates.

The majority of your programming effort goes into the **mainBody**, where calls to **widget_render** generate much of the field-specific title and value information. Various options can be passed into **widgets_render** to allow client events to be executed. The **mainBody** is usually one or more html tables and sometimes makes calls to programmer-defined sub-templates. The XSLT stylesheet generates the HTML rendered by the browser.

The main and repeating theme of the example involves calling **widget_render** with the title of an EDL field, followed by calling **widget_render** again with the input field. Widgets are a wrapper for XSLT stylesheets that offer the ability to create HTML. Parameters offer different ways to render HTML when making calls to widgets. Note that the variable value **$isPastInitiated** is passed as a parameter to widgets_render so that the html **readOnly** attribute is generated when the form is passed the initiator's node.

Chapter 3. Lazy importing of EDL Styles

You can configure Rice to lazily import an eDocLite style into the database on demand by setting a custom configuration parameter.

- Create a custom stylesheet file, e.g. myricestyle.xml containing a style with a unique name, e.g. "xyzAppStyle" and store it in a location that is locally accessible to your application server.

- Set a configuration parameter named `edl.style.<style-name>` with the value being a path to the file containing your style. Following the example above, you would name your parameter "edl.style.xyzAppStyle".

The stylesheet file could contain a full EDL, or be a standalone EDL style. On first use of that named style by an EDL, the file will be parsed and the named style will be imported into the database. The following example contains just an eDocLite XSL stylesheet:

```
 1 <data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="ns:workflow resource:WorkflowData">
 2     <edoclite xmlns="ns:workflow/EDocLite" xsi:schemaLocation="ns:workflow/EDocLite resource:EDocLite">
 3         <style name="xyzAppStyle">
 4             <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:wf="http://xml.apache.org/xalan/java/org.kuali.rice.kew.edoclite.WorkflowFunctions">
 5                 <!-- your custom stylesheet -->
 6             </xsl:stylesheet>
 7         </style>
 8     </edoclite>
 9
10 </data>
11
```

Note that in a default Rice installation (starting in version 1.0.2), the "widgets" style is lazily imported using this mechanism. In common-config-defaults.xml (which is located in the rice-impl jar), the following parameter is defined:

```
 1 <param name="edl.style.widgets" override="false">classpath:org/kuali/rice/kew/edl/default-widgets.xml</
param>
```

If you wanted to override that file, you could define your own parameter in your Rice XML configuration file using the above example as a template, but removing the `override="false"` attribute.

Document Type

A `document type` defines the workflow process for an eDocLite. You can create hierarchies where Child document types inherit attributes of their Parents. At some level, a document type specifies routing information. The document type definition for our first example follows. It contains routing information that describes the route paths possible for a document.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="ns:workflow resource:WorkflowData">
 3     <documentTypes xmlns="ns:workflow/DocumentType" xsi:schemaLocation="ns:workflow/DocumentType
resource:DocumentType">
```

```
 4            <documentType>
 5                <name>eDoc.Example1Doctype</name>
 6                <parent>eDoc.Example1.ParentDoctype</parent>
 7                <description>eDoc.Example1 Request DocumentType</description>
 8                <label>eDoc.Example1 Request DocumentType</label>
 9                <postProcessorName>org.kuali.rice.kew.edl.EDocLitePostProcessor</postProcessorName>
10                <superUserGroupName namespace="KUALI">eDoc.Example1.SuperUsers</superUserGroupName>
11                <blanketApprovePolicy>NONE</blanketApprovePolicy>
12                <defaultExceptionGroupName namespace="KUALI">eDoc.Example1.defaultExceptions</
defaultExceptionGroupName>
13                <docHandler>${workflow.url}/EDocLite</docHandler>
14                <active>true</active>
15                <routingVersion>2</routingVersion>
16                <routePaths>
17                    <routePath>
18                        <start name="Initiated" nextNode="eDoc.Example1.Node1" />
19                        <requests name="eDoc.Example1.Node1" />
20                    </routePath>
21                </routePaths>
22                <routeNodes>
23                    <start name="Initiated">
24                        <activationType>P</activationType>
25                        <mandatoryRoute>false</mandatoryRoute>
26                        <finalApproval>false</finalApproval>
27                    </start>
28                    <requests name="eDoc.Example1.Node1">
29                        <activationType>P</activationType>
30                        <ruleTemplate>eDoc.Example1.Node1</ruleTemplate>
31                        <mandatoryRoute>false</mandatoryRoute>
32                        <finalApproval>false</finalApproval>
33                    </requests>
34                </routeNodes>
35            </documentType>
36        </documentTypes>
37 </data>
38
```

The Parent element refers to a hierarchical order of the document types. Usually, you create one Root document type with limited but common information. Then, under that, you create more specific document types. In our example, there are only two levels.

The Root document type definition for our first example:

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="ns:workflow resource:WorkflowData">
 3     <documentTypes xmlns="ns:workflow/DocumentType" xsi:schemaLocation="ns:workflow/DocumentType
resource:DocumentType">
 4            <documentType>
 5                <name>eDoc.Example1.ParentDoctype</name>
 6                <description>eDoc.Example1 Parent Doctype</description>
 7                <label>eDoc.Example1 Parent Document</label>
 8                <postProcessorName>org.kuali.rice.kew.edl.EDocLitePostProcessor</postProcessorName>
 9                <superUserGroupName namespace="KUALI">eDoc.Example1.SuperUsers</superUserGroupName>
10                <blanketApprovePolicy>NONE</blanketApprovePolicy>
11                <docHandler>${workflow.url}/EDocLite</docHandler>
12                <active>true</active>
13                <routingVersion>2</routingVersion>
14                <routePaths />
15            </documentType>
16        </documentTypes>
17 </data>
18
```

A Child document type can inherit most element values, although you must define certain element values, like **postProcessor**, for each Child document type.

A brief explanation of elements that are not intuitive is below. You can find additional element explanations by reading the Document Type Guide.

Parent DocType

`postProcessorName` - Use the default, as shown above, unless special processing is needed.

`blanketApprovePolicy` – When specified as NONE, this means that a user cannot click a single button that satisfies multiple levels of approval.

`dochandler` - Use the default, as shown above, so URLs are automatically unique in each environment, based on settings in the Application Constants (i.e., unique in each Test environment and unique again in Production).

`active` - Set this element to `false` to disable this feature.

`routingVersion` - Use the default, as shown above.

Child DocType

`name` - The name value must exactly match the value in the *EDL Association* document type element.

`parent` - The parent value must exactly match the name value of the parent document type.

`superUserGroupName` - A group of people who have special privileges that can be defined using the management service that's part of the KIM module.

`defaultExceptionGroupName` - A group of people who address a document of this type when it goes into Exception routing

`routePaths` and `routePath` - The initial elements that summarize the routing path the document will follow. In our example, an initiator fills out an eDocLite form. When the initiator submits that form, where it is routed depends on the value in the `Campus` field. There is only one destination node in our first example. The submitted form goes to either the IUB person or the IUPUI person, depending on the selection in the `Campus` field.

In most cases, a workgroup of people is the destination for an EDL form, not a single person. Workgroups are used as destinations because anyone in the workgroup can open the document, edit it, and click an `Action` button that routes the document to the next node. This prevents delays when someone is out of the office and a document awaits their action.

