

Kuali Rice 2.4.2 KEW Guide

Table of Contents

1. KEW	1
What is Kualiti Enterprise Workflow?	1
What is workflow, in general?	1
What is Kualiti Enterprise Workflow, in particular?	1
What problems or functions does KEW solve?	2
What problems does KEW NOT solve?	2
With which applications can KEW integrate?	3
Can I use KEW without building an entire application?	3
Kualiti Enterprise Workflow: Overview	3
What is KEW?	5
KEW Features	5
Why Use KEW?	6
2. PeopleFlow - a new KEW feature	7
Creating a new PeopleFlow	7
3. Actions	12
Action List Guide	12
Action List Preferences	15
Action List Preferences: General Section	17
Action List Preferences: Fields Displayed	18
Action List Preferences: Document Route Status Colors	18
Action List Preferences: Email Notifications	19
Action List Refresh and Filter	21
Refresh	21
Filter	21
Keys to using the Action List Filter:	22
Filter Criteria:	22
Clear Filter	24
Action Requests and Actions Guide	25
Overview	25
Types of Action Requests	25
Action Request Recipients	26
Action Request Hierarchies	26
Action Request Activation and Deactivation	27
Actions and Action Takens	28
4. Documents	34
eDocLite Overview	34
eDocLite Lookup	34
Finding the eDocLite Lookup Screen	35
eDocLite Lookup	35
eDocLite Inquiry	37
Create New eDocLite Document	37
Document Header	37
Document body	38
Routing Action and Annotation, and Note area	38
Extendable functions	39
5. Preferences	40
General Preferences	41
Fields Displayed in Action List Preferences	42
Document Route Status Colors for Action list Entries	42
6. Routing	43
Delegate Rules and Delegation Routing Rules	43

Delegation Rule Inquiry	45
Delegate Routing Rule Creation	45
Routing Rule Delegation Maintenance	46
Document Layout	46
Delegation Details Tab	47
Delegate Rule tab	48
Persons tab	49
Groups tab	50
Rule Lookup	51
Rule Inquiry	53
Routing Rule Creation	53
Routing Report	54
Using the Report	54
Select a Rule Template	54
Enter Routing Data	54
Report Contents	55
Rule Maintenance	55
Document Layout	56
Rule tab	56
Persons tab	57
Groups tab	58
Rule Template Lookup	59
Rule Template Inquiry	60
7. KEW Document Operations	61
Document Operation	61
Document Actions	62
Document	63
Action Requests	64
Actions Taken	65
Action Items	66
Route Node Instances	67
Branch States	67
Annotation	68
Practical Uses of Document Operation	68
8. Steps to Building a KEW Application	69
Preface	69
Initial Steps - Determine the Routing Rules	69
Configure the Process Definition	69
Client PlugIn Steps	72
Build PostProcessor and Services	74
Package PlugIn	75
Client Web Application Steps	75
Build the Web Application	75
Build the Service that Connects to the Workflow Engine	75
Build the Action Class with Workflow Lifecycle Methods	76
Package the Web Application	77
Final Steps	77
Deploy the PlugIn	77
Deploy the Client Web Application	77
9. KEW Configuration	78
KEW Integration Options	78
Standalone Server	79
Embedded Deployment Diagram	79
Bundling the KEW Application	79

web.xml	79
Bundled Deployment Diagram	82
Using the Remote Java Client	82
Using the Thin Java Client	82
Required Thin Client Configuration Properties	83
Optional Thin Client Configuration Properties	83
Thin Client Spring Configuration	83
Endpoint URLs	84
Thin Client Deployment Diagram	84
Picture of an Enterprise Deployment	85
KEW Core Parameters	86
KEW Configuration Properties	87
The 'embedded.server' Parameter	88
The 'datasource.platform' Parameter	88
Custom Servlet Filters	88
Email Configuration	90
Periodic Email Reminders	90
Email Customization	91
Configure a CustomEmailAttribute	91
Create a Custom XSLT Style Sheet	92
Workflow Preferences Configuration	94
Outbox Configuration	95
Implementing KEW at your institution	96
Bootstrap data	96
KEW Routing Components and Configuration Guide	97
Routing Configuration using KIM Responsibilities	108
10. KEW Administration Guide	110
Configuration Overview	110
Application Constants	110
Production Environments	110
XML Ingestion	111
Uploading an eDocLite form	111
Message Queue Administration	112
Examining the Message Queue	112
Diagnosing and Fixing Problems	113
11. KEW System Parameters	114
System Parameters Covered	114
12. Defining Workflow Processes	118
Defining Workflow Processes Using Document Types	118
Common Fields in Document Type XML Definition	118
Document Types	120
Document Type Authorizer	151
Document Type Policies	152
Inheritance	157
Defining Workflow Processes Using PeopleFlow - a new feature in KEW	158
Technical Information about PeopleFlow	158
13. Using the Workflow Document API	159
Overview	159
WorkflowDocument	159
WorkflowInfo	159
14. Creating an eDocLite Application	160
Overview	160
Components	161
Field Definitions	161

XSLT Style Sheet	163
Lazy importing of EDL Styles	166
Document Type	167
Parent DocType	168
Child DocType	169
Rule Attributes	170
Rule Routing	170
Ingestion Order	171
15. Customizing Document Search	172
Custom Search Screen	172
What are custom document search attributes?	172
Hide Search Fields and Result Columns	173
Hide a result column	173
Hide a search field	173
Field and column visibility based on workgroup membership	173
Configure visibility for both field and column	173
No field visibility	174
Configure Lookup Function	174
Application Document Status	175
Define Keyword Search	176
Custom Search Criteria Processing	177
URL Parameter Options	177
Using a Custom Search Criteria Processor	179
Custom Search Generation	181
Implementing a Custom Result Set Limit	181
Custom Search Results	181
Custom XML Document Search Result Processor Attribute	182
Custom Document Search Result Processor Class File	183
Differences between SearchableAttribute and RuleAttribute	184
Document Security	184
Overview	184
Security Definition	184
<initiator>	185
<routeLogAuthenticated>	185
<securityAttribute>	185
<searchableAttribute>	186
<group>	186
<role>	186
Order of Evaluation	187
Security - Warning Messages	187
Document Search	187
Route Log and Doc Handler	187
Service Layer	188
16. Document Link	189
Document Link Features	189
Document Link API	189
Document Link API Example	189
17. Reporting Guide	191
Reporting Features	191
The Routing Report Screen	191
The Report APIs	191
Report Criteria	191
Routing Report against a Document	191
Terminate Report at Node or User	192

Report against a Document Type	192
Interpreting Report Results	192
18. Workflow Plugin Guide	194
Overview	194
Application Plugin	194
Plugin Layout	194
Plugin Configuration	195
Plugin XML Configuration	195
Plugin Parameters	195
Transaction and DataSource Configuration	196
OBJ Configuration within a Plugin	197
Overriding Services with a ResourceLoader	198
Commonly Overridden Services	199
Plugin Shared Space	200
19. KEW Usage of the Kuali Service Bus	201
General Usage	201
Implications of using "Synchronous" KSB messaging with KEW	201
Glossary	202

List of Figures

1.1. KEW and KEN Process Flow	4
2.1. PeopleFlow selection on the Main tab in the Kuali Portal	7
2.2. PeopleFlow Lookup	8
2.3. PeopleFlow - Create New	9
2.4. PeopleFlow - Create New (with additional type attributes)	10
2.5. PeopleFlow - Create New - with 2 stops added	11
3.1. Kuali Portal: Action List Button	12
3.2. Action List	14
3.3. Route Log	15
3.4. Action List: Preferences Button	16
3.5. Action List Preference Page	17
3.6. Action List Preferences: General section	17
3.7. Action List Preferences: Fields Displayed section	18
3.8. Action List Preferences: Document Route Status Colors Section	18
3.9. Action List Preferences: Document Route Status Colors Example	19
3.10. Action List Preferences: Document Route Status Colors Action List Example	19
3.11. Action List Preferences: Email Notifications Preferences	19
3.12. Action List Preferences: Document Type Notifications	20
3.13. Action List Preferences: Action Requested Email Notification	21
3.14. Action List Filter Page	21
3.15. Document Type Lookup	23
3.16. Date Widget	24
3.17. Action List: Clear Filter Button	24
3.18. Delegation Tree Example	27
3.19. Delegation Tree Example: Deactivation	28
3.20. New Superuser Tab Example	33
4.1. Workflow Channel: eDocLite Link	35
4.2. eDocLite Lookup	35
4.3. eDocLite Lookup: Search Results	36
4.4. eDocLite Inquiry	37
5.1. Workflow Channel: User Preferences Link	40
5.2. Workflow Preferences	41
6.1. Workflow Channel: Routing Rules Delegation Link	43
6.2. Delegation Lookup	43
6.3. Delegation Lookup: Results Example	44
6.4. Delegation Inquiry	45
6.5. Delegation Rule: Create New Screen	45
6.6. Delegate Routing Rule: Create New, Parent Selection	46
6.7. Routing Rule Delegation: Overview	46
6.8. Routing Rule Delegation: Details Section	47
6.9. Routing Rules Delegation: Edit/Copy View	47
6.10. Routing Rules Delegation: Delegate Rule Tab	48
6.11. Routing Rule Delegation: Delegate Rule Tab, Edit/Copy View	48
6.12. Routing Rules Delegation: Persons Tab	49
6.13. Routing Rules Delegation: Persons tab, Copy/Edit View	50
6.14. Routing Rules Delegation: Groups Tab	50
6.15. Routing Rules Delegation: Groups Tab, Edit/Copy View	51
6.16. Workflow Channel: Routing Rules Link	51
6.17. Routing Rules Lookup	52
6.18. Routing Rules Lookup: Results Example	52
6.19. Routing Rules Inquiry	53

6.20. Routing Rules Creation	53
6.21. Workflow Channel: Routing report	54
6.22. Routing Report: Template Selection	54
6.23. Routing Report: Template Selection, Detail	54
6.24. Routing Report: Routing Data Entry	55
6.25. Routing Report: Route Log View	55
6.26. Routing Report: Route Log View, Pending Action Requests	55
6.27. Rule Maintenance Document Type Document	56
6.28. Rule Maintenance Document Type Document: Rule Tab	57
6.29. Rule Maintenance Document Type Document: Rule Tab, Edit/Copy View	57
6.30. Rule Maintenance Document Type Document: Person Tab	58
6.31. Rule Maintenance Document Type Document: Person Tab, Edit/Copy View	58
6.32. Rule Maintenance Document Type Document: Groups Tab	58
6.33. Rule Maintenance Document Type Document: Groups Tab, Edit/Copy View	59
6.34. Rule Template Lookup	59
6.35. Rule Template Lookup: Results Example	60
6.36. Rule Template Inquiry	60
7.1. Initial Screen	61
7.2. Document Actions	62
7.3. Document	63
7.4. Action Requests	64
7.5. Actions Taken	65
7.6. Action Items	66
7.7. Route Node Instances	67
7.8. Branch States	67
7.9. Annotation	68
9.1. Embedded Deployment Diagram example	79
9.2. Bundled deployment diagram	82
9.3. Thin client deployment diagram	85
9.4. Typical enterprise deployment of Kuali Rice	86
9.5. Parallel and Sequential Activation Types	101
9.6. Parallel-Priority Activation Type	101
10.1. Ingestor	111
10.2. Ingestion Complete	111
10.3. Message Queue Screen	112
10.4. Route Queue Entry Edit Screen	113
12.1. BlanketApproveSequentialTest Workflow	123
12.2. BlanketApproveParallelTest Workflow	126
12.3. NotificationTest Workflow	129
12.4. Blanket Approve Mandatory Test	132
12.5. Save Action Event Test	134
12.6. Save Action Even Test: Non-Initiator	136
12.7. Take Workgroup Authority	138
12.8. Move Sequential Test	140
12.9. Move In Process Test	143
12.10. Adhoc Route Test	145
12.11. PreApproval Test	146
12.12. Variables Test	150
12.13. Super User Action on Requests	155
14.1. EDL Controller Chain	161
15.1. Custom Search Screen:Offer Request Example	172
15.2. Custom Document Search: Department Example	175
15.3. Document Search Screen: Application Document Status Example	176
15.4. Standard Doc Search Results Set	181

List of Tables

3.1. Types of Action Requests	25
4.1. eDocLite Lookup Attributes	35
4.2. Document Header Attributes	37
4.3. Document Body Attributes	38
4.4. Routing Action and Annotation, and Note Attributes	38
5.1. User Preferences Attributes	41
5.2. User Preferences: Fields Displayed Attributes	42
6.1. Routing Rules Delegation Attributes	48
6.2. Routing Rule Delegation: Delegate Rule Tab Attributes	49
6.3. Routing Rules Delegation: Persons Tab Attributes	50
6.4. Routing Rules Delegation: Groups Tab Attributes	51
6.5. Routing Rule Creation Attributes	53
6.6. Rule Maintenance Document Type Document: Rule Tab Attributes	57
6.7. Rule Maintenance Document Type Document: Person Tab Attributes	58
6.8. Rule Maintenance Document Type Document: Groups Tab Attributes	59
6.9. Rule Template Lookup Attributes	60
9.1. Advantages/Disadvantages of KEW Integration Options	78
9.2. Required Thin Client Configuration Properties	83
9.3. Optional Thin Client Configuration Properties	83
9.4. KEW Core Parameters	86
9.5. KEW Configuration Properties	87
9.6. Optional Properties to Configure Simple SMTP Authentication	90
9.7. Configuration Parameters for Email Reminders	90
9.8. InitiatorRoleAttribute	104
9.9. RoutedByUserRoleAttribute	104
9.10. NoOpNode	104
9.11. RequestActivationNode	105
9.12. NetworkIdRoleAttribute	105
9.13. UniversityIdRoleAttribute	108
9.14. SetVarNode	108
11.1. KEW System Parameters	114
12.1. Common Fields in Document Type XML Definition	118
15.1. Key Reference Table: Default field names and reference keys	183
18.1. Commonly Overridden Services	199

Chapter 1. KEW

What is Kual Enterprise Workflow?

What is workflow, in general?

Workflow is a very general term and means different things in different contexts. For example, it may mean the sequence of approvals needed for a Leave Request or it may refer to a complex scientific procedure.

In our context of enterprise applications within a higher education institution, we're usually talking about business process management when we discuss workflow. Usually, this revolves around business rules, authorizations, and routing for approval.

A simple example is a leave request system. It needs some workflow to get the necessary people (supervisor, etc.) to approve it. This is one example of the routing and approval side of a workflow.

You may also have business rules in workflow that dictate that some people get automatic approval for leave requests. This is a business rule detail that workflow executes by automatically routing these types of requests past the approval steps.

What is Kual Enterprise Workflow, in particular?

The Kual Enterprise Workflow (KEW) product revolves around routing and approval of documents. It is a stand-alone *workflow engine* upon which you can integrate other enterprise applications to do routing and approvals.

In addition, KEW contains an **eDocLite** system. This is a mechanism to create simple data-entry forms directly in KEW. You can also create routing rules around eDocLite forms. eDocLite forms are the rough equivalent of the basic, one- or two-page forms that are commonly used to process business and get signature approvals.

The benefit of eDocLite in KEW is that it does *not* require a separate application. You can use eDocLite in KEW simply by setting up the forms that your institution or department needs.

Overall, KEW is based on documents. In KEW, each document has a collection of transactions or *things to be done*. Each transaction is approved or denied individually in KEW.

For example, John Doe may use a *Leave Request* document in KEW to ask for a week off in June. The KEW Leave Request document contains enough information for his supervisor to make a decision about John's leave. (The document may use data keys to retrieve external information, such as John's past Leave Requests and available hours.) Once John submits his Leave Request, KEW routes it to John's supervisor for approval. Depending on how John's department has configured KEW for routing Leave Requests, after John's supervisor approves or denies his request, KEW may route it to more people for further action to be taken.

Once John's Leave Request document is processed, it triggers a **PostProcessor**, which can perform any desired additional processing. This is most commonly used to "finalize" the business transaction once all approvers have signed off on it. In this particular example, it might call another service that would update records in the Leave Request application's database, indicating that the individual has successfully scheduled leave during that time period.

In addition, the KEW **PostProcessor** contains hooks for all the stages that a document goes through. For example, an external application may use a KEW workflow for routing and approval of documents, and that application may take action at each change in state of a routed document.

What problems or functions does KEW solve?

The primary benefit of KEW workflow is the correct routing for approval of documents. It enforces your business-specific rules about who needs to approve what documents, in which scenarios.

Simple Workflow Example

Leave Request: Each person has one other person (possibly more) who needs to approve his or her leave requests. In this context, KEW is the system that manages both the approval structure and the leave requests themselves (the actual approvals).

More Complex Workflow Example

Purchasing Desktop Computers: You may need several business rules in KEW for this, such as a rule to enforce:

1. A strategic alliance requires that you buy from one vendor unless there is a justification to not do so
2. General purchasing approval by the Purchasing Department is required when the cost of the purchase exceeds a certain limit
3. Approval by the account owners who fund the purchase is required

In this example, KEW requires an approval if:

- The strategic alliance is not used
- The cost limit for Purchasing Department approval is exceeded

The workflow also requires an approval by the signer (or delegate) for each spending account that you use for the purchase.

In KEW, **Approval Types** are set up such as account approver, supervisor, or organizational/department hierarchy approver. An Approval Type contains the applicable routing and approval rules. Once you create an approval type, those routing and approval rules are available for other workflow clients and scenarios. This creates a *tipping point* situation, in which the more applications and business processes you set up through workflow, the easier it gets to do new ones.

In addition, KEW can help you with distributed management of approval structures. Each group at your institution (each college, unit, division, etc.) can create their own approval and workflow structure for their group, and you can centrally manage the workflow above those groups. This allows groups to manage their own internal controls and structures, while still being subject to higher-level institutional controls.

What problems does KEW NOT solve?