When the initiator submits the document, KEW adds that document to the Action List of the destination person or workgroup. The destination person or workgroup can then open the document, edit it (if any fields are available for editing), and click an `Action` button such as `Approve`, which routes the document onward. In our case, there is no further destination, so when the destination person or workgroup approves the document, the document becomes `Final` (it is finished). Some real-life examples have ten or more nodes for approvals or other actions. A document may bypass some of those nodes, depending on data placed into the form by previous participants.

`routeNodes`- Redefines the route path.

`activationType`

- **P** stands for `parallel` and is almost always used. This value makes more sense when considered from a `target node` perspective. From that perspective, it means that if a workgroup of people all received the document in their Action List, any one, in any order, can approve it. Once it is approved by anyone in the workgroup, it is routed to the next node, and KEW removes the document from the Action List of all the people in the workgroup. activationType

- **S** stands for **sequential** and is reserved for special cases where rules can specify that two or more people in a workgroup must take Action on a document, in a specific order, before KEW will route the document to the next node.

mandatoryRoute - Use **false** unless there is a special condition to solve. When this parameter is set to **true**, the document goes into exception routing if an approve request isn't generated by the ruleTemplate. This means that you are only expecting an **approve**, and nothing else.

finalApproval - Use false unless there is a special condition to solve. When this parm is set to true, the document goes into exception routing if approves are generated after this route node. This means this must be the last Action, or it will go into exception routing. (Be careful, because if this parameter is set to true and a user clicks a Return to Previous button, then the next action button clicked sends the document into exception handling.)

requests name= "..." - Defines the name of the node

ruleTemplate - A named entity type that helps define which routing rule fires. In our example, the **ruleTemplate** name is the same as the **request** name. These field values do NOT need to be the same. They are simply identifiers.

Rule Attributes

The RuleAttribute is a mechanism that can relate directly to an edl field. Most rule attributes are of the xml rule attribute type. This type uses an xpath statement which is used by the workflow engine to match to a rule that fires or does not fire.

In the below example, it can be seen that the edl defined field named 'campus' and its permissible values are defined. Then in the xpathexpression element says; when the value in the edl field named 'campus' matches the rule that contains 'IUB' the rule will fire. Or when the value in the edl field named 'campus' matches the rule that contains 'IUPUI' that rule will fire instead. Rules firing route a document to a person or a workgroup of people.

To make another rule attribute for a different field, clone this one, change all references to the field 'campus' to your different edl field name. Then cut and paste in the values section. Then in the edl definition, the new field must carry the extra syntax 'attributeName='. For example the eld definition for campus looks like this:

```
1 <fieldDef name="campus" title="Campus" workflowType="ALL">
```

Rule Routing

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="ns:workflow resource:WorkflowData">
 3     <ruleAttributes xmlns="ns:workflow/RuleAttribute" xsi:schemaLocation="ns:workflow/RuleAttribute
resource:RuleAttribute">
 4         <ruleAttribute>
 5             <name>EDL.Campus.Example</name>
 6             <className>org.kuali.rice.kew.rule.xmlrouting.StandardGenericXMLRuleAttribute</className>
 7             <label>EDL Campus Routing</label>
 8             <description>EDL School Routing</description>
 9             <type>RuleXmlAttribute</type>
10             <routingConfig>
11                 <fieldDef name="campus" title="Campus" workflowType="ALL">
12                     <display>
13                         <type>select</type>
14                         <values title="IUB">IUB</values>
```

```
15                        <values title="IUPUI">IUPUI</values>
16                    </display>
17                    <validation required="false" />
18                    <fieldEvaluation>
19                        <xpathexpression>//campus = wf:ruledata('campus')</xpathexpression>
20                    </fieldEvaluation>
21                </fieldDef>
22                <xmlDocumentContent>
23                    <campus>%campus%</campus>
24                </xmlDocumentContent>
25            </routingConfig>
26        </ruleAttribute>
27    </ruleAttributes>
28
29 </data>
30
```

Rule attributes can have a different types such a searchable, but this type does not have to do with routing. Instead it relates to additional columns that are displayed in doc search for a particular doc type.

Ingestion Order

Many components can go in at any time, but it is advisable to follow a pattern to minimize the conflicts that can occur. A few pieces are co-dependent.

1. Basic Components:

2. Widgets.xml (If changed or not previously in the environment)

3. Kim Group(s)

4. Rule Attributes

5. Rule Template(s)

6. Parent Doctype (often no routing so data is more generic, but do put routing here if children will use common routing.)

7. Children Doctype(s) (routing defined here or on Parent)

8. EDL Form

9. Rule routing rule (Used if rules are created; explained later- 1 per parent doctype)

10. Rules (Create or Ingest)

11. Anything else - Like optional custom Email Stylesheet

Chapter 4. eDocLite Lookup

Use the **eDocLite Lookup** screen to quickly find basic information about eDocLite documents and as the first step in creating a new eDocLite.

Figure 4.1. Workflow Channel: eDocLite Link
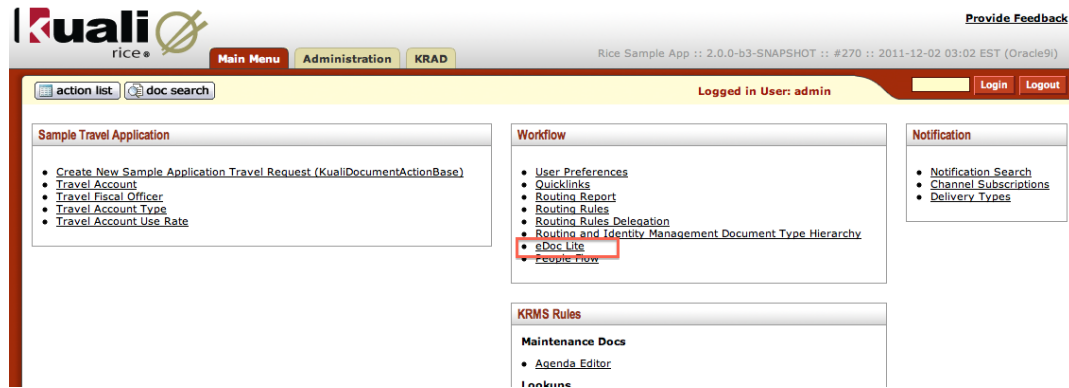
Finding the eDocLite Lookup Screen

You can go to the **eDocLite Lookup** by:

1. Click the Main Menu tab

2. Look in the Workflow section

3. Click eDoc Lite

eDocLite Lookup

Figure 4.2. eDocLite Lookup
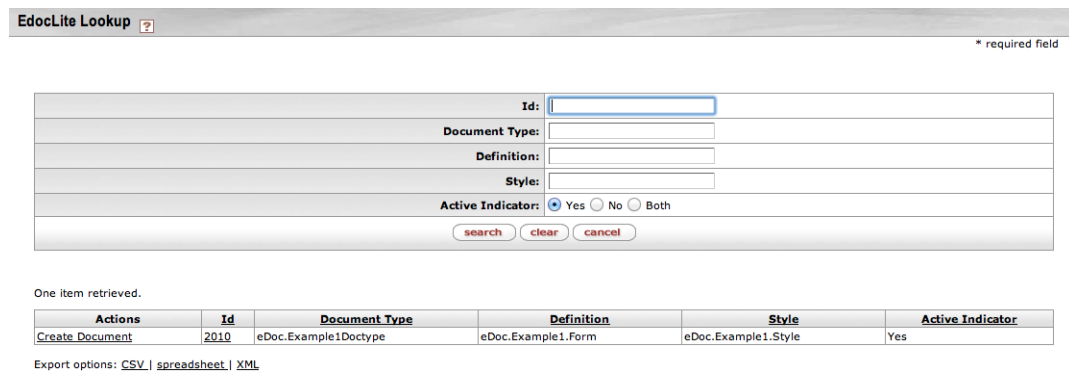
On the eDocLite lookup page, users can search for an eDocLite document based on several criteria:

Table 4.1. eDocLite Lookup Attributes

Field	Description
Id	The unique ID number assigned to each document.

Field	Description
Document Type	The name of the document type, which is specified in the Document Type attribute of an eDocLite.
Definition	The name of the EDL XML definition.
Style	The style specified in the EDL XML file is the style attribute of an eDocLite. Generally an EDL XML file has only one definition name which relates to one and only one style name.
Active Indicator	You have the choice of searching by the active status of an eDocLite.

You can use the above criteria to limit your search results. A search will produce a list of one or more results that look similar to the following:

Figure 4.3. eDocLite Lookup: Search Results

Actions	Id	Document Type	Definition	Style	Active Indicator
Create Document	2010	eDoc.Example1Doctype	eDoc.Example1.Form	eDoc.Example1.Style	Yes
Create Document	2024	InterviewRequest	InterviewRequestForm	InterviewRequestStyle	Yes
Create Document	2027	OfferRequest	OfferRequestForm	OfferRequestStyle	Yes
Create Document	2030	SearchStatus	SearchStatusForm	SearchStatusStyle	Yes
Create Document	2033	VacancyNotice	VacancyNoticeForm	VacancyNoticeStyle	Yes
Create Document	2036	WaiverRequest	WaiverRequestForm	WaiverRequest_xsl	Yes

Clicking **Create Document** on any line takes you to the eDocLite document screen where new documents can be created. The line item you choose will result in that document being used as a template for the new one you are creating. More information on this follows in the section called **Create New eDocLite Document**.

Clicking any underlined Id will take you to the **eDocLite Inquiry** screen for that document.

Note

Exporting the output list to XML will give you the option of viewing the XML used to produce the list returned from the search.

Standard in KEW there is one eDocLite for electronic routing, and new eDocLites can be added based on business requirements. Some of the functions that eDocLites are used for in business include:

- Applicant Monitoring

- BLSC Request

- Course Change Request

- Grade Replacement Request

- Internship Contract

- Interview Request

- Mass Mailing Request

- Offer Request

- Program Plan Update

- REGR Access Request

- Removal Of Incomplete

- Revenue Producing Activity

- SUDS Request Document Type

- Search Status

- Special Credit Request

- Student Trip

- User Agreement

- Unit Change Request

- Vacancy Notice

- Vehicle Replacement

- Waiver Request

- New Course Request

Chapter 5. eDocLite Inquiry

Figure 5.1. eDocLite Inquiry

EdocLite	▼ hide
Id:	2010
Document Type:	eDoc.Example1Doctype
Definition:	eDoc.Example1.Form
Style:	eDoc.Example1.Style
Active Indicator:	Yes

export close
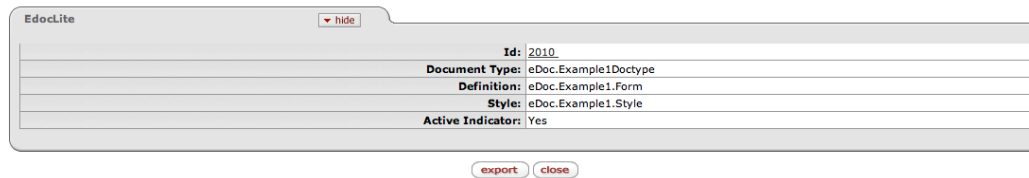
The Inquiry screen displays the same information as is found on a line of the Lookup output, but this screen provides you with the export option. Exporting from the Inquiry screen produces a XML file of the source for the eDocLite document.

Chapter 6. Create New eDocLite Document

To create a new eDocLite document to be routed in KEW, click on **Create Document** in the row for eDocLite type wanted. It will take users to different forms of eDocLites depending on the document function, but they all have three general parts:

- Document header

- Document body

- Routing action and annotation, and note area

Document Header

The Document Header contains the following fields:

Table 6.1. Document Header Attributes

Field	Description
Document Type	The name defined by the document creator of this type of eDocLite.
Document Status	The status of this document based on its routing status.
Create Date	The date and time this document is created.
Document ID	The unique system generated ID for this document.

Document body

This portion of the document is where the user identifies the routing and complete business function.

Table 6.2. Document Body Attributes

Field	Description
Title	specifies the actions users are taking on the EDocLite forms, including **editing** and **reviewing**. Other general information can be stored here such as contact information, important notes, etc.
Form	Renders form fields and input areas for the user to complete information required, depends upon the specific eDocLite requirements.

Routing Action and Annotation, and Note area

This area is used to add annotation and choose action to be taken on this eDocLite document. Annotation is the comments associated with the routing process. You can add them to different nodes of the routing process and take actions on an eDocLite by adding annotations. The annotation appears in the route log of eDocLite as comments. Notes are the comments associated with this specific eDocLite form and appear with the form and not in the route log.

Table 6.3. Routing Action and Annotation, and Note Attributes

Field	Description
Set annotation	The area to add annotation.
Action buttons	- route: begins and continues routing the eDocLite document. - save: saves the information currently on the eDocLite document.

Field	Description
	• cancel: cancels the actions on this eDocLite document, and the information on the form is not saved.
create note	Area where users can add notes and attach documents to this eDocLite form. This part keeps track of **Author, Date and time**, and the **Note** added. Users have the right to add, edit and delete the notes they create.

The following is one example of an eDocLite form:

Note

Notes that have been added to an eDocLite document can be edited or deleted.

Extendable functions

eDocLites are highly customizable. New eDocLites can be designed for new business and functional requirements. The document header and routing annotation and notes parts will be same or similar, the form will be different.

eDocLites can be designed to include following functions:

Restricted read/write rights

• Some fields in eDocLite can be set as `GlobalReadOnly`. With this setting they are disabled and can't be edited by any user other than the author.

• Some fields in eDocLite can be set as `ReadOnly`, but users with rights can still edit them. After the initiator writes them they are disabled and become locked to some of the users in the routing process. But for users with proper rights in certain nodes in the routing process, the fields will become editable again. These users can take actions on them, such as modify, add, and return to a former routing node.

Hidden fields

To accommodate some business requirements, certain fields and notes can be hidden from certain nodes in the routing process. For instance, some administrative notes on a course waiver request will be hidden from students when s/he gets the eDocLite form in the final stage of the routing process.
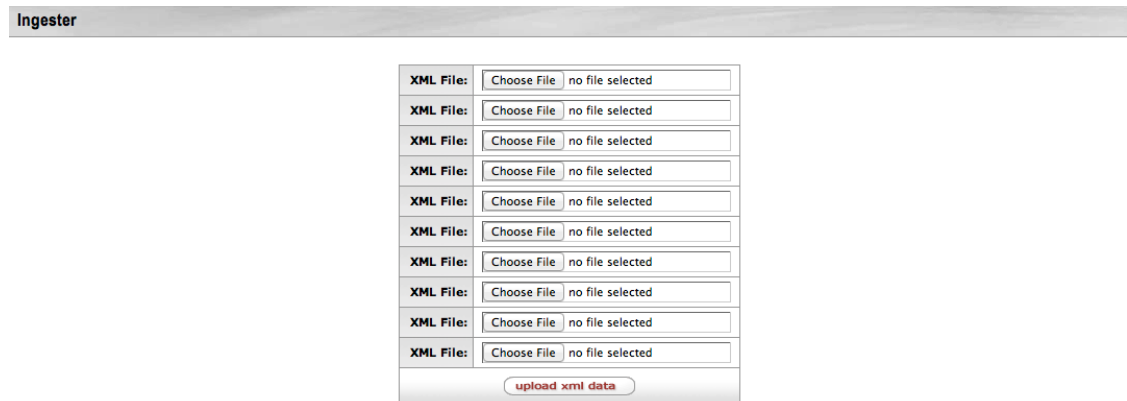
Chapter 7. XML Ingestion

KEW relies on XML for data population and routing configuration. **XML Ingester** is available from the **Administrator** channel in the portal. This allows import of various KEW components from XML, such as DocumentTypes, RuleAttributes, Rules, Workgroups, and more.

Uploading an eDocLite form

To upload XML, from the Administration menu, in the Workflow section, click on `XML Ingester` and select the (one or more) XML file(s) that you want to import:
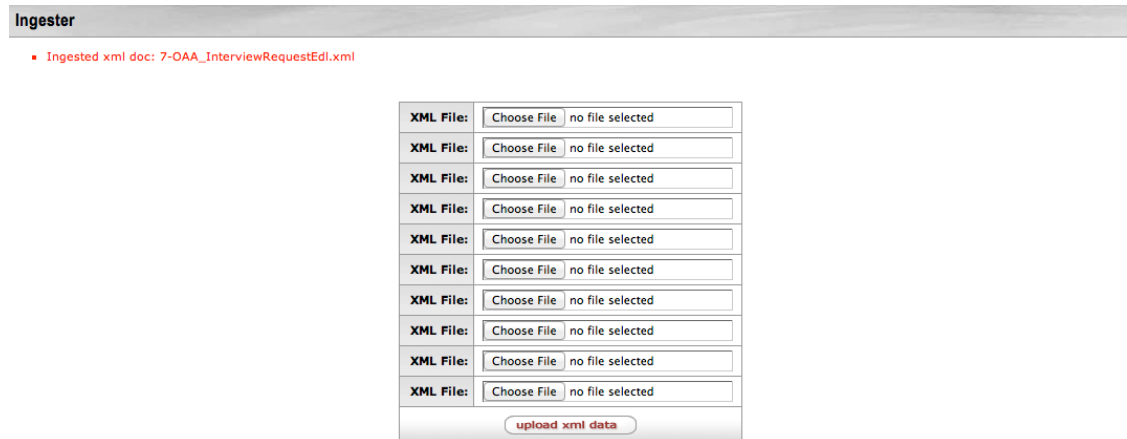