KEW is not a general-purpose application builder. For complex applications, you need to develop applications separately and then integrate them with KEW. For simple forms or documents that need approval, you can use **eDocLite**, but this only works in simple cases, analogous to a one- or two-page paper form that requires signatures. It is important to note, however, that Quali Rice does include a framework

called the Kualu Nervous System (KNS) that can be used to facilitate the development of more complex applications and includes built-in integration with KEW.

KEW is not a general-purpose business rules engine. For example, it does not know that a continuation account must be specified when an account is closed. Those types of rules are the responsibility of the application itself to manage. However, this is not a clear-cut line, as KEW does manage business rules that directly affect routing and approval.

KEW is not an Organization Hierarchy manager. For example, it will not automatically manage your organizational hierarchies and internal structures. However, integration with these hierarchies and structures can be accomplished using KEW, and leveraging such hierarchies for routing and approval is a very common need for many applications.

With which applications can KEW integrate?

Nearly anything, in theory. In the current version of KEW, any application can access KEW if it can:

- Do Java method calls, or
- Do remote method invocation, or
- Do web-services calls, or
- Communicate with the Kualu Service Bus (KSB)

(The recommended cross-platform integration method is over web services.)

Can I use KEW without building an entire application?

Yes, absolutely!

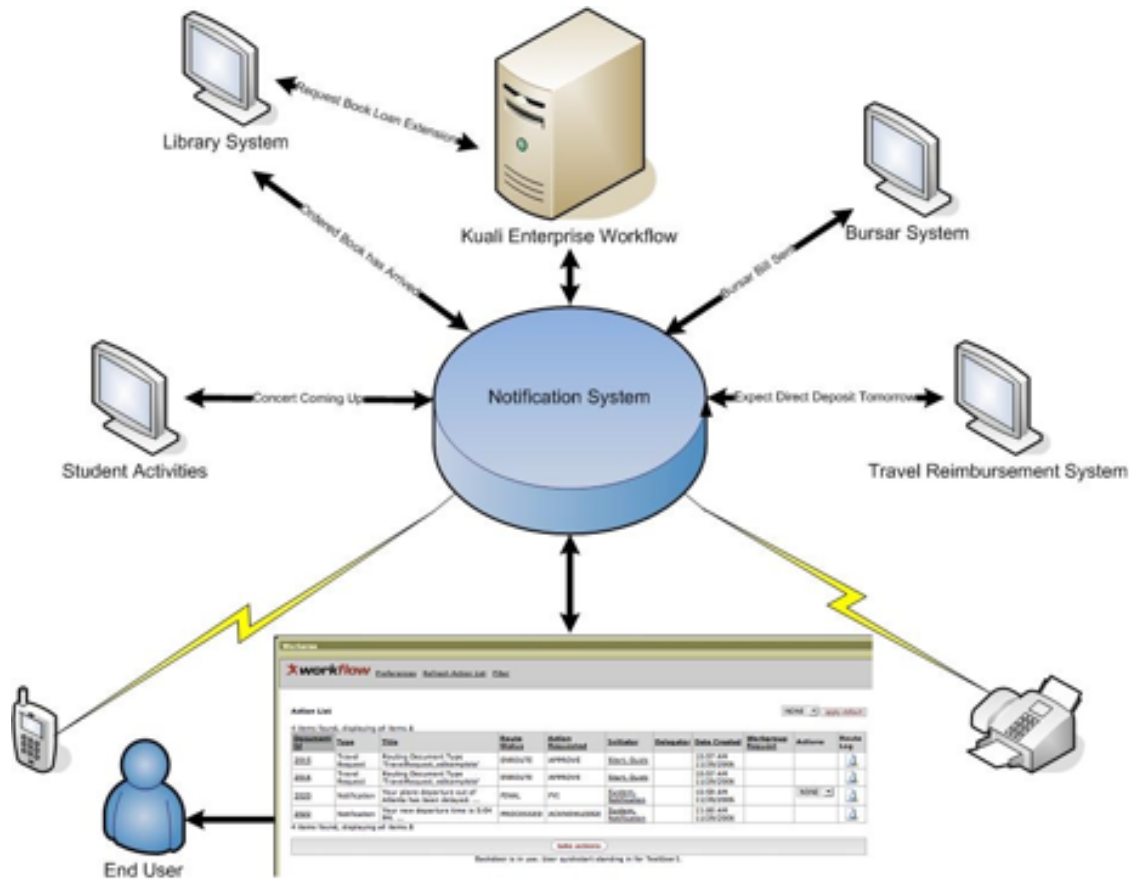
KEW is an incredibly powerful platform for routing and approval for enterprise (i.e., large) applications. However, it also includes **eDocLite**, which makes it easy to develop simple business-process forms and run them through KEW. In this situation, in its most simple form, you can do all of your work within KEW, and most of that work is in developing your form configurations. If needed, the eDocLite process can also hook into a post-processor to take an action once a document's approvals are complete.

Kualu Enterprise Workflow: Overview

Kualu Enterprise Workflow, or KEW, is a module of Kualu Rice. KEW provides useful features for automatically sending documents to people for approval or action using workflows, managing the documents you receive through these workflows, searching for documents, and checking information about documents that have been in a workflow. KEW is built by and for higher education institutions and is especially designed for automatically routing work across departmental boundaries.

The following architectural diagram represents the flow of messages in a typical Rice Environment, and illustrates the integration between KEW and the post-processor lifecycle.

Figure 1.1. KEW and KEN Process Flow



For more information, please see the [KEN Guide](#).

Using KEW you can:

- Automatically send (route) documents to individuals, groups, or roles (people who do a particular function) using workflows
- Create rules to automatically route documents based on the content of the document
- Change routing rules, and the documents affected by your changes are rerouted immediately
- Attach notes and other files to documents in workflows
- Build forms easily and use them in workflows
- Search for documents based on workflow information and some content in the document
- Check the history of documents and the people who took part in the workflow for each document
- Delegate items in a workflow to another person
- Create and update groups for workflows
- Customize the email messages that are sent when someone needs to take an action for a workflow
- Choose whether or not you receive an email message when you have a new action item from a workflow

- Integrate KEW with the Kualu Service Bus (KSB) for routing documents

Your institution may use some or all of these KEW features. Your institution may also use some of these KEW features and combine them with other non-Kuali applications, such as an application that manages users and/or groups, or an application that manages notes or attachments.

Information Sources: <http://kew.kuali.org/>

What is KEW?

Kuali Enterprise Workflow provides a common routing and approval engine that facilitates the automation of electronic processes across the enterprise. The workflow product was built by and for higher education, so it is particularly well suited to route mediated transactions across departmental boundaries. Workflow facilitates distribution of processes out into the organizations to eliminate paper processes and shadow feeder systems. In addition to facilitating routing and approval, workflow can also automate process to process related flows. Each process instance is assigned a unique identifier that is global across the organization. Workflow keeps a permanent record of all processes and their participants.

KEW Features

- **Flexible Workflow Engine** - Support for sequential, parallel and dynamic routing paths. Extensible architecture allows for easy customization.
- **Content-Based Routing** - Routing decisions can be made based on XML document content. XPath and other XML tools can be used to determine routing without writing code.
- **Pluggable Components** - Components can be deployed to the system at runtime using Plugins. Hot deployable class loading space provides a robust enterprise ready deployment environment for workflow code.
- **People in the Routing Process** - Documents can be routed to individuals, groups or roles.
- **Action List** - Displays a list of each user's pending items which require his/her attention, such as documents which are awaiting approval. Users can configure whether they receive emails when the document enters their Action List.
- **Document Search and Route Log** - Allows users to search for documents and see an audit trail of what has happened to the document during its life cycle.
- **Document Search Customization** - Document based content can be associated with workflow data and searched on using our Document Search screens. Have a single place for all of your workflow document searches.
- **EDocLite** - EDocLite allows quick document building and integration with workflow using only XML.
- **Rules System** - Provides a mechanism for defining business rules which govern how a document routes. Rule screens give functional users ability to maintain the business rules for their applications. Documents affected by rule changes are re-routed real time.
- **Notes and Attachments** - Notes and Attachments can be attached to documents using KEW's notes and attachments services out of the box. Institution based attachment and note services can be used by overriding our default services.
- **Person Services** - Maintains users of the system. You can use the Out-of-the-Box service or override with your institution's user services.

- **Group Services** - Maintains groups of users. You can use the Out-of-the-Box service or override with your institution's group services.
- **Transactions** - All transactions and documents route in a JTA transaction.
- **Web Service API** - All system functions are available through Web Service APIs.
- **Security** - Web service calls can be authorized using digital signatures.
- **Scalability** - Can be clustered and run on multiple machines to allow for horizontal scalability.
- **Embeddable Engine** - Workflow engine can be embedded in an application as well as ran as a standalone service.
- **Embeddable Web Application** - Web portion can be embedded in an application as a Struts module. Run the Action List, Document Search, Route Log, Rules System and more from within your application with minimal effort.
- **Service Bus Integration** - Integration with the Kuali Service Bus (KSB). Override any service from within workflow by having workflow grab the service from the bus or use workflow's pluggable components to deploy bus enabled services.
- **JMX Support** - A set of management functions are exposed through JMX.
- **Spring Based Integration** - KEW is designed with Spring based integration in mind.

Why Use KEW?

- Provides a single action list for the constituents of the organization to see work that requires their attention
- Establishes a configurable way for service providers to define their processes, allowing them to alter those processes over time to reflect organizational change
- Promotes transparency of processes to the institution so that people can seamlessly see the status, actors, and the history of any institutional process which leverages KEW as its workflow engine

Chapter 2. PeopleFlow - a new KEW feature

PeopleFlow is our Kuali Rice instantiation of the "maps" concept included in the original Coeus. For all intents and purposes it's a prioritized list of people to send requests to. Essentially, it's like a mini people-based workflow that doesn't require you to specify a KEW node in the document type for each individual who might need to approve or be notified. You can define "Stops" in a PeopleFlow, where everything in the same stop proceeds in parallel, but all must be done within the stop before proceeding to the next stop.

You can call/execute a PeopleFlow from within a KEW workflow node directly, or you can invoke the [Kuali Rules Management System \(KRMS\)](#) engine from an application and any PeopleFlows that get selected during rule execution, defined in a KRMS agenda, will be called. In this way, you can integrate business rules across applications and workflows.

The same PeopleFlow that defines a routing order among a set of persons, groups or roles can be called by KRMS rules, with the KRMS rules defining which type of request to pass to the PeopleFlow (for example, an "approval" routing action or a "notification").

KRMS is also a new feature in Rice 2.0. See the KRMS Users' Guide for more information on KRMS.

Creating a new PeopleFlow

You can define a PeopleFlow (simple workflow) via a maintenance document. In the Kuali Rice portal on the main tab there is a Workflow category, and PeopleFlow is in that category. Select it:

Figure 2.1. PeopleFlow selection on the Main tab in the Kuali Portal



You can search for a PeopleFlow to copy from and modify (be sure to give it a new unique name) or can create a new one (see the top right of the lookup screen).

Figure 2.2. PeopleFlow Lookup



Below is a view of the user interface to create a new PeopleFlow:

Figure 2.3. PeopleFlow - Create New

The screenshot shows the 'People Flow Maintenance' page in the Kualiti Rice application. The page header includes the Kualiti logo, navigation menus (Main Menu, Administration, KRAD), and user information (Logged in User: adr). The main content area displays document details for Document Number 3017, which is in an INITIATED status. Below the details are several sections: 'Document Overview' with fields for Description, Organization Document Number, and Explanation; 'PeopleFlow Summary' with fields for Namespace Code, Name, Type, Description, and Active; and 'PeopleFlow Members' with an 'add' button and fields for Stop, Member Type Code, and Member. At the bottom, there are buttons for submit, save, blanket approve, close, and cancel.

The view below shows that some sections are collapsed, and also shows additional attributes that are required based on the user's selection of a type. This is an example of the Kualiti Rapid Application Development (KRAD) framework's support of progressive disclosure, only showing these additional type attribute fields if/when they are required:

Figure 2.4. PeopleFlow - Create New (with additional type attributes)

Home » People Flow Maintenance

People Flow Maintenance	Document Number: 3018	Document Status: INITIATED
	Initiator Network Id: admin	Creation Timestamp: 09:23 PM 12/23/2011

* indicates required field

▶ Document Overview

▼ PeopleFlow Summary

* Namespace Code:	Kuali Rules Test	* Description:	This is just an example
* Name:	Example PeopleFlow 1	Active:	<input checked="" type="checkbox"/>
* Type:	KR-SAP - Sample Type		

Travel Account Number:	<input type="text"/>	<input type="button" value="🔍"/>
Travel Fiscal Officer Id:	<input type="text"/>	<input type="button" value="🔍"/>

▶ PeopleFlow Members

▶ Notes and Attachments (0)

▶ Ad Hoc Recipients

▶ Route Log

And below is a view of just the PeopleFlow Members section - expanded - after adding two stops:

Figure 2.5. PeopleFlow - Create New - with 2 stops added

PeopleFlow Members

add

* Stop:	<input type="text" value="1"/>
* Member Type Code:	<input type="text" value="Principal"/>
* Member:	<input type="text"/>

* Stop	* Member Type Code	* Member	* All or First Action	Actions
<input type="text" value="2"/>	<input type="text" value="Role"/>	KR-SYS : Technical Administrator	<input type="text" value="FIRST"/>	<input type="button" value="delete"/>
Member Delegates				
	<input type="text" value="Principal"/>	<input type="text"/>	<input type="text" value="Primary"/>	<input type="button" value="add"/>
<input type="text" value="1"/>	<input type="text" value="Principal"/>	eric <small>Employee, Eric</small>		<input type="button" value="delete"/>
Member Delegates				
	<input type="text" value="Principal"/>	<input type="text"/>	<input type="text" value="Primary"/>	<input type="button" value="add"/>
	<input type="text" value="Group"/>	KR-WKFLW : MagazineManagers	<input type="text" value="Primary"/>	<input type="button" value="delete"/>

PeopleFlows that you create can be called by rules, with the rules determining whether the PeopleFlow will be called for an action (such as approvals) or for notifications. These PeopleFlows (similar to the Coeus concept of "maps") can be integrated with the new Kuali Rules Management System (KRMS), with KEW workflows, or with other custom-built applications.

Chapter 3. Actions

Action List Guide

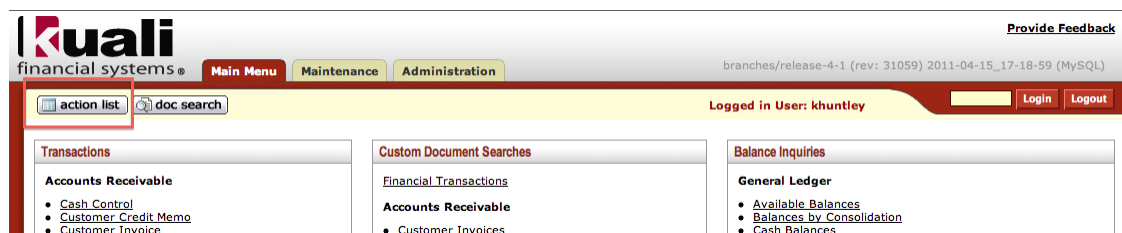
The **Action List** page includes your:

- Action List
- Outbox
- Action List Filter
- Preferences

In the Kualu Rice portal, the Action List button is available in the upper left hand corner of the screen on most pages.

Your institution may have KEW links inside other applications, such as Kualu Financial Systems (KFS). KFS may display an Action List button at the top left of the KFS page like this:

Figure 3.1. Kualu Portal: Action List Button



Clicking an Action List link or button displays your **Action List** page. This is a sample KEW Action List; your institution's may look somewhat different:

On your Action List page is the list of documents on which you need to take an action, such as Approve, Disapprove, or Acknowledge, and those which you have already completed. These are your action requests. Action requests that someone has delegated to you are also listed on your Action List page. Note: You can only see the Action List for the person who is logged into KEW or another Kualu application.

You can click the title at the top of a column to sort all of your action requests by the information in that column. For example, if you click the column title, Type, KEW sorts your action requests by their Type.

On your Action List, you can see this information about the documents on which you need to take action:

- **Id:** A unique, system-generated ID for each document
 - When you click a Document Id, KEW displays the actions you can take on this document, such as Blanket Approve, Complete, Approve, Disapprove, and Cancel. The actions you can take are determined by the document's Type.
 - Based on your institution's setup, you may see a Show button in the Document Id column. Click the Show button to see a summary of the document.
- **Type:** The document's type determines the actions you can take on it.

- **Title:** The title that the initiator gives to the document. Your institution may have standards for document titles that give you specific information about that document.
- **Status:** The current location of the document in its routing path. The Status may be:
 - **Approved:** These documents have been approved by all required reviewers and are waiting for additional post-processing.
 - **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 with the appropriate KIM Permission (or someone from the Exception Group for this Document Type, although this feature is deprecated) must take action on this document, and it has been sent to the Action List of this workgroup.
 - **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 their post-processing is completed. 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.
 - **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.
 - **Recalled:** The document has been recalled after being submitted and canceled. No changes are allowed to a document that is in Recalled status.
- **Action Requested:** The action that you have been asked to take on a document, such as to Approve it.
- **Initiator:** The person who created this document in KEW. Click the initiator's name to see information about the initiator, including (depending on the initiator's privacy preferences):
 - Principal Id and Name
 - Entity Id
 - Affiliations
 - Names
 - Phone Numbers
 - Email Addresses

- Group and Role memberships
- **Delegator:** If someone else has given authority in KEW for you to take action on their action requests, their name appears in this column. KEW displays both primary and secondary delegates this way on your Action List. There are two types of delegate in KEW, you may be a Primary Delegate or a Secondary Delegate:
 - **Primary Delegate:** The delegator turns over his or her full authority to a primary delegate. The Action Requests for the delegator only appear in the Action List of the primary delegate.
 - **Secondary Delegate:** The secondary delegate acts as a temporary backup and has the same authority as the delegator when the delegator is not available to take action. 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. People often have a secondary delegate who takes care of their action requests when they are out of the office.
- **Date Created:** The date and time the document for this action request was created.
- **Group Request:** When you receive an action request because you are part of a group, KEW displays the name of the group in this column. Only one member of a group is required to complete an action request. Click the group name on your Action List to see more details, including:
 - Group Id
 - Group Name
 - Group Active Indicator
 - Group Members and their user names
- **Actions:** For documents that you receive only for your information (they don't require that you take any further action), there is a dropdown menu in the Actions column. You can use this menu to agree that you've received the document for information purposes: Select FYI, then click the Take Actions button at the bottom of the page.