Figure 7.1. Ingester

After upload, notice the statement in red, `Ingested xml doc: <name of file>`:

Figure 7.2. Ingestion Complete

Ingestion Order

Many components can go in at any time, but it is advisable to follow a pattern to minimize the conflicts that can occur. A few pieces are co-dependent.

1. Basic Components:

2. Widgets.xml (If changed or not previously in the environment)

3. Kim Group(s)

4. Rule Attributes

5. Rule Template(s)

6. Parent Doctype (often no routing so data is more generic, but do put routing here if children will use common routing.)

7. Children Doctype(s) (routing defined here or on Parent)

8. EDL Form

9. Rule routing rule (Used if rules are created; explained later- 1 per parent doctype)

10. Rules (Create or Ingest)

11. Anything else - Like optional custom Email Stylesheet

A set of sample eDocLite documents you can upload and start using from the Developer Documentation wiki page at [this] location.

Note

Please ingest the samples in the correct order, by title (which includes number). That is:

01-OAA_Workgroups.xml
02-OAA_Attributes.xml
03-OAA_RuleTemplates.xml
04-OAA_ParentDocumentType.xml
05-OAA_ChildDoctypes.xml
06-OAA_Rules.xml
07-OAA_InterviewRequestEdl.xml
08-OAA_OfferRequestEdl.xml
09-OAA_SearchStatusEdl.xml
10-OAA_VacancyNoticeEdl.xml
11-OAA_WaiverRequestEdl.xml
You can ingest the first 10 at one time on the first screen, if they are ordered as above.

The [examples] section of this guide looks at some of these.

Chapter 8. eDocLite Examples

Example 1 Form

This simple first example form can be used to gather data from an initiator, then route the document when the submit button is pressed to one of two nodes depending on the value the initiator selects from a field called campus. It makes use of several common html field types. Any number of notes can be added as well as any number of attachments. In this example, the campus field has two possible selection choices (only first on shown in the picture).

Interview Request Form

The Interview Request document is to be filled out by the department initiating the search and is used to declare persons to be interviewed.

If more than 8 Candidates are to be interviewed, an additional Interview Request should be created.
The OAA number used is the same as the one provided by the Office Affirmative Action for the corresponding Vacancy Notice.
A separate section towards the bottom is presented for Applicant Pool data. This section is to be completed only by the OAA group. After they have entered the information, this data will be available for viewing when it is routed back for acknowledgement or by using the document search option to view the document

Offer Request Form

The Offer Request is to be filled out by the department initiating the search and is used to document a potential offer to the candidate.

- The OAA number used is the same as the one for the corresponding Vacancy Notice.

- You may wish to attach a letter such as an offer to recommend an appointment.

After you#ve created the letter (which may be another eDocLite document), you need to attach it to the document for review at the approval levels. Multiple versions of the letter can be created and attached to this document. Then attach the letter(s).

Click on the **Choose File** button in the Create a Note section towards the bottom of the document. Select the file and then click the **save** button.

Finally, click the **save** button to save your document for further changes. If it is ready for approvals, then click the **submit** button instead.

Search Status Form

The Search Status document is to be filled out by the department initiating the search and is used to document the final results of the search. The OAA number used is the same as the one for the corresponding Vacancy Notice or Waiver Request.

In the event that the Offer Status is selected as Declined or No Offer was made, a new list of values is provided to outline what the next steps for this search will be.

Vacancy Form

The Vacancy Notice document is to be filled out by the department who is initiating a search for a vacant position.

The first field on the document is an OAA number; this is not a required field and will be entered when the document reaches the Office of Affirmative Action for approval.

A field provided for instances where a prior search is being continued is provided. The field, "Reactivating Prior Year Search: OAA#", is used for linking this new search to the prior one by providing the prior OAA #.

A separate section towards the bottom is presented for Utilization data. This section is to be completed only by the OAA group. After they have entered the information, this data will be available for viewing using the document search option to view the document.

Waiver Request

The Waiver Request document is similar to the Vacancy Notice in that the department initiating the search completes it, but in this case the department is also requesting a waiver from the normal academic search requirements.

The first field on the document is an OAA number; this is not a required field and will be entered when the document reaches the Office of Affirmative Action for approval.

You may wish to attach a letter such as an offer to recommend an appointment.

After you've created the letter (which may be another eDocLite document), you need to attach it to the document for review at the approval levels. Multiple versions of the letter can be created and attached to this document. Then attach the letter(s).
Click on the **Choose File** button in the Create a Note section towards the bottom of the document.

Select the file and then click the **save** button.

Finally, click the **save** button to save your document for further changes. If it is ready for approvals, then click the **submit** button instead.

Chapter 9. eDocLite Support Notes

eDocLite Supporting Material

- The Notes section of widgets allows notes to be saved and attachments to be uploaded/removed while the document is editable. An Application Constant called `saveAttachments` must be set to true before the attachments portion of the Notes will render. Application Constants can be accessed from the main workflow menu, near the same location where the Ingester link appears. I believe the server must be restarted to invoke Application Constant changes.

- Route Paths describe the flow of the document between various individuals or groups. Each Route path has a further description called a routeNode.

- Route Nodes:

  activationType should be P for parallel, rather than S for sequential. S forces sequential routing which means that if a document is sent to two different nodes, it will only appear in the action list in of the first, and the second will only see it after the first has approved it.
  Each Route Node other than the first has a `ruleTemplate` element.
  Rule Templates define rules for a node.
  They may have 0-n* number of rule attributes. No attributes means to go ahead and route document to next node if approved.

- In a Rule Template zero to N number of rule attributes can be listed. For EDocLite there are two types of rule attributes: Searchable (used for searching not used for routing), and RuleXmlAttribute (used to drive workflow engine to determine where the form should be routed). We will focus on the Xml type. Use the admin menu 'Rule Attribute' to report on it. To be used, Rule Attributes must be defined in the EDL after the `<fieldDef`. For example: `<fieldDef attributeName="UGSRemonstranceFiled" name="remonstranceFiled" title="Remonstrance Filed?">`

- Fields that can participate in a Rule Attribute are often drop down selection fields but do not have to be.

- It is important to understand how the workflow engine figures out where to route a document. For each Route Node defined for the doctype:

  - The name of the rule template is used to access the rule template. If the rule Template contains a rule attribute name then the rule attribute is accessed.

  - The rule attribute usually contains an `xpathexpression` element. The left side of the equals sign accesses the form value of this field.

  - On the right side of the equals sign: The java workflow engine access each rule (keyed by doctype + rule template) and attempts to match the rule data to the form data. If no match is found, the workflow engine skips this route node. If a match is found the rule tells the workflow engine what workgroup to route the document to.

  - If the rule template does NOT contain any rule attributes, then all forms stop at this node (referred to as 'AUTO-STOP').

- To review what rules exist for a doctype one can find them in the Main Menu tab under the Workflow section. The link to click is `Quicklinks`. For the doctype that carries the route paths, click search. Then without adding any other info click search. If further info is required for the page - you will be

told so. There is likely a rule that is active for the doctype/ruletemplate combination. A report on that rule will show the 'Reviewer' which normally is a workgroup where the document should be routed.

- When changing a Rule Template, like adding another rule attribute, inactivate existing rules first because old rules do NOT automatically deactivate. In the route log of a document, one can examine the rules (by id number) that fired. This is one way to identify if old rules that are not expected or desired to fire are in fact firing.

- To change a rule template

  export the xml, modify it accordingly, then add an attribute to the **`<ruleTemplate>`** making it look like: **`<ruleTemplate allowOverwrite="true" >`**. This is a safety feature, because when a template is changed the rules behind it are affected. Note that the schema definition file is named RuleTemplate.xsd in Eclipse; which describes what is valid for the xml construct. If you added a rule attribute to the template, validate the rule attribute exists, or create it new if you must.
  Next you would create the routing rule(s) for the modified doctype/ruleTemplate.

- RuleAttributes can also be used for verification purposes by associating them with a java method. Unique code can be written to perform the logic to do the validation (usually involves hitting a database). See class workflowAttribute -> method validateClientRoutingData.

Glossary

A

Action List
A list of the user's notification and workflow items. Also called the user's Notification List. Clicking an item in the Action List displays details about that notification, if the item is a notification, or displays that document, if it is a workflow item. The user will usually load the document from their Action List in order to take the requested action against it, such as approving or acknowledging the document.

Action List Type
This tells you if the Action List item is a notification or a more specific workflow request item. When the Action List item is a notification, the Action List Type is "Notification."

Action Request
A request to a user or Workgroup to take action on a document. It designates the type of action that is requested, which includes:

- Approve: requests an approve or disapprove action.

- Complete: requests a completion of the contents of a document. This action request is displayed in the Action List after the user saves an incomplete document.

- Acknowledge: requests an acknowledgment by the user that the document has been opened - the doc will not leave the Action List until acknowledgment has occurred; however, the document routing will not be held up and the document will be permitted to transaction into the processed state if neccessary.

- FYI: a notification to the user regarding the document. Documents requesting FYI can be cleared directly from the Action List. Even if a document has FYI requests remaining, it will still be permitted to transition into the FINAL state.

Action Request Hierarchy
Action requests are hierarchical in nature and can have one parent and multiple children.

Action Requested
The action one needs to take on a document; also the type of action that is requested by an Action Request. Actions that may be requested of a user are:

- Acknowledge: requests that the users states he or she has reviewed the document.

- Approve: requests that the user either Approve or Disapprove a document.

- Complete: requests the user to enter additional information in a document so that the content of the document is complete.

- FYI: intended to simply makes a user aware of the document.

Action Taken
An action taken on a document by a *Reviewer* in response to an Action Request. The Action Taken may be:

- Acknowledged: Reviewer has viewed and acknowledged document.

- Approved: Reviewer has approved the action requested on document.

- Blanket Approved: Reviewer has requested a blanket approval up to a specified point in the route path on the document.

- Canceled: Reviewer has canceled the document. The document will not be routed to any more reviewers.

- Cleared FYI: Reviewer has viewed the document and cleared all of his or her pending FYI(s) on this document.

- Completed: Reviewer has completed and supplied all data requested on document.

- Created Document: User has created a document

- Disapproved: Reviewer has disapproved the document. The document will not being routed to any subsequent reviewers for approval. Acknowledge Requests are sent to previous approvers to inform them of the disapproval.

- Logged Document: Reviewer has added a message to the Route Log of the document.

- Moved Document: Reviewer has moved the document either backward or forward in its routing path.

- Returned to Previous Node: Reviewer has returned the document to a previous routing node. When a Reviewer does this, all the actions taken between the current node and the return node are removed and all the pending requests on the document are deactivated.

- Routed Document: Reviewer has submitted the document to the workflow engine for routing.

- Saved: Reviewer has saved the document for later completion and routing.

- Superuser Approved Document: *Superuser* has approved the entire document, any remaining routing is cancelled.

- Superuser Approved Node: Superuser has approved the document through all nodes up to (but not including) a specific node. When the document gets to that node, the normal Action Requests will be created.

- Superuser Approved Request: Superuser has approved a single pending Approve or Complete Action Request. The document then goes to the next routing node.

- Superuser Cancelled: Superuser has canceled the document. A Superuser can cancel a document without a pending Action Request to him/her on the document.

- Superuser Disapproved: Superuser has disapproved the document. A Superuser can disapprove a document without a pending Action Request to him/her on the document.

- **Superuser Returned to Previous Node:** Superuser has returned the document to a previous routing node. A Superuser can do this without a pending Action Request to him/her on the document.

Activated	The state of an action request when it is has been sent to a user's Action List.
Activation	The process by which requests appear in a user's Action List
Activation Type	Defines how a route node handles activation of Action Requests. There are two standard activation types:

- **Sequential:** Action Requests are activated one at a time based on routing priority. The next Action Request isn't activated until the previous request is satisfied.

- **Parallel:** All Action Requests at the route node are activated immediately, regardless of priority

Active Indicator	An indicator specifying whether an object in the system is active or not. Used as an alternative to complete removal of an object.
Ad Hoc Routing	A type of routing used to route a document to users or groups that are not in the Routing path for that Document Type. When the Ad Hoc Routing is complete, the routing returns to its normal path.
Annotation	Optional comments added by a *Reviewer* when taking action. Intended to explain or clarify the action taken or to advise subsequent Reviewers.
Approve	A type of workflow action button. Signifies that the document represents a valid business transaction in accordance with institutional needs and policies in the user's judgment. A single document may require approval from several users, at multiple route levels, before it moves to final status.
Approver	The user who approves the document. As a document moves through Workflow, it moves one route level at a time. An Approver operates at a particular route level of the document.
Attachment	The pathname of a related file to attach to a Note. Use the "Browse..." button to open the file dialog, select the file and automatically fill in the pathname.
Attribute Type	Used to strongly type or categorize the values that can be stored for the various attributes in the system (e.g., the value of the arbitrary key/value pairs that can be defined and associated with a given parent object in the system).
Authentication	The act of logging into the system. The Out of the box (OOTB) authenticaton implementation in Rice does not require a password as it is intended for testing puposes only. This is something that must be enabled as part of an implementation. Various authentication solutions exist, such as CAS or Shibboleth, that an implementer may want to use depending on their needs.
Authorization	Authorization is the permissions that an authenticated user has for performing actions in the system.
Author Universal ID	A free-form text field for the full name of the Author of the Note, expressed as "Lastname, Firstname Initial"

B

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

- Active: A true/false flag to indicate if the Routing Rule is active. If false, then the rule will not be evaluated during routing.

- Document Type: The *Document Type* to which the Routing Rule applies.

- From Date: The inclusive start date from which the Routing Rule will be considered for a match.

- Force Action: a true/false flag to indicate if the review should be forced to take action again for the requests generated by this rule, even if they had taken action on the document previously.

- Name: the name of the rule, this serves as a unique identifier for the rule. If one is not specified when the rule is created, then it will be generated.

- Rule Template: The Rule Template used to create the Routing Rule.

- To Date: The inclusive end date to which the Routing Rule will be considered for a match.