Figure 3.2. Action List

The screenshot shows the 'Action List' interface. At the top, there are navigation links: 'preferences', 'refresh', 'filter', and 'help desk action list login'. Below these is a dropdown menu set to 'NONE' and an 'apply default' button. The main content area displays a table with 16 items. The table has columns for Id, Type, Title, Route Status, Action Requested, Initiator, Delegator, Date Created, Group Request, Actions, and Log. The 'Actions' column for the selected row (Id 3822) is open, showing a dropdown menu with 'NONE' selected and 'FYI' as an option. Below the table, there is a 'take actions' button.

Id	Type	Title	Route Status	Action Requested	Initiator	Delegator	Date Created	Group Request	Actions	Log
3402	Requisition	Requisition -	SAVED	COMPLETE	HUNTLEY, KEISHA Y		04:57 PM 07/11/2011			
3403	Requisition	Requisition -	SAVED	COMPLETE	HUNTLEY, KEISHA Y		04:57 PM 07/11/2011			
3404	Requisition	Requisition -	SAVED	COMPLETE	HUNTLEY, KEISHA Y		04:57 PM 07/11/2011			
3405	Requisition	Requisition - PC req - DV - 2	ENROUTE	APPROVE	HUNTLEY, KEISHA Y		05:01 PM 07/11/2011			
3461	Disbursement Voucher Wire Charge	New WireCharge - Phil	EXCEPTION	APPROVE	HUNTLEY, KEISHA Y		10:49 PM 07/11/2011			
3809	Vendor	New VendorDetail - New Vendor	FINAL	FYI	HUNTLEY, KEISHA Y		07:26 PM 11/28/2011		NONE	
3819	Disbursement Voucher	Disbursement Voucher - test check amount	SAVED	COMPLETE	HUNTLEY, KEISHA Y		08:29 PM 11/28/2011			
3822	Vendor	New VendorDetail - New Vendor	FINAL	FYI	HUNTLEY, KEISHA Y		08:53 PM 11/28/2011		<input checked="" type="checkbox"/> NONE <input type="checkbox"/> FYI	
3842	Requisition	Requisition - test of equipment requisition	SAVED	COMPLETE	HUNTLEY, KEISHA Y		09:48 PM 11/28/2011			
3961	Disbursement Voucher	Disbursement Voucher - test delegation	SAVED	COMPLETE	HUNTLEY, KEISHA Y		02:58 PM 11/29/2011			

- **Log:** (also known as the Route Log) Click the paper/magnifying glass icon in the Log column to display the Route Log page. This page has details about the routing path and Actions Taken on this document. Learn more about this information in the Route Log Guide. A sample Route Log page:

Figure 3.3. Route Log

Route Log
refresh

ID: 3064 hide

Title			
Type	Vacancy Notice	Created	08:35 PM 12/02/2011
Initiator	admin_admin	Last Modified	08:40 PM 12/02/2011
Route Status	PROCESSED	Last Approved	
Node(s)	InitiatorAcknowledgement	Finalized	

Actions Taken hide

	Action	Taken By	For Delegator	Time/Date	Annotation
	COMPLETED	admin_admin		08:36 PM 12/02/2011	
show	APPROVED	supervisor_supervisor		08:38 PM 12/02/2011	
show	APPROVED	director_director		08:39 PM 12/02/2011	
show	APPROVED	Two_Admin		08:39 PM 12/02/2011	
show	APPROVED	One_Admin		08:40 PM 12/02/2011	

Pending Action Requests hide

	Action	Requested Of	Time/Date	Annotation
show	IN ACTION LIST ACKNOWLEDGE	OAA_BI-BUS (Secondary Delegate)	08:40 PM 12/02/2011	
show	IN ACTION LIST ACKNOWLEDGE	OAA_BI-COAS (Secondary Delegate)	08:40 PM 12/02/2011	
show	IN ACTION LIST ACKNOWLEDGE	OAA_BI-DFAC (Secondary Delegate)	08:40 PM 12/02/2011	
show	IN ACTION LIST ACKNOWLEDGE	OAA_BI-PSY (Secondary Delegate)	08:40 PM 12/02/2011	
show	IN ACTION LIST ACKNOWLEDGE	admin_admin (Secondary Delegate) (Initiator)	08:40 PM 12/02/2011	

If enabled for a particular document, messages can also be logged through the route log tab (or page). This allows users to add a deferral message or other message that will be reflected in the routing without taking an action on the document.

This function is enabled by KIM permissions. These permissions have template 'Add Message to Route Log' which takes a document type name as a detail. The bootstrap dataset contains one permission with this template and document type detail 'KualiDocument', however this permission is not granted to any role. This means the functionality will be disabled until the permission is assigned or another permission record with this template and for another document type is created and assigned. When matching permissions for a particular document type the document type hierarchy is considered. Permission records for children document types will override any permission record for its parents.

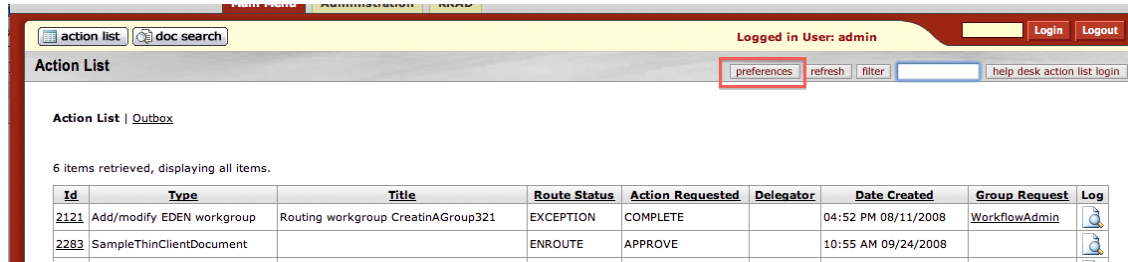
For example, if we wanted to enable the feature on all KualiDocuments except document type 'Foo', we would first grant the permission with document type detail 'KualiDocument' to a role. Then we would create an additional permission with template 'Add Message to Route Log' and document type detail 'Foo'. For the second permission we do not assign any roles.

By granting these permission(s) to one of the derived workflow roles we can enable this feature to users who have active routing requests. Examples of such roles are KR-WKFLW Approve Request Recipient and KR-WKFLW Acknowledge Request Recipient.

Action List Preferences

You can customize the appearance and features of your Action List using the Action List Preferences page on KEW. To display this page, click the Preferences button at the top of the Action List page or click the User Preferences link in the Workflow pane of the Main Menu tab. Action List page:

Figure 3.4. Action List: Preferences Button



This displays the Action List Preferences page, where you can make choices about how KEW displays and refreshes (updates) your Action List and about how and when you receive email reminders about your new action requests:

Figure 3.5. Action List Preference Page

Workflow Preferences

General

Automatic Refresh Rate: 15 in whole minutes - 0 is no automatic refresh.

Action List Page Size 10

Delegator Filter Secondary Delegates on Action List Page ▾

Primary Delegate Filter Primary Delegates on Action List Page ▾

Fields Displayed In Action List

Document Type

Title

ActionRequested

Initiator

Delegator

Date Created

Date Approved

Current Route Node(s)

WorkGroup Request

Document Route Status

Clear FYI

Use Outbox

Document Route Status Colors for Actionlist Entries

Saved	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Initiated	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Disapproved	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Enroute	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Approved	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Final	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Processed	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Exception	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Canceled	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Email Notification Preferences

Receive Primary Delegate Emails

Receive Secondary Delegate Emails

Default Email Notification Immediate ▾

Document Type	Notification Preference	Actions
DocumentTypeDocument	Immediate	<input type="button" value="delete"/>
<input type="text" value=""/>	None ▾	<input type="button" value="add"/>

Send Email Notifications For

Complete

Approve

Acknowledge

FYI

The Action List Preferences page has four sections:

Action List Preferences: General Section

Figure 3.6. Action List Preferences: General section

General

Automatic Refresh Rate: 15 in whole minutes - 0 is no automatic refresh.

Action List Page Size 10

Delegator Filter Secondary Delegates on Action List Page ▾

Primary Delegate Filter Primary Delegates on Action List Page ▾

Fields Displayed In Action List

- In the General section of the Action List Preferences page, you can set the Automatic Refresh Rate for your Action List. Your action requests come in from the server and your Actions Taken are sent to the server each time KEW refreshes (updates) your Action List page. You only see changes to your Action List when KEW refreshes it.
- Depending on the network and computers at your workplace, your computer may or may not slow down when KEW refreshes your Action List. If a refresh slows down your computer, you may decide to set the Automatic Refresh Rate for a longer period, such as 10 or 15 minutes.
- The Action List Page Size tells KEW how many action requests to display at once on your Action List page.
- The Delegator Filter field lets you decide where to show action requests for which you are a secondary delegate. You can choose to have KEW display these action requests on your Action List page or not, using this dropdown menu. If you do not display action requests on which you are the Secondary Delegate on your Action List page, you can only see them on a Filter Page (see Filter, below).

Action List Preferences: Fields Displayed

Figure 3.7. Action List Preferences: Fields Displayed section

Fields Displayed In Action List	
Document Type	<input checked="" type="checkbox"/>
Title	<input checked="" type="checkbox"/>
ActionRequested	<input checked="" type="checkbox"/>
Initiator	<input type="checkbox"/>
Delegator	<input checked="" type="checkbox"/>
Date Created	<input checked="" type="checkbox"/>
Date Approved	<input type="checkbox"/>
Current Route Node(s)	<input type="checkbox"/>
WorkGroup Request	<input checked="" type="checkbox"/>
Document Route Status	<input checked="" type="checkbox"/>
Clear FYI	<input checked="" type="checkbox"/>
Use Outbox	<input checked="" type="checkbox"/>

- You determine which fields are displayed on your Action List page using this section of the Action List Preferences page. Place a checkmark in the box next to fields that you want to display on your Action List page. (Click the box to checkmark it and click it again to un-checkmark it, and vice versa.)

Action List Preferences: Document Route Status Colors

Figure 3.8. Action List Preferences: Document Route Status Colors Section

Document Route Status Colors for Actionlist Entries	
Saved	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
Initiated	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
Disapproved	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
Enroute	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
Approved	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
Final	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
Processed	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
Exception	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
Canceled	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>

- In this section of the Action List Preferences page, you can choose a color for each different Status of action requests. This can help you quickly find action requests on which you need to take action.

For example, if you set these colors for these routing status levels:

- Green for Saved status
- Red for Approved status
- Blue for Exception status

Your Action List Preferences page looks like this:

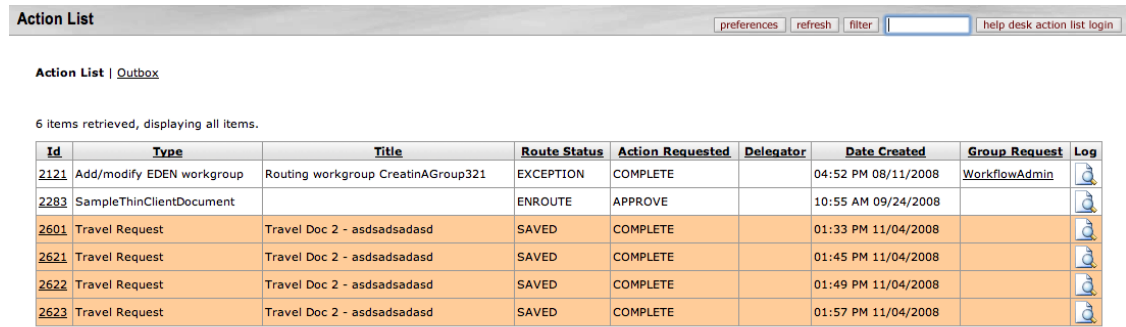
Figure 3.9. Action List Preferences: Document Route Status Colors Example



Click the Save button at the bottom of the page to make these changes in your Action List.

Action Lists your color changes by Status:

Figure 3.10. Action List Preferences: Document Route Status Colors Action List Example



Action List Preferences: Email Notifications

Figure 3.11. Action List Preferences: Email Notifications Preferences



The fields used to maintain email preferences are located in a new "Email Notification Preferences" section at the bottom of the Workflow Preferences screen. The "Email Notification Preferences" section accommodates two new options and allows selection of how and when to receive reminder emails on items sent to the Action List.

In this section, one can select whether or not to receive emails as the Primary Delegate, as the Secondary Delegate, and also set the Default Email Notification preference (None, Daily, Weekly, or Immediate). If you are NOT a Primary or Secondary Delegate, you can leave those fields as unchecked.

You can choose to get email reminders on one of these Default Email Notification schedules:

- **None** (KEW won't send you any email reminders for your Action List.)
- **Daily** (If you have any new action requests, KEW sends you an email reminder once a day.)
- **Weekly** (If you have any new action requests, KEW sends you an email reminder once a week.)
- **Immediately** (KEW sends you an email reminder as soon as you receive a new action request on a document. This is the default – the setting when you start to use KEW.)

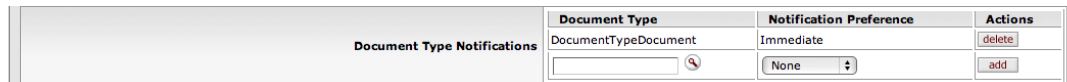
The first new option allows overriding of the 'Default Email Notification' preference and setting of different frequencies to receive notifications based on the document type of the action request (Timesheet, EPTO, Hire Employee eDoc, Create Additional Pay eDoc, Requisition, etc.).

The parameters set for the default email notification and document-type notification work together to determine when a given action item is included in the immediate, daily, or weekly email reminders.

If one sets a 'Default Email Notification' without any 'Document Type Notifications,' then emails are sent (or not sent), based on the default selection.

The second new option allows one to turn notifications on or off based on the request type (Complete, Approve, Acknowledge, or FYI) when "Immediate" is set as the 'Default Email Notification.'

Figure 3.12. Action List Preferences: Document Type Notifications



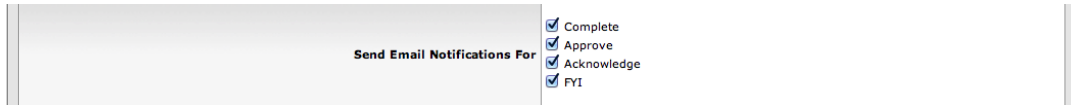
However, adding a 'Document Type Notification' that is different from the 'Default Email Notification,' forces the document-type notification to take precedence. This allows users to customize by document-type for email reminders. For example:

- If 'Immediate' is set as the 'Default Email Notification,' but a document-type rule is set to 'None' for the Maintain Time Assignment eDoc; then no email notifications are sent for that specific document.
- If 'Daily' is the 'Default Email Notification,' and a document-type rule is added to receive 'Weekly' notifications for the Hire Employee eDoc; then for that document, the only action request email received is a Weekly reminder.
- If 'None' is set as the 'Default Email Notification,' and a document-type rule is added to receive "Daily" notifications for EPIC Purchase Orders; then for that document, the only action request email received is a Daily reminder.

In addition, when the 'Default Email Notification' is set to "Immediate" one can use the Send Email Notifications For option to decide whether or not to receive notifications by request-type.

All four of the request-types are already "checked" by default in your Workflow Preferences screen.

Figure 3.13. Action List Preferences: Action Requested Email Notification



When an action request is received, and the preference is set to receive an immediate reminder, the system checks to ensure that the request-type checkbox is turned-on (checked) before an email is sent.

This allows these checkboxes to be used as a switch to 'turn-on or turn-off' the "immediate" email reminders for a given request type. For example:

- To stop the receipt of Acknowledgement emails when your preference is set to receive "Immediate" reminders, just uncheck the "Acknowledge" checkbox.

Remember, all of these preferences and settings are on your Action List Preferences page.

Action List Refresh and Filter

Refresh

As described above, you can use the Action List Preferences page to tell KEW to automatically update (refresh) the information on your Action List on a regular schedule, such as every 15 minutes. You can also click the Refresh button near the top of your Action List page to update the page at any time.

Filter

If you have a large Action List, the Filter function can help you by displaying only action requests of certain types. To use the Filter function, click the Filter button in the top right corner of your Action List page:

This displays the **Action List Filter** page:

Figure 3.14. Action List Filter Page

Action List Filter

Parameters	
Document Title	<input type="text"/> Exclude? <input type="checkbox"/>
Document Route Status	All <input type="button" value="v"/> Exclude? <input type="checkbox"/>
Action Requested	All <input type="button" value="v"/> Exclude? <input type="checkbox"/>
Action Requested Group	No Filtering <input type="button" value="v"/> Exclude? <input type="checkbox"/>
Document Type	<input type="text"/> Exclude? <input type="checkbox"/>
Date Created	from: <input type="text"/> <input type="button" value="v"/> to: <input type="text"/> <input type="button" value="v"/> Exclude? <input type="checkbox"/>
Date Last Assigned	from: <input type="text"/> <input type="button" value="v"/> to: <input type="text"/> <input type="button" value="v"/> Exclude? <input type="checkbox"/>

Keys to using the Action List Filter:

- Each of the fields on the Action List Filter page is called a criteria. You can choose to display only the action requests that are the same as one or more criteria, or you can choose to display all action requests except those that are the same as one or more criteria. You can also use some criteria to display action requests and some to not display action requests.
- If you want the filter to display action requests that meet one or more criteria, then just enter those criteria in the appropriate fields (field descriptions are below). Leave the Exclude? checkboxes blank for these criteria.
- If you want the filter to display only the action requests that don't match one or more criteria, then checkmark the Exclude? checkbox for each of these criteria. This tells KEW not to display action requests that match these criteria.
- When you've selected the criteria you want to use in your filter, click the Filter button at the bottom of the Action List Filter page. KEW displays your Action List filtered in the way you requested.
- To display all action requests on your Action List page again, click the Clear button at the bottom of the Action List Filter page or the Clear Filter button at the top of your Action List page.
- To undo your most recent filter criteria, click the Reset button at the bottom of the Action List Filter page. This returns you to the list of Action Requests that met your previous set of filter criteria.