Blanket Approval	Authority that is given to designated *Reviewers* who can approve a document to a chosen route point. A Blanket Approval bypasses approvals that would otherwise be required in the *Routing* For an authorized Reviewer, the *Doc Handler* typically displays the Blanket Approval button along with the other options. When a Blanket Approval is used, the Reviewers who are skipped are sent Acknowledge requests to notify them that they were bypassed.
Blanket Approve Workgroup	A workgroup that has the authority to Blanket Approve a document.
Branch	A path containing one or more Route Nodes that a document traverses during routing. When a document enters a *Split Node* multiple branches can be created. A *Join Node* joins multiple branches together.
Business Rule	1. Describes the operations, definitions and constraints that apply to an organization in achieving its goals.
	2. A restriction to a function for a business reason (such as making a specific object code unavailable for a particular type of disbursement). Customizable business rules are controlled by Parameters.

C

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

Canceled	A routing status. The document is denoted as void and should be disregarded.
CAS - Central Authentication Service	http://www.jasig.org/cas - An open source authentication framework. Kuali Rice provides support for integrating with CAS as an authentication provider (among other authentication solutions) and also provides an implementation of a CAS server that integrates with Kuali Identity Management.
Client	A Java Application Program Interface (API) for interfacing with the Kuali Enterprise Workflow Engine.
Client/Server	The use of one computer to request the services of another computer over a network. The workstation in an organization will be used to initiate a business transaction (e.g., a budget transfer). This workstation needs to gather information from a remote database to process the transaction, and will eventually be used to post new or changed information back onto that remote database. The workstation is thus a Client and the remote computer that houses the database is the Server.
Close	A workflow action available on documents in most statuses. Signifies that the user wishes to exit the document. No changes to Action Requests, Route Logs or document status occur as a result of a Close action. If you initiate a document and close it without saving, it is the same as canceling that document.
Comma-separated value	A file format using commas as delimiters utilized in import and export functionality.
Complete	A pending action request to a user to submit a saved document.
Completed	The action taken by a user or group in response to a request in order to finish populating a document with information, as evidenced in the Document Route Log.
Country Restricted Indicator	Field used to indicate if a country is restricted from use in procurement. If there is no value then there is no restriction.
Creation Date	The date on which a document is created.
CSV	See *comma-separated value*

D

Date Approved	The date on which a document was most recently approved.
Date Finalized	The date on which a document enters the FINAL state. At this point, all approvals and acknowledgments are complete for the document.
Deactivation	The process by which requests are removed from a user's *Action List*
Delegate	A user who has been registered to act on behalf of another user. The Delegate acts with the full authority of the Delegator. Delegation may be either *Primary Delegation* or *Secondary Delegation*.
Delegate Action List	A separate Action List for Delegate actions. When a Delegate selects a Delegator for whom to act, an Action List of all documents sent to the Delegator is displayed.

For both *Primary* and *Secondary Delegation* the Delegate may act on any of the entries with the full authority of the Delegator.

Disapprove	A workflow action that allows a user to indicate that a document does not represent a valid business transaction in that user's judgment. The initiator and previous approvers will receive Acknowledgment requests indicating the document was disapproved.
Disapproved	A status that indicates the document has been disapproved by an approver as a valid transaction and it will not generate the originally intended transaction.
Doc Handler	The Doc Handler is a web interface that a Client uses for the appropriate display of a document. When a user opens a document from the Action List or Document Search, the Doc Handler manages access permissions, content format, and user options according to the requirements of the Client.
Doc Handler URL	The URL for the *Doc Handler*.
Doc Nbr	See *Document Number*.
Document	Also see *E-Doc*.

An electronic document containing information for a business transaction that is routed for Actions in KEW. It includes information such as Document ID, Type, Title, Route Status, Initiator, Date Created, etc. In KEW, a document typically has XML content attached to it that is used to make routing decisions.

Document Id	See *Document Number*.
Document Number	A unique, sequential, system-assigned number for a document
Document Operation	A workflow screen that provides an interface for authorized users to manipulate the XML and other data that defines a document in workflow. It allows you to access and open a document by Document ID for the purpose of performing operations on the document.
Document Search	A web interface in which users can search for documents. Users may search by a combination of document properties such as Document Type or Document ID, or by more specialized properties using the Detailed Search. Search results are displayed in a list similar to an Action List.
Document Status	See also *Route Status*.
Document Title	The title given to the document when it was created. Depending on the Document Type, this title may have been assigned by the Initiator or built automatically based on the contents of the document. The Document Title is displayed in both the Action List and Document Search.
Document Type	The Document Type defines the routing definition and other properties for a set of documents. Each document is an instance of a Document Type and conducts the same type of business transaction as other instances of that Document Type.

Document Types have the following characteristics:

- They are specifications for a document that can be created in KEW

- They contain identifying information as well as policies and other attributes

- They defines the Route Path executed for a document of that type (Process Definition)

- They are hierarchical in nature may be part of a hierarchy of Document Types, each of which inherits certain properties of its *Parent Document Type*.

- They are typically defined in XML, but certain properties can be maintained from a graphical interface

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

E

ECL	1. An acronym for Educational Community License.
	2. All Kuali software and material is available under the Educational Community License and may be adopted by colleges and universities without licensing fees. The open licensing approach also provides opportunities for support and implementation assistance from commercial affiliates.
E-Doc	An abbreviation for electronic documents, also a shorthand reference to documents created with eDocLite.
eDocLite	A framework for quickly building workflow-enabled documents. Allows you to define document screens in XML and render them using XSL style sheets.
Embedded Client	A type of client that runs an embedded workflow engine.
Employee Status	Found on the Person Document; defines the employee's current employment classification (for example, "A" for Active).
Employee Type	Found on the Person Document; defines the employee's position classification (for example, "P" for Professional).

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

F

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

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

Force Action

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

FYI

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

G

Group

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

Groups can be given authorization to perform actions within applications by assigning them as members of *Roles*.

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

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

Group Attribute

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

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

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

H

Hierarchical Tree Structure

A hierarchical representation of data in a graphical form.

I

Initialized

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

Initiated

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

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

J

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

K

KC - Kuali Coeus	TODO
KCA - Kuali Commercial Affiliates	A designation provided to commercial affiliates who become part of the Kuali Partners Program to provide for-fee guidance, support, implementation, and integration services related to the Kuali software. Affiliates hold no ownership of Kuali intellectual property, but are full KPP participants. Affiliates may provide packaged versions of Kuali that provide value for installation or integration beyond the basic Kuali software. Affiliates may also offer other types of training, documentation, or hosting services.
KCB – Kuali Communications Broker	KCB is logically related to KEN. It handles dispatching messages based on user preferences (email, SMS, etc.).
KEN - Kuali Enterprise Notification	A key component of the Enterprise Integration layer of the architecture framework. Its features include: • Automatic Message Generation and Logging • Message integrity and delivery standards • Delivery of notifications to a user's Action List
KEW – Kuali Enterprise Workflow	Kuali Enterprise Workflow is a general-purpose electronic routing infrastructure, or workflow engine. It manages the creation, routing, and processing of electronic documents (eDocs) necessary to complete a transaction. Other applications can also use Kuali Enterprise Workflow to automate and regulate the approval process for the transactions or documents they create.
KFS – Kuali Financial System	Delivers a comprehensive suite of functionality to serve the financial system needs of all Carnegie-Class institutions. An enhancement of the proven functionality of Indiana University's Financial Information System (FIS), KFS meets GASB and FASB standards while providing a strong control environment to keep pace with advances in both technology and business. Modules include financial transactions, general ledger, chart of accounts, contracts and grants, purchasing/accounts payable, labor distribution, budget, accounts receivable and capital assets.
KIM – Kuali Identity Management	A Kuali Rice module, Kuali Identity Management provides a standard API for persons, groups, roles and permissions that can be implemented by an institution. It also provdies an out of the box reference implementation that allows for a university to use Kuali as their Identity Management solution.

KNS – Kuali Nervous System	A core technical module composed of reusable code components that provide the common, underlying infrastructure code and functionality that any module may employ to perform its functions (for example, creating custom attributes, attaching electronic images, uploading data from desktop applications, lookup/ search routines, and database interaction).
KPP - Kuali Partners Program	The Kuali Partners Program (KPP) is the means for organizations to get involved in the Kuali software community and influence its future through voting rights to determine software development priorities. Membership dues pay staff to perform Quality Assurance (QA) work, release engineering, packaging, documentation, and other work to coordinate the timely enhancement and release of quality software and other services valuable to the members. Partners are also encouraged to tender functional, technical, support or administrative staff members to the Kuali Foundation for specific periods of time.
KRAD - Kuali Rapid Application Development	TODO
KRMS - Kuali Rules Management System	TODO
KS - Kuali Student	Delivers a means to support students and other users with a student-centric system that provides real-time, cost-effective, scalable support to help them identify and achieve their goals while simplifying or eliminating administrative tasks. The high-level entities of person (evolving roles-student, instructor, etc.), time (nested units of time - semesters, terms, classes), learning unit (assigned to any learning activity), learning result (grades, assessments, evaluations), learning plan (intentions, activities, major, degree), and learning resources (instructors, classrooms, equipment). The concierge function is a self-service information sharing system that aligns information with needs and tasks to accomplish goals. The support for integration of locally-developed processes provides flexibility for any institution's needs.
KSB – Kuali Service Bus	Provides an out-of-the-box service architecture and runtime environment for Kuali Applications. It is the cornerstone of the Service Oriented Architecture layer of the architectural reference framework. The Kuali Service Bus consists of:

- A services registry and repository for identifying and instantiating services

- Run time monitoring of messages

- Support for synchronous and asynchronous service and message paradigms

Kuali	1. Pronounced "ku-wah-lee". A partnership organization that produces a suite of community-source, modular administrative software for Carnegie-class higher education institutions. See also *Kuali Foundation*
	2. (n.) A humble kitchen wok that plays an important role in a successful kitchen.
Kuali Foundation	Employs staff to coordinate partner efforts and to manage and protect the Foundation's intellectual property. The Kuali Foundation manages a growing portfolio of enterprise software applications for colleges and universities. A lightweight Foundation staff coordinates the activities of Foundation members for critical software development and coordination activities such as source code control, release engineering, packaging, documentation, project management,

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

Kuali Rice

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

L

Last Modified Date

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

M

Maintenance Document

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

Message

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

Message Queue

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

N

Namespace

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

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

Namespaces can be maintained at runtime through a maintenance document.

Note Text

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

Notification Content

This section of a *notification message* which displays the actual full message for the notification along with any other content-type-specific fields.

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

O

OOTB	Stands for "out of the box" and refers to the base deliverable of a given feature in the system.
Optimistic Locking	A type of "locking" that is placed on a database row by a process to prevent other processes from updating that row before the first process is complete. A characteristic of this locking technique is that another user who wants to make modifications at the same time as another user is permitted to, but the first one who submits their changes will have them applied. Any subsequent changes will result in the user being notified of the optimistic lock and their changes will not be applied. This technique assumes that another update is unlikely.
Optional Rule Extension Attribute	An Extension Attribute that is not required in a Rule Template. It may or may not be present in a *Routing Rule* created from the Template. It can be used as a conditional element to aid in deciding if a Rule matches. These Attributes are simply additional criteria for the Rule matching process.
Org Doc #	The originating document number.
Organization	Refers to a unit within the institution such as department, responsibility center, campus, etc.
Organization Code	Represents a unique identifier assigned to units at many different levels within the institution (for example, department, responsibility center, and campus).

P

Parameter Component Code	Code identifying the parameter Component.
Parameter Description	This field houses the purpose of this parameter.
Parameter Name	This will be used as the identifier for the parameter. Parameter values will be accessed using this field and the namespace as the key.
Parameter Type Code	Code identifying the parameter type. Parameter Type Code is the primary key for its' table.
Parameter Value	This field houses the actual value associated with the parameter.
Parent Document Type	A Document Type from which another *Document Type* derives. The child type can inherit certain properties of the parent type, any of which it may override. A Parent Document Type may have a parent as part of a hierarchy of document types.
Parent Rule	A Routing Rule in KEW from which another Routing Rule derives. The child Rule can inherit certain properties of the parent Rule, any of which it may override. A Parent Rule may have a parent as part of a hierarchy of Rules.
Permission	Permissions represent fine grained actions that can be mapped to functionality within a given system. Permissions are scoped to *Namespace* which roughly correlate to modules or sections of functionality within a given system.

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

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

Permissions are aggregated by *Roles*.

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

Attributes

1. Id - a system generated unique identifier that is the primary key for any Permission record in the system

2. Name - the name of the permission; also a human understandable unique identifier

3. Description - a full description of the purpose of the Permission record

4. Namespace - the reference to the associated *Namespace*

Relationships

1. Permission to *Role* - many to many; this relationship ties a Permission record to a Role that is authorized for the Permission

2. Permission to *Namespace* - many to one; this relationship allows for scoping of a Permission to a Namespace that contains functionality which keys its authorization checking off of said

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

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

R

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

Route Header Id	Another name for the *Document Id*.