Note

Note: The filter is case sensitive – use upper and lower case (capital letters and small letters) exactly the same as the item you're filtering. For example, if an actual document title is Travel Doc 2, then you must type this in the Document Title field with a capital letter at the beginning of each word, exactly like the real document title. If you type it as travel doc 2, then the filter won't recognize this document because the capitalization is different.

Filter Criteria:

- **Document Title:** The title given to the document when it was created.
- **Document Route Status:** The current routing Status of the document. You may filter for documents with these routing statuses:
 - **All:** Selecting All causes KEW to display all documents that have been submitted for routing. If you select All and Exclude (for the Document Route Status criteria), then KEW displays only documents that are in Initiated status.

Note

Documents that are in Initiated Status have not yet been submitted for routing.

- **Approved**
- **Disapproved**
- **Enroute**
- **Exception**
- **Processed**

- **Saved**
- **Action Requested:** The action you need to take on a document. You may filter your action requests by these Action Requests:
 - **All:** matches all Action Requested types
 - **Acknowledge**
 - **Approve**
 - **Complete**
 - **FYI**
- **Action Requested Group:** You may filter your action requests by the group to which the action requests were sent. For example, if you are a member of two groups, you might use this filter to display only the action requests you've received as part of one of those groups.
- **Document Type:** Use this to filter based on Document Type.
 - The document's type tells KEW who needs to take what action on that document. This determines where KEW routes the document.
 - You can use the Type filter to display only documents of one type, or to display all of your documents except one type. Click the checkbox next to **Exclude?** if you do NOT want to display the selected Type.
 - To choose a Type for your filter, click the magnifying glass icon on the Type row. This takes you to the **Document Type Lookup** page.

Enter information in one or more of these fields to find the Document Type you need for your filter, then click the Search button. This displays a list of document Types that match your search. Find the one you need, and click return value for that Type. This returns you to the previous page and automatically enters it in the Type field. For example, in the screenshot below you searched for a Type with maintenance in the Name. If you want to use the first one in the list, KIM Principal Maintenance Document, click return value in the row for KIM Principal Maintenance Document.

Figure 3.15. Document Type Lookup

Document Type Lookup [create new](#)

* required field

Parent Name:	<input type="text"/>
Name:	<input type="text"/>
Label:	<input type="text"/>
Id:	<input type="text"/>
Active Indicator:	<input checked="" type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Both
<input type="button" value="search"/> <input type="button" value="clear"/> <input type="button" value="cancel"/>	

39 items retrieved, displaying all items.

Actions	Id	Name	Parent Name	Label	Active Indicator	Application ID
edit copy	2011	DocumentTypeDocument	RiceDocument	Workflow Maintenance Document Type Document	Yes	TRAVEL (System Default)
edit copy	2012	RoutingRuleDocument	RiceDocument	Rule Maintenance Document Type Document	Yes	TRAVEL (System Default)
edit copy	2023	KualNotification		Notification	Yes	TRAVEL (System Default)

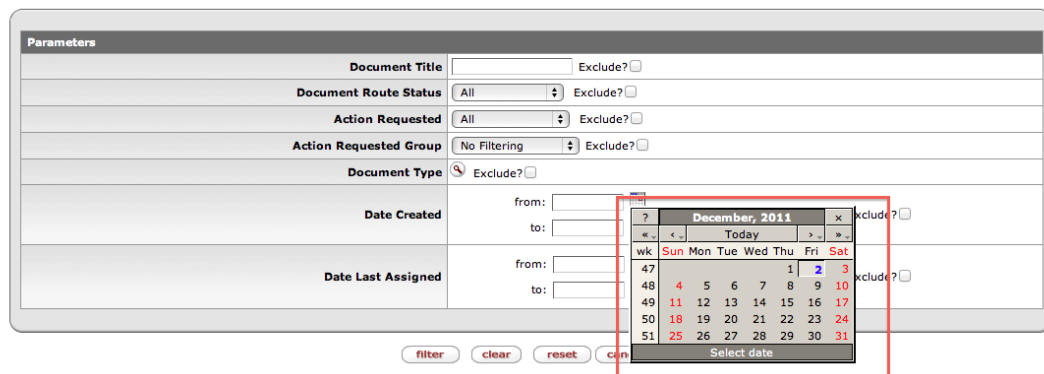
Note

If you do not find the Type that you need, you can either do another search or click the link for return with no value to return to the Action List Filter page without a Type.

- Date Created:** Use this filter criteria to select only action requests for documents that were (or were not) created between dates that you choose. You must enter each date in this format: MM/DD/YYYY, e.g., 12/25/2000 for December 25, 2000. Instead of typing the date in the field, you can click the small Calendar icon next to the date field and then click the date you need from the calendar. When you click a date on the calendar, the small calendar disappears and that date is automatically entered in the date field. Use the small arrowheads on either side of the month's name to display other months. A single arrowhead moves the calendar one month forward or back. A double arrowhead moves the calendar to the same month a year ahead or back.

For example, if you could click the single arrowhead to the left of "April" on this calendar, the month displayed would change to March 2009. If you could click the double arrowhead to the right of "April", the month would change to April 2010.

Figure 3.16. Date Widget



- Date Last Assigned:** Use this filter criteria to select only action requests for documents that were (or were not) assigned to someone between dates that you choose. You must enter each date in this format: MM/DD/YYYY. For more detail on entering or selecting dates, see Date Created above.

Clear Filter

When you have used a filter on your Action List, you can see a Clear Filter button at the top of your page:

Figure 3.17. Action List: Clear Filter Button



Click this button to display all of your action requests again.

Note

If your Action List is not filtered, you cannot see the Clear Filter button.

Remember, you can click the Filter button at the top of your Action List page to return to the Filter page and change or Reset your filters.

Action Requests and Actions Guide

Overview

Action Requests are one of the core functions of the Quali Enterprise Workflow (KEW) system. An Action Request is a call from KEW to a person or group to take action on a document. Action Requests are created when the system or a person routes (sends) a document to another user and ask that user to do something with the document. You might ask someone to approve an expense, for example, or to acknowledge that he or she received a copy of an expense report.

Once an Action Request has been created in KEW, the associated document is sent to the appropriate users' action lists. A user can open his or her Action List and from there, view information about the document, open the document, and/or take the requested action. If a user takes an action, for example, approves or disapproves a document, that is called an Action Taken.

The path that the document takes from its starting point to final approval is called its route. Each point along that path where a person needs to do something with the document (approve, acknowledge, etc.) is called a node on the route.

Types of Action Requests

Each action request has one row on your Action List. There are five types of Action Requests:

Table 3.1. Types of Action Requests

Action Requested	Description	Stops Document Until Action is Taken?
Complete	You need to Complete the document. You may cancel a document when you have a Complete action request on it.	Yes
Approve	You should verify the document and either Approve or Disapprove it. You may cancel a document when you have an Approve action request on it.	Yes
Acknowledge	You need to acknowledge (agree) that you have seen the document	No If there are no other requests, the document changes to Processed status even if there are outstanding Acknowledge requests.
FYI	You have been told about this document and you need to Clear the FYI on it	No If there are no other requests, the document changes to Final status even if there are outstanding FYI requests.
Print	You need to print this document and take some action with it, such as mailing it to a vendor.	

- You can only Disapprove or Cancel a document if you have a Complete or Approve request.
- Complete and Approve are essentially the same: A Complete action satisfies an Approve request, and an Approve action satisfies a Complete request.
- When you Clear an FYI request, that action is not recorded on the Route Log (history of where the document has been and what has been done on it) for that document.

There are other actions that you can take in addition to Action Requests. We will look at those in the Action Taken section below.

Action Request Recipients

Every action request has a recipient. The recipient is the individual or group of individuals whose action is requested on the document. There are three types of recipients:

1. Person: An individual
2. Group: A group of users who are members of a KIM Group
3. Role: A group of users who are members of a KIM Role

Roles are groups of users or KIM Groups associated with a document.

The first user in a Group to take action on a request satisfies (completes) that request. Role requests can be set up to work the same way as Group requests or to require that everyone in that Role approves the document before the request is satisfied.

Action Request Hierarchies

In certain situations, you can use roles for action requests and you can delegate action requests. You can think of these as trees or as a parent action request that has children action requests.

Roles are groups of Principals or Groups that KEW creates based on rules your institution sets up. Since KEW can only create an action request for a single person or group but a role may have several people or groups in it, KEW creates an action request for the role (a parent request), and that action request creates child action requests that go to each person or group in that role.

The parent role request can have any number of children requests that are each either a user request or a group request.

Action Request Delegation

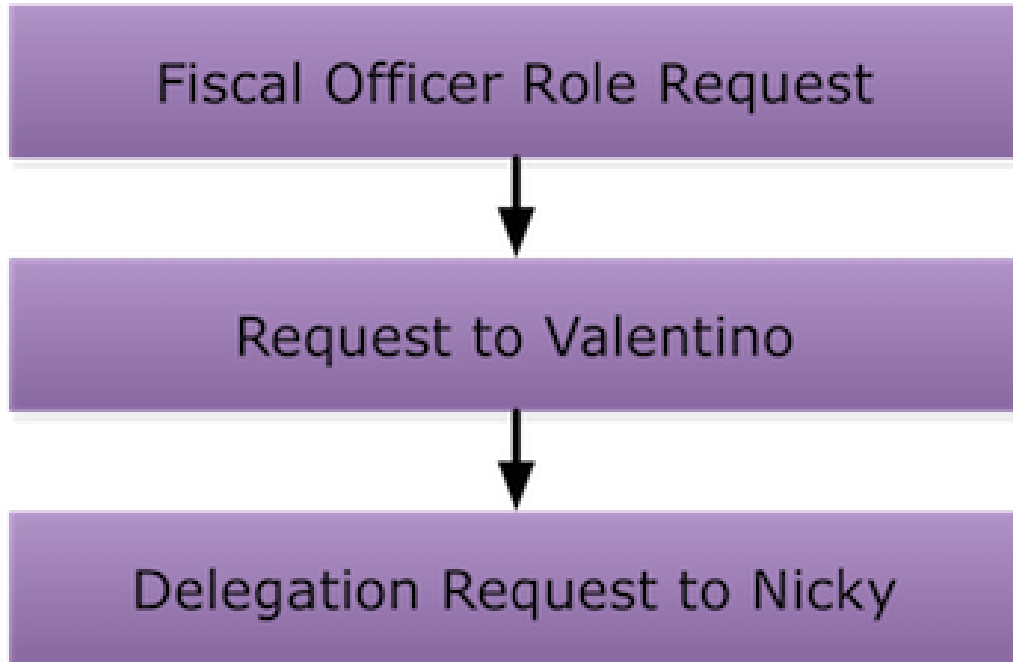
In KEW, you may delegate, or give, your request authority to other people. For example, if Joe is the fiscal officer for an account but he doesn't actually approve documents, he may delegate his approval authority to Jane. Both Joe and Jane can now take actions on Joe's action requests

If Jane approves a document as Joe's delegate, Joe's action request is satisfied. If Joe approves the document, Jane's action request is satisfied.

Combining Role Requests and Delegation

You can combine a role request with a delegate. In fact, in the previous example, the action request that is sent to Joe as a fiscal officer is really a role request. The action request tree for that scenario is:

Figure 3.18. Delegation Tree Example



This tree is three levels deep and shows a role delegating to a user. You can also have a role delegate to another role.

Action Request Activation and Deactivation

Activation

When action requests are created, they are in the Initialized state. They stay in this state until KEW determines that they need to be Activated. When a request becomes Activated, it appears in the user's Action List.

When action requests are created, they can be set to ignore previous actions or not.

- If *Ignore Previous* is true, the activation process will not consider previous actions that the user has taken.
- If *Ignore Previous* is false, the activation process will consider previous actions by the user and may even consider the action request satisfied by previous user actions. (This is sometimes referred to as a request being "auto-approved.")

During request activation, if the Ignore Previous flag is false and KEW determines that a previous action satisfies the current request, it will Deactivate the request instead of Activating it. Activation begins at a root request, but Deactivation begins at the request where KEW finds that a satisfying action has occurred if that request is set to not ignore previous actions.

Note

Action request structures can be hierarchical. Activation of requests always starts at the parent request and works down the tree, activating each level of the tree in turn.

Roles can have an All Approve policy. All Approve means that all members of a role must approve the document before the entire request is deactivated. (See Deactivation below.)

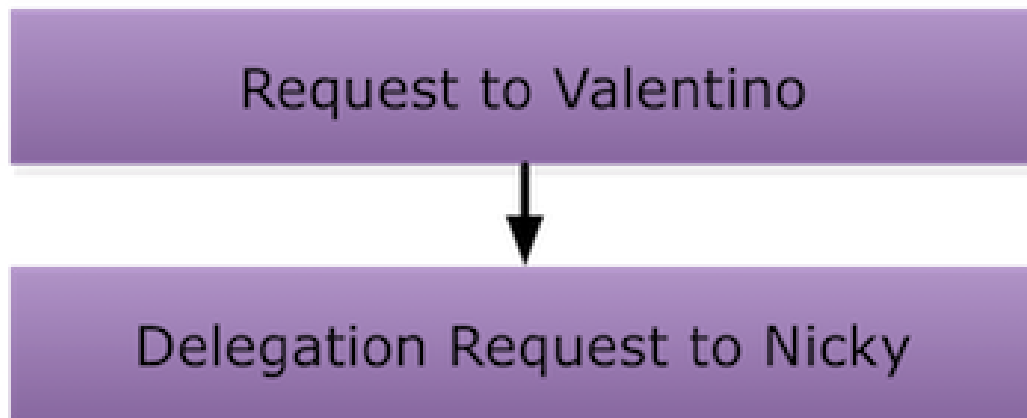
Deactivation

When a user takes action on a document, that action may start the deactivation of other action requests on the document. For example, if Joe has a pending request and he takes an Approve action on it, his request is deactivated. Request deactivation changes the status of the request from Initialized or Activated to Deactivated.

If Joe's action taken causes other action requests to be Deactivated, his action taken is associated with all of the requests that it Deactivates.

Unlike Activation, Deactivation always starts at the request that was satisfied by the action taken. For example, if Joe is delegating to Jane and Jane issues an Approve action on her request, Deactivation starts at Jane's request. However, if Joe issues the Approve action, Deactivation starts at the parent.

Figure 3.19. Delegation Tree Example: Deactivation



If there are no roles set to "All Approve" on an action request, one action can Deactivate the entire request tree. This is because KEW works up and down the tree, Deactivating requests as it goes.

However, when there is an "All Approve" role, requests for all parties within the role must be Deactivated before the parent role request is Deactivated. For example, if we have an "All Approve" role request with three children requests, when one of the users takes action, their request is Deactivated, but since there are two other requests that still need to be satisfied, the Deactivation cannot go back to the parent role request. When the last of the three users has taken action, the parent role request is Deactivated.

Actions and Action Takens

Actions are how a user interacts with KEW. Actions can be performed in response to an Action Request (such as an Approve action) or they can be user-initiated (such as the routing of a document). Most, but not all, actions create Action Takens. The Action Taken is recorded on the Route Log and associated with a satisfied action request, where appropriate.

The KEW Action library contains these Actions, explained below:

- Acknowledge
- Approve

- Blanket Approve
- Cancel
- Clear FYI
- Complete
- Create Document
- Disapprove
- Log Document
- Move Document
- Return to Previous Node
- Route Document
- Save
- Recall
- Superuser Actions
 - Superuser Approve Document
 - Superuser Approve Node
 - Superuser Approve Request
 - Superuser Cancel
 - Superuser Disapprove

Acknowledge Action

Use this action to satisfy an Acknowledge action request. When you take an Acknowledge action, it usually means that you have examined the document. Acknowledgement provides no real authority over the document; you can only take the Acknowledge action on this document.

When you take an Acknowledge action, KEW finds all pending acknowledge requests for that document that are routed to you and Deactivates them. It then records an Acknowledge Action Taken and associates it with the Deactivated requests. You may take an Acknowledge action even if you have no Acknowledge Action Requests.

Approve Action

Use this action to satisfy an Approve or Complete action request. When you use an Approve action, it means that you have looked at the document and approve that transaction.

When you use the Approve action, KEW Deactivates all pending approve or complete requests that have been routed to you for this document. You may take an Approve action even if you have no Approve or Complete action requests.

Blanket Approve Action

Use the Blanket Approve action to force a document to complete a set of action requests. This allows certain users to "push" a document through its routing. To take this action, you must have the appropriate KIM Permission (or be in the Blanket Approve Group which is set in the Document Type, but that is a deprecated feature). Unless you select a specific node (point in the document's routing path) when you take a Blanket Approve Action, KEW finds all pending Approve or Complete requests at the current level in the request tree and Deactivates them. It records a Blanket Approve Action Taken and associates it with each of the Deactivated requests.

Then, KEW sends Acknowledge action requests to each person whose action request was satisfied by the Blanket Approve. It does this for all levels in the request tree until it reaches the end point set in the Blanket Approve Action or it reaches the end of the request tree.