Route Log	Displays information about the routing of a document. The Route Log is usually accessed from either the Action List or a Document Search. It displays general document information about the document and a detailed list of Actions Taken and pending *Action Requests* for the document. The Route Log can be considered an audit trail for a document.
Route Module	A routing component that the engine uses to generate action requests at a particular *Route Node*. *FlexRM* (Flexible Route Module) is a general Route Module that is rule-based. Clients can define their own Route Modules that can conduct specialized Routing based on routing tables or any other desired implementation.
Route Node	Represents a step in the routing process of a document type. Route node "instances" are created dynamically as a document goes through its routing process and can be defined to perform any function. The most common functions are to generate Action Requests or to split or join the route path.

- Simple: do some arbitrary work

- Requests: generate action requests using a Route Module or the Rules engine

- Split: split the route path into one or more parallel branches

- Join: join one or more branches back together

- Sub Process: execute another route path inline

- Dynamic: generate a dynamic route path

Route Path	The path a document follows during the routing process. Consists of a set of route nodes and branches. The route path is defined as part of the *document type* definition.
Route Status	The status of a document in the course of its routing:

- Approved: These documents have been approved by all required reviewers and are waiting additional postprocessing.

- Cancelled: These documents have been stopped. The document's initiator can 'Cancel' it before routing begins or a reviewer of the document can cancel it after routing begins. When a document is cancelled, routing stops; it is not sent to another Action List.

- Disapproved: These documents have been disapproved by at least one reviewer. Routing has stopped for these documents.

- Enroute: Routing is in progress on these documents and an action request is waiting for someone to take action.

- Exception: A routing exception has occurred on this document. Someone from the Exception Workgroup for this Document Type must take action on this document, and it has been sent to the Action List of this workgroup.

- Final: All required approvals and all acknowledgements have been received on these documents. No changes are allowed to a document that is in Final status.

- Initiated: A user or a process has created this document, but it has not yet been routed to anyone's Action List.

- Processed: These documents have been approved by all required users, and is completed on them. They may be waiting for Acknowledgements. No further action is needed on these documents.

- Saved: These documents have been saved for later work. An author (initiator) can save a document before routing begins or a reviewer can save a document before he or she takes action on it. When someone saves a document, the document goes on that person's Action List.

Routed By User

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

Routing

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

Routing Priority

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

Routing Rule

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

Technical considerations for a Routing Rules are:

- Configured via a GUI (or imported from XML)

- Created against a Rule Template and a Document Type

- The Rule Template and it's list of Rule Attributes define what fields will be collected in the Rule GUI

- Rules define the users, groups and/or roles who should receive action requests

- Available Action Request Types that Rules can route

  - Complete

  - Approve

  - Acknowledge

  - FYI

- During routing, Rule Evaluation Sets are "selected" at each node. Default is to select by Document Type and Rule Template defined on the Route Node

- Rules match (or 'fire') based on the evaluation of data on the document and data contained on the individual rule

- Examples

  - If dollar amount is greater than $10,000 then send an Approval request to Joe.

  - If department is "HR" request an Acknowledgment from the HR.Acknowledgers workgroup.

Rule Attribute

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

Technical considerations for a Rule Attribute are:

- They might be backed by a Java class to provide lookups and validations of appropriate values.

- Define how a Routing Rule evaluates document data to determine whether or not the rule data matches the document data.

- Define what data is collected on a rule.

- An attribute typically corresponds to one piece of data on a document (i.e dollar amount, department, organization, account, etc.).

- Can be written in Java or defined using XML (with matching done by XPath).

- Can have multiple GUI fields defined in a single attribute.

Rule QuickLinks

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

Rule Template

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

Technical considerations for a Rule Templates are:

- They are a composition of Rule Attributes

- Adding a 'Role' attribute to a template allows for the use of the Role on any rules created against the template

- When rule attributes are used for matching on rules, each attribute is associated with the other attributes on the template using an implicit 'and' logic attributes

- Can be used to define various other aspects to be used by the rule creation GUI such as rule data defaults (effective dates, ignore previous, available request types, etc)

S

Save	A workflow action button that allows the Initiator of a document to save their work and close the document. The document may be retrieved from the initiator's Action List for completion and routing at a later time.
Saved	A routing status indicating the document has been started but not yet completed or routed. The Save action allows the initiator of a document to save their work and close the document. The document may be retrieved from the initiator's action list for completion and routing at a later time.
Searchable Attributes	Attributes that can be defined to index certain pieces of data on a document so that it can be searched from the *Document Search screen*.

Technical considerations for a Searchable Attributes are:

- They are responsible for extracting and indexing document data for searching

- They allow for custom fields to be added to Document Search for documents of a particular type

- They are configured as an attribute of a Document Type

- They can be written in Java or defined in XML by using Xpath to facilitate matching

Secondary Delegation	The Secondary Delegate acts as a temporary backup Delegator who acts with the same authority as the primary Approver/the Delegator when the Delegator is not available. Documents appear in the Action Lists of both the Delegator and the Delegate. When either acts on the document, it disappears from both Action Lists.

Secondary Delegation is often configured for a range of dates and it must be registered in KEW or KIM to be in effect.

Service Registry	Displays a read-only view of all of the services that are exposed on the Service Bus and includes information about them (for example, IP Address, or Endpoint URL).
Simple Node	A type of node that can perform any function desired by the implementer. An example implementation of a simple node is the node that generates Action Requests from route modules.
SOA	An acronym for Service Oriented Architecture.
Special Condition Routing	This is a generic term for additional route levels that might be triggered by various attributes of a transaction. They can be based on the type of document, attributes of the accounts being used, or other attributes of the transaction. They often represent special administrative approvals that may be required.
Split Node	A node in the routing path that can split the route path into multiple branches.
Spring	The Spring Framework is an open source application framework for the Java platform.
State	Defines U.S. Postal Service codes used to identify states.
Status	On an Action List; also known as Route Status. The current location of the document in its routing path.

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

T

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

U

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

V

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

W

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

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

Workflow Engine

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

Workflow QuickLinks

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

- Quick EDoc Watch: The last five Actions taken by this user. The user can select and repeat these actions.

- Quick EDoc Search: The last five EDocs searched for by this user. The user can select one and repeat that search.

- Quick Action List: The last five document types the user took action with. The user can select one and repeat that action.

X

XML

See also *XML Ingester*.

1. An acronym for Extensible Markup Language.

2. Used for data import/export.

XML Ingester

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

XML RuleAttribute

Similar in functionality to a RuleAttribute but built using XML only