Cancel Action

Use this action when a document is no longer valid and you need to cancel it. You must have a pending Approve or Complete action request on the document before you can Cancel it. Cancelling a document is similar to Disapproving it except that the Cancel action does not send out notifications.

When you take a Cancel action, KEW finds all pending requests on the document and Deactivates them. It records a Cancel Action Taken and associates it with the Deactivated requests. Then it sets the document's status to CANCELLED and the document is effectively dead.

Clear FYI Action

Use this action to satisfy an FYI action request. You usually get an FYI just to notify you about a document. You don't need to take any action on an FYI request, but if you want to remove that FYI and document from your Action List, take the Clear FYI action.

When you take the Clear FYI action KEW locates all pending FYI requests on this document that are routed to you and Deactivates them. It then records an FYI Action Taken. You must have a pending FYI request on a document to use the Clear FYI action.

Complete Action

Use this action to satisfy an Approve or Complete action request. When you use a Complete, it means that you have looked at the document and completed any of the necessary information on the document so it can proceed. Complete action requests are typically created as the result of a Save action or in response to an Exception Routing request.

When you use the Complete action KEW finds all pending Approve or Complete requests that are routed to you and Deactivates them. If the document is in the Exception state, it changes it back to Enroute. It then records a Complete Action Taken. You may take a Complete action even if you have no Complete or Approve action requests.

Create Document Action

This action creates a new document in KEW of the chosen Document Type. This action does not record an Action Taken of any sort but simply begins a new document and assigns a document ID to it. The document begins in Initiated status. The Document type determines how the document is routed and what must be done for it. For example, an expense report might be routed (sent) to two or three different people for approval before it is paid.

Disapprove Action

When you have an Approve action request, you use a Disapprove action if the document does not meet the approval criteria. You must have a pending Approve or Complete request on the document before you can Disapprove it. Disapproving a document is similar to Cancelling it, except that Disapproval sends out notifications.

When you Disapprove a document, KEW finds all pending requests on the document and Deactivates them. Next, it creates Acknowledge notification requests to all users that had Approve or Complete action requests at the current node and all previous nodes. It also sends an Acknowledge notification request to the initiator of the document. Then the document's status is set to Disapproved and the document is effectively dead.

Log Document Action

This action simply enters a message on the document. It records an Action Taken but it is never associated with any action requests. This action displays a message on the document in the Route Log.

Move Document Action

This action moves a document either backward in the route path or forward. Moving the document backward works like the Return to Previous Node action and moving the document forward works like a Blanket Approve action. The difference between these is that the Action Taken is recorded as a Move action.

Return to Previous Node Action

Use this action to move a document to a previous node in the route. You must have a pending Approve or Complete request on the document before you can use this action.

When you use this action, KEW Deactivates all pending requests on the document and sets the current node on the document to the previous node that you requested (this is called the target node). It records a Return to Previous Node Action Taken and associates it with the Deactivated requests. Any requests on nodes between the current node and the target node are removed, effectively erasing them. KEW then sends the document through again to recreate requests on the target node. This way a document is "returned" to a previous node, rolling it back to its previous state at that point in the route.

Route Document Action

Use the Route Document action to route an Initialized or Saved document to other users. Only the initiator of the document can take this action unless the user has an "Initiate Document" KIM permission (or initiator_must_route policy is set to false in the Document Type, although this feature is deprecated). The Action Taken for a Route Document action is Complete.

When you use the Route Document action, it Deactivates all pending requests for the initiator of the document and it associates a Complete Action Taken with the Deactivated requests. The document's state is then set to Enroute.

Save Document Action

Use this action to put a newly created document in the Action List of the document's initiator. You can only use this action on a document in an Initiated status. When you use this action, KEW creates and Activates a Complete action request for the document's initiator and then changes the document's status to Saved.

Recall

Use this action to recall a document that has been submitted to your action list. You can only use this action on a document if you have a specific role for the application and/or document you are trying to recall. When you use this action, you are given the option to recall the document to your action list for updating or you can recall and cancel the document with one action if it is no longer needed.

Superuser Actions

Superuser actions let you move a document past workflow nodes where it may be held up because another user is unavailable to take action on it or due to a system problem. Superuser actions are an administrative tool and safety net. Superusers are designated with a KIM Permission (or they are listed in the Document Type, but this feature is deprecated).

Using the Superuser Functions

To use the Superuser functions, go to Document Search (it is a link in the left menu) and click the Superuser Search link.

Now, do a normal document search for the document on which you need to perform a Superuser action. When you find the document, click the Document/Notification Id link. If you are authorized to perform a Superuser action on this document, this takes you to the Superuser page. Otherwise, KEW displays a message that you are not authorized to take Superuser action on this document.

Superuser Approve Document Action

This action fully approves a document, Deactivating all remaining routing requests. KEW also records a Superuser Approve Action Taken and associates it with the Deactivated requests. The document's state changes to Approved and the document is scheduled for processing. The document status automatically changes to Final when it goes into the Workflow Engine.

Superuser Approve Node Action

This action approves the document through all nodes up to (but not including) a specified node. When the document gets to the specified node, requests are created as usual. This action is exactly the same as Blanket Approve except that no notification requests are created from the Superuser Approve Node action.

Superuser Approve Request Action

This action approves a single pending Approve or Complete action request. This works the same way as a standard Approve action except that the Superuser is acting in place of the responsible user.

Superuser Cancel Action

This action cancels the document. This action works exactly the same way as the standard Cancel action except that the Superuser does not need to have a pending request to Cancel a document.

Superuser Disapprove Action

This action disapproves the document. This action works the same way as the standard Disapprove action except that the Superuser does not need to have a pending request to perform the action. Also, on a Superuser disapprove, KEW does not send notification requests.

Superuser Return to Previous Node Action

This action returns the document to a previous node in the route path. This action works exactly the same way as the standard Return to Previous Node action except that the Superuser does not need to have a pending request to perform the action.

Superuser Quick Action's Tab

With the release of Rice 2.1 a new option is available for applications to enable. Superuser's can now access documents directly from the standard document search and quickly perform approval, acknowledgment, FYI, or disapproval actions from this new tab on the document. This streamlined version of the still available superuser form of document search will allow for common actions to be taken in a more user friendly fashion. Pending actions are displayed in a list that the super user can check off and take the actions necessary all at once (see example below). Before actions can be taken, a annotation must be provided.

By default, the new tab will not appear on a document.

```
<kul:superUserActions />
```

This needs to be included in the JSP of a document for the superuser tab to be shown. The logic for whether it should actually display anything is contained inside of this. Developers may also look at KualiMaintenanceDocument.jsp in the Rice code for an example.

Figure 3.20. New Superuser Tab Example

The screenshot shows a web interface for 'Super User Action'. At the top, there is a tab labeled 'Super User Action' with a 'hide' button. Below the tab is a table with the following data:

<input type="checkbox"/>	Action	Requested Of	Time/Date	Annotation
<input type="checkbox"/>	APPROVE	Two, Admin	03/28/2012 05:52 PM	

Below the table is a text area labeled 'Annotation*' and two buttons: 'take selected actions' and 'disapprove document'.

Chapter 4. Documents

eDocLite Overview

eDocLite is a framework designed for rapid development and implementation within an existing Quali Enterprise Workflow infrastructure. It allows for the development of simple web applications, their forms and 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.

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.

The form uses an XSLT stylesheet to generate the html code. 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.

Note

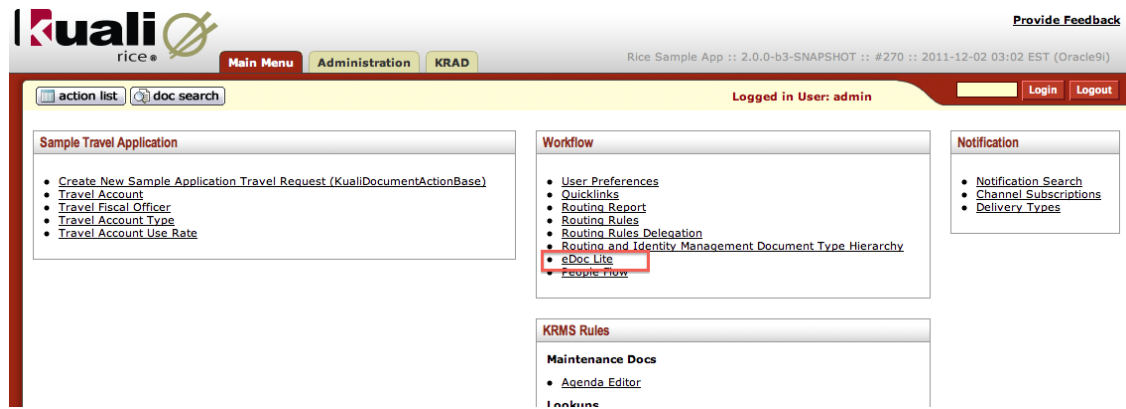
Key Ideas:

- 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'.

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



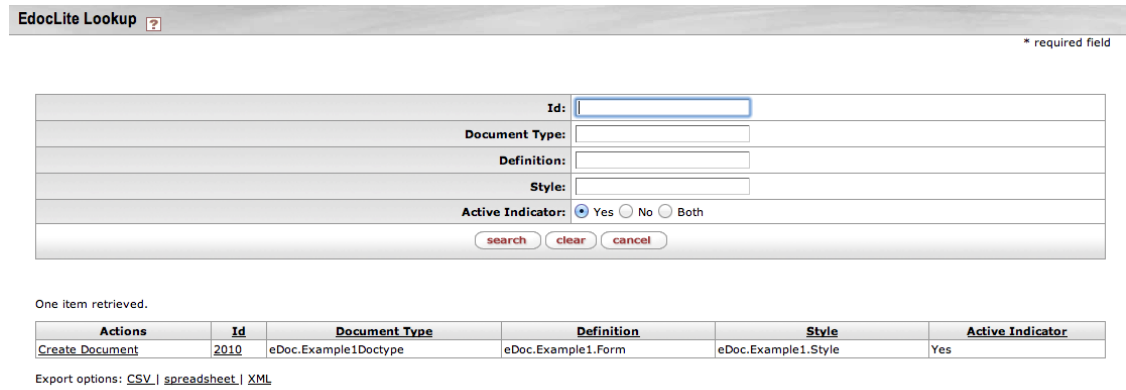
Finding the eDocLite Lookup Screen

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.
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.

Field	Description
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

eDocLite Inquiry

Figure 4.4. eDocLite Inquiry

The screenshot shows a window titled 'EdocLite' with a 'hide' button. The main area displays the following information:

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

At the bottom of the window, there are two buttons: 'export' and '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.

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 4.2. 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 4.3. 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 4.4. Routing Action and Annotation, and Note Attributes

Field	Description
Set annotation	The area to add annotation.
Action buttons	<ul style="list-style-type: none"> route: begins and continues routing the eDocLite document. save: saves the information currently on the eDocLite document. 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:

Editing Document

** Questions with an asterisk are required.

Indiana University EDL EDocLite Example	eDocLite Example 1 Form		
User Information			
Full Name*	<input type="text"/>		
Other Information			
Requested Date of Implementation:*	<input type="text"/>		
Campus:*	IUB <input type="button" value="v"/>		
Description of Request:	<input type="text"/>		
(Check all that apply)			
<input type="checkbox"/> My research/sponsored program work is funded by NIH or NSF.			
<input type="checkbox"/> My research/sponsored program work involves human subjects.			
Supporting Materials			
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.			
Create Note			
Author	Date	Note	Action
admin, admin	12/02/2011	<input type="text"/>	<input type="button" value="save"/>
Attachment: <input type="button" value="Choose File"/> no file selected			
<input type="button" value="submit"/> <input type="button" value="save"/> <input type="button" value="cancel"/>			

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.

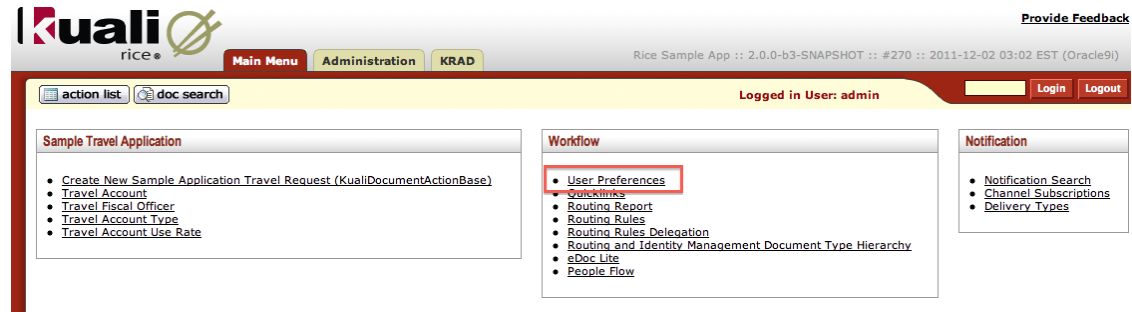
Hidden fields

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 5. Preferences

Workflow allows for users to individually configure certain aspects of the system. You should be able to access the User Preferences from the Main portal page:

Figure 5.1. Workflow Channel: User Preferences Link



After clicking the link you should be taken to Workflow Preferences screen:

Figure 5.2. Workflow Preferences

There are three configuration sections on this screen:

General Preferences

Note

Many of these preferences can have their system wide default values changed via configuration parameters. Look in the KEW technical documentation for information on how to override the default values.

Table 5.1. User Preferences Attributes

Name	Default Value	Description
Automatic Refresh Rate	15	How often your action list updates (minutes).

Preferences

Name	Default Value	Description
Action List Page Size	10	# of actions displayed on action list
Email Notification	Immediate	When action items emails should be sent. Can be none, immediate, daily, weekly
Receive Primary Delegation Emails	True	User will receive primary delegate emails
Receive Secondary Delegation Emails	False	User will receive secondary delegate emails
Delegator Filter	Secondary Delegators on Action List Page	Determines what is displayed on the action list page. Options: <ul style="list-style-type: none">• Secondary Delegators on Action List Page• Secondary Delegators only on Filter Page
Primary Delegate Filter	Primary Delegators on Action List Page	Determines what is displayed on the action list page. Options: <ul style="list-style-type: none">• Primary Delegators on Action List Page• Primary Delegators only on Filter Page

Fields Displayed in Action List Preferences

These are the columns that will be displayed on the user's action list page.

Table 5.2. User Preferences: Fields Displayed Attributes

Name	Default Value
Document Type	TRUE
Title	TRUE
ActionRequested	TRUE
Initiator	TRUE
Delegator	TRUE
Date Created	TRUE
Date Approved	FALSE
Current Route Node(s)	FALSE
Workgroup Request	TRUE
Document Route Status	TRUE
Application Document Status	FALSE
Clear FYI	TRUE
Use Outbox	TRUE

Document Route Status Colors for Action list Entries

Each Document Route Status can be displayed with a certain color on the action list. This section allows you to configure which colors are used. By default, no colors are set.

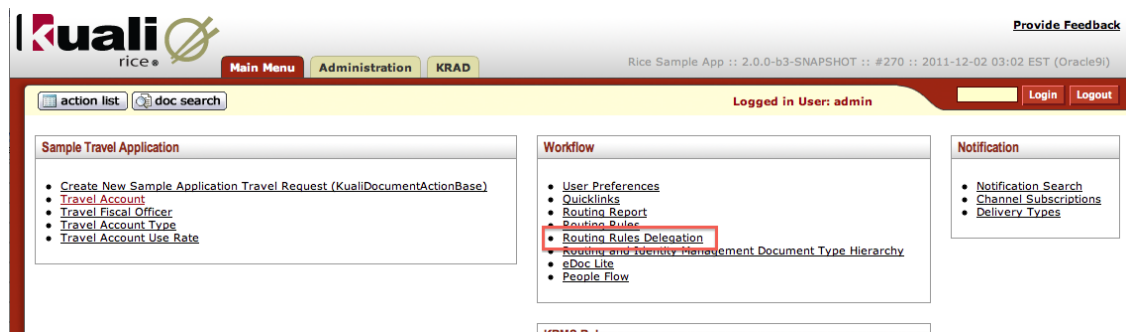
Chapter 6. Routing

Please Note: The following sections on routing and rules are written as they relate the legacy/traditional KNS workflow. A new concept of workflows, called PeopleFlows, and their use and maintenance is covered in a separate section.

Delegate Rules and Delegation Routing Rules

Rice provides a quick and convenient way to obtain information on Delegation Rules through the **Delegate Rule Lookup**. This Lookup also lets you create new Delegation Rules using the **create new** function and edit existing Delegation Rules.

Figure 6.1. Workflow Channel: Routing Rules Delegation Link



To access the **Delegate Rule Lookup** screen, click **Routing Rules Delegation** on the Rice Main Menu.

Figure 6.2. Delegation Lookup

The screenshot shows the 'Delegate Rule Lookup' form. At the top left is the title 'Delegate Rule Lookup' and a help icon. At the top right is a 'create new' button and a note '* required field'. The form contains several input fields: 'Parent Rule Id:', 'Responsibility Id:', 'Document Type:', 'Rule Template:', 'Description:', 'Group Reviewer Namespace:', 'Group Reviewer Name:', 'Person Reviewer:', 'Person Reviewer Type:' (with radio buttons for User, Group Member, and Both), 'Active:' (with radio buttons for Yes, No, and Both), 'Rule Id:', and 'Delegation Type:' (with radio buttons for Primary, Secondary, and Both). At the bottom are 'search', 'clear', and 'cancel' buttons.

Use the **Delegate Rule Lookup** screen to find information about a Delegation Rule. You can also create a new Rule by clicking the create new button in the upper right corner of the screen. The create new function is described in the Routing Rule Creation section later in this document.

When you do a search from the Delegate Rule Lookup screen, you can:

- View the Delegation Rule information returned by the search

- Edit a Delegation Rule
- Copy an existing Delegation Rule and create a new one based on that copy

Information about the Lookup function is in the next section of this document. The other two functions are described in the Routing Rule Delegation Maintenance section later in this document.

To find a Delegation Rule quickly, enter information in one or more of the fields on the Lookup screen, then click the **search** button. If you are not familiar with the search fields, you can leave all of the fields blank and click the **search** button. The **Delegation Rule Lookup** page then displays all of the Delegation Rule templates.

The fields on this screen are described in the Routing Rule Delegation Maintenance section of this document.

Note

Parent Rule Id is a unique identification number for a Rule Template; therefore, searching by the Parent Rule Id can be the fastest way to find a particular Rule.

After you click the search button, Delegate Rule Lookup displays a list of Rules that looks similar to this:

Figure 6.3. Delegation Lookup: Results Example

Actions	Delegation Rule Id	Parent Rule Id	Name	Rule Template	Active	Document Type	Delegation Type
edit copy	1641	1046	A4CAFSAA-5352-SEC8-D2CA-977FD066FD50	WorkflowDocumentDelegationTemplate	Yes	TravelRequest	Secondary

Click **edit** on any row to go to the **Routing Rule Delegation** document for that Rule. You can use this document to do maintenance on that Rule. This is discussed later in this document, in the Routing Rule Delegation Maintenance section.

Click **copy** on any row to go to the **Routing Rule Delegation** document with the information from that Rule copied into the document. This lets you add a new Delegation Rule based on that information. This is discussed later in this document in the Routing Rule Delegation Maintenance section.

Click a **Delegation Rule Id** to go to the **Delegation Rule Inquiry** screen for that Rule. This screen displays specific information about how this Rule is set up in Rice. This screen is discussed later in this document in the **Delegation Rule Inquiry** section.

Click a Parent Rule Id or a Name to go to the **Rule Inquiry** screen, where you can see details about that Parent Rule ID. More information about this screen can be found in the Rule Inquiry section of this User Guide.

Click a Rule Template to go to the **Rule Template Inquiry** screen. More information about this screen can be found in the Rule Template Inquiry section of this User Guide.

The standard export options apply to this list of search results. An explanation of the export options can be found in the Common Features and Functions section of this User Guide.

Delegation Rule Inquiry

Figure 6.4. Delegation Inquiry

This screen provides specific information on how a **Rule** works. From this screen, you can either click the **close** button to return to the previous screen or you can export an XML version of this information.

The fields on this screen are described in the Rule Maintenance section of this document.

Delegate Routing Rule Creation

Figure 6.5. Delegation Rule: Create New Screen

To create a new Delegate Routing Rule, start by selecting a Parent Rule to associate with your new Delegate Rule:

1. Do a field lookup on the Select parent rule field.
2. This displays the **Rule Lookup** screen where you search in the Rules your institution has defined in Rice to find the appropriate Parent Rule for your new Delegate Routing Rule.
3. When you have found the appropriate Parent Rule, click its return value link to return to the Delegate Routing Rule Creation screen with this Parent Rule.

Figure 6.6. Delegate Routing Rule: Create New, Parent Selection

Select	Reviewer	Type	Action Request Code
<input type="radio"/>	admin	PERSON	APPROVE

4. Information from the Parent Rule you selected is now in the **Select Parent Responsibility to Delegate From** section of the **Delegate Routing Rule Creation** screen.
5. Click the **Select** button for this parent Rule.
6. Click the **continue** button.
7. Rice displays the **Routing Rule Delegation** screen. Use this screen to enter information for your new Rule. This is described in the **Routing Rule Delegation Maintenance** section (next).

Routing Rule Delegation Maintenance

You can use the **Routing Rule Delegation** document screen for both editing existing Delegation Rules and for adding new Delegation Rules to Rice.

Figure 6.7. Routing Rule Delegation: Overview

Note

When Rice first displays the **Routing Rule Delegation** screen, all of the tabs except the Notes and Attachments, Ad Hoc Recipients, and Route Log tabs, are expanded. They are all collapsed in the screen print above so you can see all of them.

Document Layout

The Routing Rule Delegation screen you see when creating a new Rule has eight tabs:

- Document Overview
- Delegation Details
- Delegate Rule
- Persons
- Groups
- Notes and Attachments
- Ad Hoc Recipients
- Route Log

The **Delegation Details**, **Delegate Rule**, **Persons**, and **Groups** tabs are discussed in the sections that follow. The remaining four tabs are commonly used throughout Rice and are described in the Common Features and Functions section of the User Guide.

Delegation Details Tab

The Delegation Details tab is where you associate a Parent Rule with a Delegate Rule. It is also where you assign the Reviewer role.

The Delegation Details screen is displayed differently, depending on whether you are creating a new Rule or editing/copying a Rule. The version you see when creating a new Rule looks similar to this:

Figure 6.8. Routing Rule Delegation: Details Section

New	
Parent Rule	Parent Rule Id: 1034 Description: workflowdocumenttemplate description
Parent Rule Responsibility	Reviewer: admin Type: PERSON Action Request: APPROVE

The information displayed is a copy of the information for the Parent Rule. Descriptions of these fields are below. Change this information as needed, then click the **submit** button to create this new Rule.

The version you see when editing or copying a Rule looks similar to this:

Figure 6.9. Routing Rules Delegation: Edit/Copy View

Old		New	
Parent Rule	Parent Rule Id: 1046 Description: Destination Rule	Parent Rule	Parent Rule Id: 1046 Description: Destination Rule
Parent Rule Responsibility	Reviewer: user4 Type: PERSON Action Request: APPROVE	Parent Rule Responsibility	Reviewer: user4 Type: PERSON Action Request: APPROVE

When you first see this version, the information in both columns is the same. You can modify the information in the fields in the right column, but not the left column. After changing the information appropriately in the right column, click the submit button to save your changes to this Rule.

The fields are the same in both versions of the screen:

Table 6.1. Routing Rules Delegation Attributes

Field	Description
Parent Rule Id	Rule Id of the Parent Rule for this Rule
Description	General description of this Rule
Reviewer	The person responsible for reviewing a document in the routing process for this Rule
Type	The Type of the Reviewer
Action Request	The type of Action this Reviewer can take. This may be: <ul style="list-style-type: none"> • Acknowledge • Complete • Approve • FYI

Delegate Rule tab

The Delegate Rule tab is displayed differently, depending on whether you are creating a new Rule or editing/copying a Rule. The version you see when creating a new Rule looks similar to this:

Figure 6.10. Routing Rules Delegation: Delegate Rule Tab

Enter the appropriate information for your new Delegate Rule in these fields. There are descriptions of each of these fields below.

The version you see when editing or copying a Rule looks similar to this:

Figure 6.11. Routing Rule Delegation: Delegate Rule Tab, Edit/Copy View

When you first see this version, the information in both columns is the same. You can modify the information in the fields in the right column, but not the left column. After changing the information in the right column appropriately.

The fields are the same in both versions of the screen:

Table 6.2. Routing Rule Delegation: Delegate Rule Tab Attributes

Field	Description
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.
Rule Template	A Rule Template serves as a pattern or design for routing Rules.
Delegation Type	Required. Options for Delegation Type are: <ul style="list-style-type: none"> • Primary: This means that the delegator turns over full authority to the delegate. The Action Requests for this delegator only show up in the Action List of the primary delegate. A primary delegate must be registered in KEW to be in effect. • Secondary: This means that the secondary delegate acts as a temporary backup delegator who has the same authority as the primary approver, or the secondary delegate acts as the delegator if the delegator is not available. Documents appear in the Action Lists of both the delegator and the secondary delegate. When either person acts on the document, it disappears from both Action Lists. A secondary delegate must be registered in KEW as being in effect for a specific time period(s). • Both
Name	he unique Name of the routing delegate Rule you are creating or editing. Note: You are not required to enter a Name, but if you don't, Rice assigns a unique Name to the Rule.
Description	The general description of the Rule
From Date	The start date of the routing Rule
To Date	The expiration date of the routing Rule
Force Action	This is a Yes or No flag. A checkmark means Yes, blank means No.
Active	A True/False flag to indicate whether this Routing Delegate Rule is active in Rice. If it is active, you can use it in Rice.

Persons tab

Rice displays the **Persons** tab differently, depending on whether you are creating a new Rule or editing/copying a Rule. This tab lets you add people to the Rule and set the type of Action each person needs to take under this Rule. The version of the Persons screen that you see when creating a new Delegate Rule looks similar to this:

Figure 6.12. Routing Rules Delegation: Persons Tab

Enter the appropriate information for the Persons section of your new Delegate Rule in these fields. There are descriptions of each of these fields below.

The version you see when editing or copying a Rule looks similar to this:

Figure 6.13. Routing Rules Delegation: Persons tab, Copy/Edit View

Use the top portion of this tab to add a new Person to this Rule and set the Action that Person needs to take and its Priority. Use the lower portion of this tab to change existing values for this Rule.

Note

When you create a new person, you must click the **add** button when you have entered information in the three required fields.

Table 6.3. Routing Rules Delegation: Persons Tab Attributes

Field	Description
Person	Required. An individual user who receives the document for the Action Requested.
Action Request	Required. The type of Action this person needs to take. This Action may be one of these: <ul style="list-style-type: none"> • Acknowledge • Complete • Approve • FYI
Priority	Required. The priority associated with the Person in this Rule. One is the highest Priority.

Groups tab

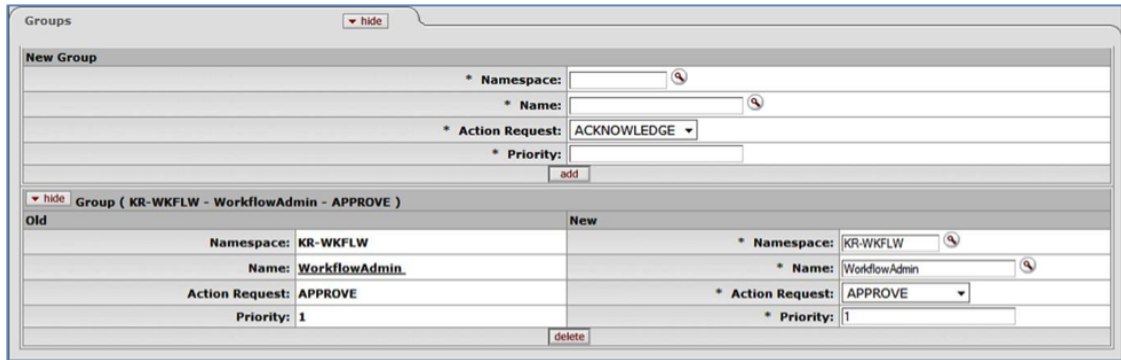
The Groups tab differently, depending on whether you are creating a new Rule or editing/copying a Rule. The version you see when creating a new Rule looks similar to this:

Figure 6.14. Routing Rules Delegation: Groups Tab

Enter the appropriate information for the Groups section of your new Delegate Rule in these fields. These fields are described below.

The version you see when editing or copying a Rule looks similar to this:

Figure 6.15. Routing Rules Delegation: Groups Tab, Edit/Copy View



Use the top portion of this tab to add a new Group. Use the lower portion to change existing information for a Group.

Note

When you create a new group, you must click the **add** button after you enter information in the required fields.

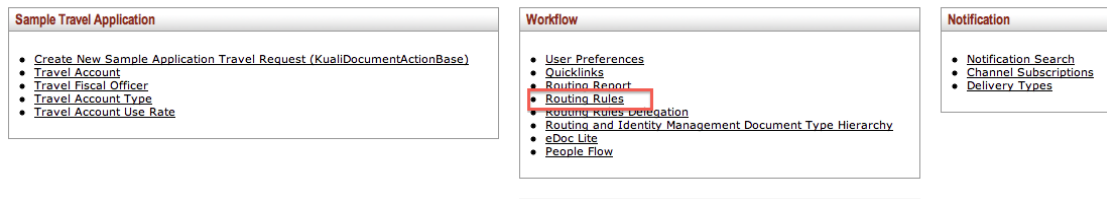
Table 6.4. Routing Rules Delegation: Groups Tab Attributes

Field	Description
Namespace	The Namespace to which this Group will belong. A Namespace is a way to set both Permissions and Entity Attributes. Each Namespace instance is one level of Permissions and Entity Attributes and is one record in Rice.
Name	The unique name of this Group
Action Request	Required. The type of Action this Group is expected to take. The Action may be one of these: <ul style="list-style-type: none"> • Acknowledge • Complete • Approve • FYI
Priority	Required. The priority associated with the Group in this Rule. One is the highest Priority.

Rule Lookup

The Rice system provides a quick and convenient way to obtain information on Rules through the **Rule Lookup** feature. This feature also provides an entry point to creating a new Rule through the **create new** function.

Figure 6.16. Workflow Channel: Routing Rules Link



To access the **Rule Lookup** screen click on **Routing Rules** on the Rice **Main Menu**.

The **Rule Lookup** screen is used to view, edit or copy information about routing rules defined in Rice. It is also the starting point for creating a new rule using the **Routing Rule Creation** screen.

Figure 6.17. Routing Rules Lookup

Click on the **create new** button to go to the Routing Rule Creation screen. The process for creating a new Routing Rule is discussed in the **Routing Rule Creation** section of this document.

To limit your search results you can select one or more of the available search criteria.

The fields on this screen are described in the **Rule Maintenance** section of this document.

Note

Rule Id is a unique identification number for a particular rule template; therefore, search by rule Id can produce the most accurate search result.

If you are not familiar with the search criteria provided above you can simply click on the **search** button and the Rule Lookup page will automatically provide a list of rule templates.

Rule Lookups return a list that looks similar to the following:

Figure 6.18. Routing Rules Lookup: Results Example

Actions	Name	Rule Id	Rule Template	Active	Document Type	Delegate Rule
edit copy	SendNotificationRequest.Reviewers	1044	ReviewersRouting	Yes	SendNotificationRequest	No
edit copy	TravelRequest.Destination.LasVegas	1046	TravelRequest-DestinationRouting	Yes	TravelRequest	No
edit copy	TravelRequest.Supervisor	1049	TravelRequest-TravelerRouting	Yes	TravelRequest	No
edit copy	TravelRequest.DeanDirector	1050	TravelRequest-SupervisorRouting	Yes	TravelRequest	No
edit copy	TravelRequest.FiscalOfficer	1051	TravelRequest-SupervisorRouting	Yes	TravelRequest	No
edit copy	659D86718DD514C7E0404F8189D877C3	1052	TravelRequest-AccountRouting	Yes	TravelRequest	No
edit copy	eDoc.Example1Doctype.IUB	1103	eDoc.Example1.Node1	Yes	eDoc.Example1Doctype	No
edit copy	eDoc.Example1Doctype.IUPUI	1106	eDoc.Example1.Node1	Yes	eDoc.Example1Doctype	No
	RecipeMastersGroupApproval	1642		Yes	RecipeParentMaintenanceDocument	No
	ChickenRecipeMastersGroupApproval	1643		Yes	RecipeMaintenanceDocument	No
	MagazineGroupApproval	1644		Yes	RecipeMaintenanceDocument	No

Clicking **edit** on any line item takes you to the **Rule Maintenance Document Type Document** where you can perform maintenance on that rule. This is discussed later in this document in the Rule Maintenance section.

Clicking **copy** on any line item takes you to the **Rule Maintenance Document Type Document** where you can add a new rule based upon the rule you are copying. This is discussed later in this document in the Rule Maintenance section.

Clicking a **Rule Id** will take you to the Rule Inquiry screen for that specific rule. This is discussed later in this document in the **Rule Inquiry** section.

Clicking a **Document Type** will take you to the **Document Type Inquiry** screen for that specific rule. This is discussed in the **Document Type Inquiry** section of the User Guide.

The standard export options apply to this list. Explanation of the export options can be found in the **Common Features and Functions** section of the User Guide.

Rule Inquiry

Figure 6.19. Routing Rules Inquiry

This screen provides specific information on how the **Rule** is set up in Rice. The fields on this screen are described in the **Rule Maintenance** section of this document.

Routing Rule Creation

Figure 6.20. Routing Rules Creation

When creating a new Routing Rule you start by entering the **Document Type** and the **Rule Template** you want associated with your new Rule, then click the continue button. This takes you to the **Rule Maintenance Document Type Document** described in the **Rule Maintenance** section that follows.

Table 6.5. Routing Rule Creation Attributes

Field	Description
Document Type	You can select a new document type by clicking on the field lookup button located at the right side of the Doc Type field.
Rule Template	The name of the Rule Template with which this rule is associated. A Rule Template serves as a pattern or design for the routing rules and is a mechanism for providing configuration information to rules. You can select a new template by clicking on the field lookup button located at right side of the Rule Template field.

Routing Report

The routing report is a tool to check workflow routing. You select a rule template and enter any routing data required by the rule template. You can then view the workflow routing that would be assigned to such a document.

Figure 6.21. Workflow Channel: Routing report



Using the Report

In the default implementation of Rice you can find the link to the Routing Report on the "Main" tab of the portal inside the "Workflow" pane. If for some reason the link is not visible on the "Main" tab you can still directly navigate to the Routing Report. To access the tool directly, replace the last part of the Rice Portal URL (`portal.jsp`) with `kew/RoutingReport.do` (for example, `http://my.kuali.host/kuali/portal.jsp` would become `http://my.kuali.host/kuali/kew/RoutingReport.do`). The following page is displayed:

Figure 6.22. Routing Report: Template Selection



Select a Rule Template

The first thing to do is select the rule template to use for the test from the drop-down list. When you select a template the page will update to show the routing data that you need to enter to run the test:

Figure 6.23. Routing Report: Template Selection, Detail



Enter Routing Data

The data entry fields for the routing data use the standard controls described in the Standard Buttons on Lookup Screens document. Here is the example with the routing data:

Figure 6.24. Routing Report: Routing Data Entry

The specific information you need to enter is specific to the rule template you selected. Once you have entered the information, click the button to display the report:

Figure 6.25. Routing Report: Route Log View

Report Contents

The upper section of the report (labeled "ID: 0") contains the basic document information fields as if this was an actual document; several fields are blank since there is no actual document.

The lower section of the report (labeled "Pending Action Requests") contains the routing that the document would be given. Clicking the button on any action displays more detail of the action:

Figure 6.26. Routing Report: Route Log View, Pending Action Requests

Rule Maintenance

The **Rule Maintenance Document Type Document** screen is used for both editing existing Rules and for adding new Rules to Rice.

Figure 6.27. Rule Maintenance Document Type Document

Rule Maintenance Document Type Document ?		Doc Nbr: 3069	Status: INITIATED
		Initiator: admin	Created: 09:25 PM 12/02/2011

* required field

Document Overview	<input type="button" value="show"/>
Rule	<input type="button" value="show"/>
Rule Attributes	<input type="button" value="show"/>
Persons	<input type="button" value="show"/>
Groups	<input type="button" value="show"/>
Notes and Attachments (0)	<input type="button" value="show"/>
Ad Hoc Recipients	<input type="button" value="show"/>
Route Log	<input type="button" value="show"/>

Note

When Rice presents the **Rule Maintenance Document Type Document** screen all of the tabs except the Notes and Attachments, Ad Hoc Recipients and Route Log are expanded.

Document Layout

The Rule Maintenance Document Type Document screen you see when creating a new rule is composed of seven tabs:

- Document Overview
- Rule
- Persons
- Groups
- Notes and Attachments
- Ad Hoc Recipients
- Route Log

The screen you see when you are editing an existing Rule includes an additional tab called **Roles**.

The Rule, Persons, Groups and Roles tabs are unique and are discussed in the sections that follow. The remaining four tabs are commonly used throughout Rice and are described in the Common Features and Functions section of the User Guide.

Rule tab

The Rule tab is presented differently depending on whether you are creating a new Rule, or editing/copying a Rule. The version you see when creating a new Rule looks similar to this:

Figure 6.28. Rule Maintenance Document Type Document: Rule Tab

The version you see when editing or copying looks similar to this:

Figure 6.29. Rule Maintenance Document Type Document: Rule Tab, Edit/Copy View

Table 6.6. Rule Maintenance Document Type Document: Rule Tab Attributes

Field	Description
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.
Rule Template	The name of the Rule Template with which this rule is associated. A Rule Template serves as a pattern or design for the routing rules and is a mechanism for providing configuration information to rules.
Name	The unique Name of the routing Name you are creating or editing. Note: It is not required that you enter a Name, but if you don't the system will assign a unique value.
Description	The general description for the rule selected.
From Date	The start date of the routing rule.
To Date	The expiration date of the routing rule.
Force Action	Force Action is used to control workflow routing by making the rule actions mandatory. This is a yes or no flag. A check means yes, blank means no.
Active	A true/false flag to indicate if Routing Rule is active in Rice can be applied or not.

Persons tab

The Persons tab is presented differently depending on whether you are creating a new Rule, or editing/copying a Rule. The version you see when creating a new Rule looks similar to this:

Figure 6.30. Rule Maintenance Document Type Document: Person Tab

The screenshot shows a web interface titled 'Persons' with a 'hide' button. Below it is a 'New Person' section containing three input fields: '* Person:' (with a search icon), '* Action Request:' (a dropdown menu), and '* Priority:' (a text input). An 'add' button is located at the bottom right of this section.

The version you see when editing or copying looks similar to this:

Figure 6.31. Rule Maintenance Document Type Document: Person Tab, Edit/Copy View

The screenshot shows the 'Persons' tab with a 'hide' button. Below it is a 'New Person' section with fields for '* Person:', '* Action Request:' (set to 'ACKNOWLEDGE'), and '* Priority:'. An 'add' button is at the bottom. Below this is a section titled 'Person (admin - APPROVE)' which is split into 'Old' and 'New' columns. The 'Old' column shows 'Person: admin', 'Action Request: APPROVE', and 'Priority: 1'. The 'New' column shows 'Person: admin', '* Action Request: APPROVE', and '* Priority: 1'. A 'delete' button is at the bottom right of this section.

Table 6.7. Rule Maintenance Document Type Document: Person Tab Attributes

Field	Description
Person	An individual user who receives the document for the Action Requested.
Action Request	The type of action this person can take. <ul style="list-style-type: none"> • Acknowledge • Complete • Approve • FYI
Priority	The priority associated with the person in this rule. Lower values mean higher priorities, with 1 being the highest priority.

Groups tab

The Groups tab is presented differently depending on whether you are creating a new Rule, or editing/copying a Rule. The version you see when creating a new Rule looks similar to this:

Figure 6.32. Rule Maintenance Document Type Document: Groups Tab

The screenshot shows a web interface titled 'Groups' with a 'hide' button. Below it is a 'New Group' section containing four input fields: '* Namespace:' (with a search icon), '* Name:' (with a search icon), '* Action Request:' (a dropdown menu set to 'ACKNOWLEDGE'), and '* Priority:' (a text input). An 'add' button is located at the bottom right of this section.

The version you see when editing or copying looks similar to this:

Figure 6.33. Rule Maintenance Document Type Document: Groups Tab, Edit/Copy View

The screenshot shows a web interface for managing groups. At the top, there is a 'Groups' tab with a 'hide' button. Below it, there are two main sections: 'New Group' and 'Old'. The 'New Group' section has four required fields: 'Namespace', 'Name', 'Action Request' (a dropdown menu), and 'Priority'. An 'add' button is located below these fields. The 'Old' section is divided into 'Old' and 'New' columns. The 'Old' column shows the current values: Namespace: KR-WKFLW, Name: WorkflowAdmin, Action Request: APPROVE, and Priority: 1. The 'New' column shows the same fields for editing. A 'delete' button is located at the bottom right of the 'Old' section.

The top portion of this tab is used to add a new Group while the lower portion is used to change existing values.

Table 6.8. Rule Maintenance Document Type Document: Groups Tab Attributes

Field	Description
Namespace	The Namespace to which this Group will belong. 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.
Name	The unique name of the Group.
Action Request	The type of action this Group is expected to take. <ul style="list-style-type: none"> • Acknowledge • Complete • Approve • FYI
Priority	The priority associated with the Group in this rule. One being the highest.

Rule Template Lookup

The **Rules Template Lookup** function is available from any screen with the field lookup button next to the Rule Template field. Simply click on the button and you will go a screen that looks like this.

Figure 6.34. Rule Template Lookup

The screenshot shows a 'Rule Template Lookup' form. It has a title bar with a question mark icon and a '* required field' label. The form contains three input fields: 'Name', 'Id', and 'Description'. Below the fields are three buttons: 'search', 'clear', and 'cancel'.

The **Rule Template Lookup** page will automatically provide a complete list of the rule templates. You can reduce the number of items returned by your search by entering values in the fields, or if you are not sure or familiar with the key word(s) for the search criteria, simply click on the search button.

Figure 6.35. Rule Template Lookup: Results Example

Return Value	Id	Name	Description	Delegation Rule Template
return_value	1015	WorkflowDocumentDelegationTemplate	WorkflowDocumentDelegationTemplate	
return_value	1016	WorkflowDocumentTemplate	Workflow Document Template	...
return_value	1017	Ack1Template	Acknowledgement 1 Template	
return_value	1018	Ack2Template	Acknowledgement 2 Template	
return_value	1019	RemoveReplaceWorkgroupTemplate	Remove/Replace Workgroup Routing	
return_value	1021	RemoveReplaceRuleTemplate	Remove/Replace Rule Routing	
return_value	1023	ReviewersRouting	Routes to channel reviewers	

Clicking on the **Id** field will take you to the **Rule Template Inquiry** screen for that particular Rule Template.

Table 6.9. Rule Template Lookup Attributes

Field	Description
Return Value	This is a link which will copy the Rule Template information back to the screen which you originated from.
Id	A unique Id number for a specific rule template.
Name	The name of the rule template.
Description	A general description of the rule template.
Delegation Rule Template	The delegate rule template selected.

Rule Template Inquiry

The **Rule Template Inquiry** screen provides specific information about one particular **Rule Template**.

Figure 6.36. Rule Template Inquiry

The screenshot shows a web interface for a Rule Template Inquiry. At the top, there is a header 'Rule Template' with a 'hide' button. Below this, the details for a specific rule template are displayed:

- Id:** 1016
- Name:** WorkflowDocumentTemplate
- Description:** Workflow Document Template
- Delegation Rule Template:** WorkflowDocumentDelegationTemplate

 Below the details, there is a section for 'Rule Attributes' with another 'hide' button. At the bottom of the screen, there are two buttons: 'export' and 'close'.

Clicking the export button will export a copy of the complete Rule Template in XML format.

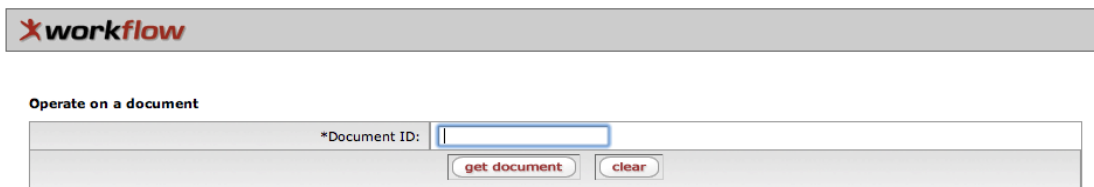
Chapter 7. KEW Document Operations

Document Operation

The Document Operation screen allows for low-level modifications to document data. It's available from the Administrator channel in the portal.

In certain scenarios or failure cases it may be necessary to make modifications to the document so that the state of the document in the KEW system is consistent with that of the integrating application. It may also be necessary to make modifications to the XML content of a document if, for example, there is a bug in the application integrating with KEW which results in incomplete or insufficient XML content to allow for proper routing.

Figure 7.1. Initial Screen



The screenshot shows the 'xworkflow' logo at the top. Below it, the text 'Operate on a document' is displayed. A form field labeled '*Document ID:' is present, followed by two buttons: 'get document' and 'clear'.

The initial screen prompts for the ID of a document to load. The administrator is then presented with a view of the document and can perform various operations on it. The screen is divided into various sections, including:

- Document Actions - Additional functions for reassigning and reprocessing document
- Document - simple data associated with the document
- Action Requests - the Action Requests associated with the document, includes requests for action which have already been satisfied
- Actions Taken - The actions that have been taken against this document (i.e. Approved by user X)
- Action Items - Items related to this document that are in users' Action Lists
- Route Node Instances - The node instances that form the document's instantiated route path
- Branch States - The branches on this document and the state of those branches
- Annotation - An annotation that will show up on the Route Log when the operation is performed.

Each of the pieces of data within the aforementioned sections has a set of radio buttons at the top that indicates Update, Delete, or No Operation. No Operation is the default. If it is desired to change or delete one of these pieces of data then the appropriate button should be selected. This is to guard against unintended or accidental changes to the document.

Each of the different sections is described in detail below.

Document Actions

Figure 7.2. Document Actions

Document Actions	
Queue Document <input type="button" value="submit"/>	
Index Searchable Attributes <input type="button" value="submit"/>	
Queue Document Refresh <input type="button" value="submit"/>	
Queue Document Blanket Approve <input type="button" value="submit"/>	User: <input type="text"/>
Queue Document Move <input type="button" value="submit"/>	Action Taken Id: <input type="text"/>
	Node Names: <input type="text"/>
Queue Action Invocation <input type="button" value="submit"/>	User: <input type="text"/>
	Action Item Id: <input type="text"/>
	Action Code: <input type="text"/>

Several functional buttons are added under the Document Action section:

- Queue Document - Requeuing and reprocessing the document by the engine
- Index Searchable Attributes - Update searchable data of the document
- Queue Document Requeuer - Refresh document and regenerate request of current node
- Queue Document Blanket Approve - Move blanket approve document forward; User, Action Taken Id, and Node Names are required
 - User - Enter initiator's network Id
 - Action Taken Id - Enter an entry's action taken Id
 - Node Names - Enter a comma separated list of node names
- Queue Document Move - Move document forward or backward; User, Action Taken Id, and Node Names are required
 - User - Enter initiator's network Id
 - Action Taken Id - Enter an entry's action taken Id
 - Node Names - Enter a comma separated list of node names
- Queue Action Invocation - Reassign action request based on initiator, entry ID, and action code; User, Action Item Id, and Action Code are required
 - User - Enter initiator's network Id
 - Action Item Id - Enter an entry's action item Id
 - Action Code - A, F, K, or C; A for Approve, F for FYI, K for acknowledge, and C for Complete

Document

Figure 7.3. Document

Document	
Document ID: 3063	<input type="radio"/> Update <input checked="" type="radio"/> No Operation
* Document Version:	1
* Initiator ID:	admin
* Initial Route Node Instances:	3044
* Route Status:	ENROUTE
* Route Level:	2
* Create Date:	02:44 PM 01/10/2012
* Doc Status Modification Date:	02:46 PM 01/10/2012
Approved Date:	
Finalized Date:	
Route Status Modification Date:	
Route Level Modification Date:	
Doc Type ID:	3057
Doc Title:	
Application Doc ID:	
Doc Content:	<pre><documentContent> <applicationContent> <data ediName="VacancyNotice"> <version current="false" date="Tue Jan 10 14:46:53 UTC 2012" version="0"> <field name="oaa"></pre>

- Document Version - A legacy field indicating whether the document was upgraded from version 2.0 to 2.1
- Initiator ID - The workflow id of the initiator
- Initial Route Node Instances - The ID of the initial route node instance on the document
- Route Status - The current status of the document
- Route Level - A legacy field providing a numerical representation of where the document is in the route path
- Create Date - The initial date the document was created, doesn't not reflect whether the document was routed, saved, etc.
- Doc Status Modification Date - The date at which the document's status was last modified
- Approved Date - The date at which the document's state transitioned to APPROVED
- Finalized Date - The date at which the document's state transitioned to FINAL
- Route Status Modification Date - Legacy value, similar to Doc Status Modification Date
- Route Level Modification Date - Legacy value, no longer used
- Doc Type ID - The ID of the DocumentType definition for this document
- Doc Title - The title of the document
- Application Doc ID - A special id that can be set by client applications to associate the document to an ID in their system
- Override Indicator - Legacy value, no longer use

- Lock Code - Legacy value, no longer used
- Doc Content - The XML Content of the document

Action Requests

Figure 7.4. Action Requests

Action Requests	
Action Request ID: 2369	<input type="radio"/> Update <input type="radio"/> Delete <input checked="" type="radio"/> No Operation
* Document Version:	1
* Document ID:	3063
* Route Node Instance ID:	3045
* Action Requested:	APPROVE
* Create Date:	01/10/2012
* Status:	DONE
* Priority:	1
* Route Level:	1
* Responsibility ID:	2145
Responsibility Description:	School/RC of BI-BUS for
Action Request Parent ID:	
Recipient Type:	group
Person ID:	
Workgroup ID:	10008
Role Name:	
Qualified Role Name:	
Qualified Role Label:	
Action Taken ID:	2337
Force Action:	No
Approve Policy:	F
Delegation Type:	
Current Indicator:	Yes
Annotation:	
Action Request ID: 2370	<input type="radio"/> Update <input type="radio"/> Delete <input checked="" type="radio"/> No Operation
* Document Version:	1
* Document ID:	3063
* Route Node Instance ID:	3046
* Action Requested:	APPROVE

- Document Version - A legacy field indicating whether the request was upgraded from version 2.0 to 2.1
- Document ID - The ID of the associated document
- Route Node Instance ID - The ID of the node instance that this request is attached to
- Action Request - The type of action that is requested
- Create Date - The date the request was created
- Status - The current status of the request
- Priority - The activation priority of the request
- Route Level - A legacy field providing a numerical representation of where in the route path the request was generated
- Responsibility ID - The id of the responsibility associated with this request (relates to Rules and/or Route Modules)
- Responsibility Description - A description of the responsibility of this request

- Action Request Parent ID - ID of the parent action request if there is one
- Recipient Type - The type of recipient for this request (user, workgroup, or role)
- Person ID - If the recipient type is "user", the workflow id of the user recipient
- Workgroup ID - If the recipient type is "workgroup", the workgroup id of the workgroup recipient
- Role Name - If the recipient type is "role", the name of the role
- Qualified Role Name - If the recipient type is "role", the value of the qualified role
- Qualified Role Label - If the recipient type is "role", the label for the qualified role
- Action Taken ID - If this request has been satisfied, the id of the ActionTaken that satisfied the request
- Ignore Previous - The ignore previous indicator of the request
- Approve Policy - The approve policy of the request (only used by role requests)
- Delegation Type - If the request is a delegation, the type of delegation (primary or secondary)
- Current Indicator - Indicates if the request is "Current" or not
- Annotation - The value of the annotation on the request

Actions Taken

Figure 7.5. Actions Taken

Actions Taken	
Action Taken ID: 2336	<input type="radio"/> Update <input type="radio"/> Delete <input checked="" type="radio"/> No Operation
* Document ID:	3063
* Document Version:	1
* Action Taken:	COMPLETED
* Action Date:	01/10/2012
* Action Taken Person ID:	admin
Delegator Person ID:	
Delegator Workgroup ID:	
Current Indicator:	Yes
Annotation:	
Action Taken ID: 2337	<input type="radio"/> Update <input type="radio"/> Delete <input checked="" type="radio"/> No Operation
* Document ID:	3063
* Document Version:	1
* Action Taken:	APPROVED
* Action Date:	01/10/2012
* Action Taken Person ID:	supervisor

- Document ID - The ID of the associated document
- Document Version - A legacy field indicating whether the Action Taken was upgraded from version 2.0 to 2.1
- Action Taken - the type of the action that was taken
- Action Date - the date at which the action was taken
- Action Taken Person ID - the workflow id of the user or delegate who took action

- Delegator Person ID - if this action was performed by a delegate, the workflow id of the person whose authority was delegated
- Delegator Workgroup ID - if this action was performed by a delegate, the workflow id of the Workgroup whose authority was delegated
- Current Indicator - Indicates if the Action Taken is "Current" or not, non-current actions have been revoked by an action such as ReturnToPreviousNode
- Annotation - The value of the annotation on the Action Taken

Action Items

Figure 7.6. Action Items

Action Items	
Action Item ID: 10227	<input type="radio"/> Update <input type="radio"/> Delete <input checked="" type="radio"/> No Operation
* Document ID:	<input type="text" value="3063"/>
* Doc Type Name:	<input type="text" value="VacancyNotice"/> 🔍
* Doc Type Label:	<input type="text" value="Vacancy Notice"/>
* Doc Handler URL:	<input type="text" value="http://dev1.rice.kuali.or"/>
* Date Assigned:	<input type="text" value="01/10/2012"/> 📅
* Action Request ID:	<input type="text" value="2370"/>
* Action Requested:	<input type="text" value="APPROVE"/> ⌵
* Responsibility ID:	<input type="text" value="2191"/>
* Person ID:	<input type="text" value="director"/> 🔍
Workgroup ID:	<input type="text" value="10005"/> 🔍
Role Name:	<input type="text"/> 🔍
Delegator Person ID:	<input type="text"/> 🔍
Delegator Workgroup ID:	<input type="text"/> 🔍
Document Title:	<input type="text"/>

- Document ID - The ID of the associated document
- Doc Type Name - The name of the DocumentType for this item
- Doc Type Label - The label of the document type
- Doc Handler URL - The URL used to access the doc handler for this item
- Date Assigned - The creation date of the item
- Action Request ID - The id of the Action Request from which this item is derived
- Action Requested - The type of action requested by this item
- Responsibility ID - The responsibility id of the associated request
- Person ID - The workflow id of the person responsible for the item
- Workgroup ID - The workgroup id of the workgroup responsible for the item
- Role Name - If the item was derived from a role request, the name of the role
- Delegator Person ID - If the item was delegated, the workflow id of the delegating party
- Delegator Workgroup ID - If the item was delegated, the workgroup id of the delegating party

- Document Title - The title of the document

It is important to note that the ActionItem is a de-normalized representation of an Action Request on the document that is used to render the Action List in an efficient matter. Therefore, it contains some copies of data from both the document and the request itself.

Route Node Instances

Figure 7.7. Route Node Instances

Route Node Instances	
Route Node Instance ID: 3045	<input type="radio"/> Update <input type="radio"/> Delete <input checked="" type="radio"/> No Operation
Instance Name:	School
Active Indicator:	<input type="text" value="No"/>
Complete Indicator:	<input type="text" value="Yes"/>
Initial Indicator:	<input type="text" value="No"/>
Previous Route Node Instances:	3044
Next Route Node Instances:	3046
Route Node States:	None
Route Node Instance ID: 3044	<input type="radio"/> Update <input type="radio"/> Delete <input checked="" type="radio"/> No Operation
Instance Name:	Initiated
Active Indicator:	<input type="text" value="No"/>

- Instance Name - The name of the node
- Active Indicator - indicates if the node is active
- Complete Indicator - indicates if the node's processing has completed
- Initial Indicator - indicates if the node has been processed by the engine yet
- Previous Route Node Instances - A comma-separated display of the IDs of the previous Route Node Instances of the node
- Next Route Node Instances - A comma-separated display of the IDs of the next Route Node Instances of the node
- Route Node States - A representation of the state attached to the node

The Route Node Instances are modeled as a Directed Acyclic Graph starting at the node instance pointed to by the Initial Route Node Instances field in the Document section. Therefore, if you delete a route node instance, it will follow all links through its set of Next Route Node Instances and delete those as well.

Branch States

Figure 7.8. Branch States

Branch States	
Branch ID: 3043	<input type="radio"/> Update <input checked="" type="radio"/> No Operation
Branch Name:	<input type="text" value="PRIMARY"/>
Branch States:	None

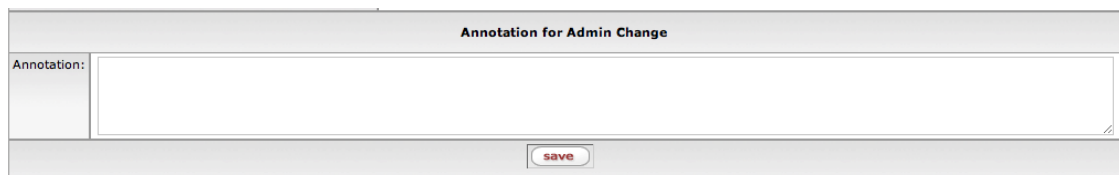
- Branch Name - The name of the branch
- Branch State ID - The ID of that piece of branch state
- Branch State Key - The key of the branch state

- Branch State Value - The value of the branch state

All documents are required to have at least one branch that is named PRIMARY. Therefore, it is advisable to not rename the PRIMARY branch.

Annotation

Figure 7.9. Annotation



The screenshot shows a web form titled "Annotation for Admin Change". It features a text input field labeled "Annotation:" and a "save" button at the bottom right.

Here you can enter an annotation explaining the changes being made. This will be logged on the Route Log so that it can be preserved as part of the audit trail for the document.

Once the changes have been made on the document operation screen, hit the Save button and the changes will be executed on the server. Remember that in order for a change to take place the appropriate radio button must be selected on the data that requires modification.

Practical Uses of Document Operation

- Requeuing a document that was stuck
- Moving a document to FINAL
- Rolling a document back to a previous point in the route path

Examples of each are outlined below

Requeuing a document that was stuck

Moving a document to FINAL

Rolling a document back to a previous point in the route path

Chapter 8. Steps to Building a KEW Application

Preface

In its simplest form, KEW is merely a set of services that can be used to submit documents to a workflow engine and then interact with those documents as the progress through the routing process. Therefore, there are many different ways to build an application that uses KEW. Quali Rice itself has a few built-in solutions (eDocLite and KNS) that make it easier to build applications that use KEW. Alternatively, an application can be built from scratch or retrofitted to use KEW.

In this section, we will look at some common approaches to designing and building an application which leverages KEW. However, it is by no means exhaustive and is simply meant to get you started and give you ideas as you embark upon development of your own applications that use Quali Enterprise Workflow.

Initial Steps - Determine the Routing Rules

Determine to whom you want to route the document and when it should be routed. For example, in the **Travel Request Sample Workflow Client Application**, the steps in the routing process are:

1. Someone submits a travel request for a traveler
2. Traveler receives an *Approve Action Item*
3. Traveler's supervisor receives *Approve Action Item*
4. Traveler's dean/director receives *Acknowledge Action Item*
5. Fiscal Officer for account(s) receives *Approve Action Item*

Configure the Process Definition

In KEW, process definitions are attached to **Document Types**. The Document Type allows for configuration of various pieces of the business process in addition to the process definition.

The Document Type is defined in XML format. KEW can ingest files containing this Document Type configuration to set up the specified workflows and then executes the workflows based on that configuration.

One example of routing configuration is the Travel Request application. The Document Type configuration is defined in the following four XML files:

TravelRoutingConfiguration.xml - Defines the **travelDocument** Document Type, including *PostProcessor*, *docHandler*, and *routeNodes*:

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

```

<description>Create a New Travel Request</description>
<label>Travel Request</label>
<postProcessorName>org.kuali.rice.kns.workflow.postprocessor.KualiPostProcessor</postProcessorName>
<superUserGroupName namespace="TVL">SuperUserGroup</superUserGroupName>
<blanketApproveGroupName namespace="TVL">BlanketApproveGroup</blanketApproveGroupName>
<defaultExceptionGroupName namespace="TVL">ExceptionGroup</defaultExceptionGroupName>

<docHandler>${application.url}/travelDocument2.do?methodToCall=docHandler</docHandler>
<routePaths>
  <routePath>
    <start name="Initiated" nextNode="DestinationApproval" />
    <requests name="DestinationApproval" nextNode="TravelerApproval" />
    <requests name="TravelerApproval" nextNode="SupervisorApproval" />
    <requests name="SupervisorApproval" nextNode="AccountApproval" />
    <requests name="AccountApproval" />
  </routePath>
</routePaths>
<routeNodes>
  <start name="Initiated">
    <activationType>P</activationType>
  </start>
  <requests name="DestinationApproval">
    <ruleTemplate>TravelRequest-DestinationRouting</ruleTemplate>
  </requests>
  <requests name="TravelerApproval">
    <ruleTemplate>TravelRequest-TravelerRouting</ruleTemplate>
  </requests>
  <requests name="SupervisorApproval">
    <ruleTemplate>TravelRequest-SupervisorRouting</ruleTemplate>
  </requests>
  <requests name="AccountApproval">
    <ruleTemplate>TravelRequest-AccountRouting</ruleTemplate>
  </requests>
</routeNodes>
</documentType>
</documentTypes>
</data>

```

TravelRuleAttributes.xml – Defines the attributes used by the Workflow Engine to determine to whom to route to next:

```

<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <ruleAttributes xmlns="ns:workflow/RuleAttribute" xsi:schemaLocation="ns:workflow/RuleAttribute
resource:RuleAttribute">
    <ruleAttribute>
      <name>EmployeeAttribute</name>
      <className>edu.sampleu.travel.workflow.EmployeeAttribute</className>
      <label>Employee Routing</label>
      <description>Employee Routing</description>
      <applicationId>TRAVEL</applicationId>
      <type>RuleAttribute</type>
    </ruleAttribute>

    <ruleAttribute>
      <name>AccountAttribute</name>
      <className>edu.sampleu.travel.workflow.AccountAttribute</className>
      <label>Account Routing</label>
      <description>Account Routing</description>
      <applicationId>TRAVEL</applicationId>
      <type>RuleAttribute</type>
    </ruleAttribute>
  </ruleAttributes>
</data>

```

TravelRuleTemplates.xml - Defines the **RuleTemplates** that represent each routeNode listed in the Document Type configuration:

```

<?xml version="1.0" encoding="UTF-8"?>

```



```

<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <ruleTemplates xmlns="ns:workflow/RuleTemplate" xsi:schemaLocation="ns:workflow/RuleTemplate
resource:RuleTemplate">
    <ruleTemplate allowOverwrite="true">
      <name>TravelRequest-DestinationRouting</name>
      <description>Destination Routing</description>
      <attributes>
        <attribute>
          <name>DestinationAttribute</name>
        </attribute>
      </attributes>
    </ruleTemplate>
    <ruleTemplate allowOverwrite="true">
      <name>TravelRequest-TravelerRouting</name>
      <description>Traveler Routing</description>
      <attributes>
        <attribute>
          <name>EmployeeAttribute</name>
        </attribute>
      </attributes>
    </ruleTemplate>
    <ruleTemplate allowOverwrite="true">
      <name>TravelRequest-SupervisorRouting</name>
      <description>Supervisor Routing</description>
      <attributes>
        <attribute>
          <name>EmployeeAttribute</name>
        </attribute>
      </attributes>
    </ruleTemplate>
    <ruleTemplate allowOverwrite="true">
      <name>TravelRequest-AccountRouting</name>
      <description>Travel Account Routing</description>
      <attributes>
        <attribute>
          <name>AccountAttribute</name>
        </attribute>
      </attributes>
    </ruleTemplate>
  </ruleTemplates>
</data>

```

TravelRules.xml - Defines the rules (a rule is a combination of *Document Type*, *Rule Template* and *Responsibilities*) that the workflow engine uses to determine to whom to route to next:

```

<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <rules xmlns="ns:workflow/Rule" xsi:schemaLocation="ns:workflow/Rule resource:Rule">
    <rule>
      <name>TravelRequest-DestinationLasVegas</name>
      <documentType>TravelRequest</documentType>
      <ruleTemplate>TravelRequest-DestinationRouting</ruleTemplate>
      <description>Destination Rule</description>
      <ruleExtensions>
        <ruleExtension>
          <attribute>DestinationAttribute</attribute>
          <ruleTemplate>TravelRequest-DestinationRouting</ruleTemplate>
          <ruleExtensionValues>
            <ruleExtensionValue>
              <key>destination</key>
              <value>las vegas</value>
            </ruleExtensionValue>
          </ruleExtensionValues>
        </ruleExtension>
      </ruleExtensions>
      <responsibilities>
        <responsibility>
          <principalName>user4</principalName>
          <actionRequested>A</actionRequested>
        </responsibility>
      </responsibilities>
    </rule>
  </rules>
</data>

```

```

<rule>
  <name>TravelRequest-EmployeeRole</name>
  <documentType>TravelRequest</documentType>
  <ruleTemplate>TravelRequest-TravelerRouting</ruleTemplate>
  <description>Traveler Routing</description>
  <responsibilities>
    <responsibility>
      <role>edu.sampleu.travel.workflow.EmployeeAttribute!employee</role>
      <actionRequested>A</actionRequested>
    </responsibility>
  </responsibilities>
</rule>
<rule>
  <name>TravelRequest-SupervisorRole</name>
  <documentType>TravelRequest</documentType>
  <ruleTemplate>TravelRequest-SupervisorRouting</ruleTemplate>
  <description>Supervisor Routing</description>
  <responsibilities>
    <responsibility>
      <role>edu.sampleu.travel.workflow.EmployeeAttribute!supervisr</role>
      <actionRequested>A</actionRequested>
    </responsibility>
  </responsibilities>
</rule>
<rule>
  <name>TravelRequest-DirectorRole</name>
  <documentType>TravelRequest</documentType>
  <ruleTemplate>TravelRequest-SupervisorRouting</ruleTemplate>
  <description>Dean/Director Routing</description>
  <responsibilities>
    <responsibility>
      <role>edu.sampleu.travel.workflow.EmployeeAttribute!director</role>
      <actionRequested>K</actionRequested>
    </responsibility>
  </responsibilities>
</rule>
<rule>
  <name>TravelRequest-FiscalOfficerRole</name>
  <documentType>TravelRequest</documentType>
  <ruleTemplate>TravelRequest-AccountRouting</ruleTemplate>
  <description>Fiscal Officer Routing</description>
  <responsibilities>
    <responsibility>
      <role>edu.sampleu.travel.workflow.AccountAttribute!FO</role>
    </responsibility>
  </responsibilities>
</rule>
</rules>
</data>

```

Client PlugIn Steps

Your plugin should contain Java classes that correspond to the attributes defined in the XML configuration file. The Travel Request Sample Client contains two attribute classes: *EmployeeAttribute* and *AccountAttribute*. Each of these classes implements these two interfaces:

```

org.kuali.rice.kew.rule.RoleAttribute
org.kuali.rice.kew.rule.WorkflowAttribute

```

Using the *EmployeeAttribute* as an example, here are the implementations for the *RoleAttribute* interface:

getRoleNames() - Returns a list of role names to display on the routing rule GUI in the KEW web application:

```

private static final Map ROLE_INFO;

static {

```

```

ROLE_INFO = new TreeMap();
ROLE_INFO.put(EMPLOYEE_ROLE_KEY, "Employee");
ROLE_INFO.put(SUPERVISOR_ROLE_KEY, "Supervisor");
ROLE_INFO.put(DIRECTOR_ROLE_KEY, "Dean/Director");
}

public List getRoleNames() {
    List roleNames = new ArrayList();
    for (Iterator iterator = roles.keySet().iterator(); iterator.hasNext();) {
        String roleName = (String) iterator.next();
        roleNames.add(new Role(getClass(), roleName, roleName));
    }
    return roleNames;
}

```

getQualifiedRoleNames() - Returns a list of strings that represents the qualified role name for the given roleName and XML docContent which is attached to the workflow document:

```

/**
 * Returns a String which represent the qualified role name of this role for the given
 * roleName and docContent.
 * @param roleName the role name (without class prefix)
 * @param documentContent the document content
 */

public List<String> getQualifiedRoleNames(String roleName, DocumentContent documentContent) {
    List<String> qualifiedRoleNames = new ArrayList<String>();
    Map<String, List<String>> qualifiedRoles = (Map<String, List<String>>)roles.get(roleName);
    if (qualifiedRoles != null) {
        qualifiedRoleNames.addAll(qualifiedRoles.keySet());
    } else {
        throw new IllegalArgumentException("TestRuleAttribute does not support the given role " + roleName);
    }
    return qualifiedRoleNames;
}

```

resolveQualifiedRole() - Returns a list of workflow users that are members of the given Qualified Role. (Used to help determine to whom to route the document.):

```

/**
 * Returns a List of Workflow Users which are members of the given qualified role.
 * @param routeContext the RouteContext
 * @param roleName the roleName (without class prefix)
 * @param qualifiedRole one of the the qualified role names returned from the {@link
 * #getQualifiedRoleNames(String, DocumentContent)} method
 * @return ResolvedQualifiedRole containing recipients, role label (most likely the roleName), and an
 * annotation
 */

public ResolvedQualifiedRole resolveQualifiedRole(RouteContext routeContext, String roleName, String
qualifiedRole) {
    ResolvedQualifiedRole resolved = new ResolvedQualifiedRole();
    Map<String, List<String>> qualifiedRoles = (Map<String, List<String>>)roles.get(roleName);

    if (qualifiedRoles != null) {
        List<String> recipients = (List<String>)qualifiedRoles.get(qualifiedRole);
        if (recipients != null) {
            resolved.setQualifiedRoleLabel(qualifiedRole);
            resolved.setRecipients(convertPrincipalIdList(recipients));
        } else {
            throw new IllegalArgumentException("TestRuleAttribute does not support the qualified role " +
qualifiedRole);
        }
    } else {
        throw new IllegalArgumentException("TestRuleAttribute does not support the given role " + roleName);
    }
    return resolved;
}

```

Using the *EmployeeAttribute* example, here are the implementations for the *WorkflowAttribute* interface:

getRoutingDataRows() – Returns a list of *RoutingDataRows* that contain the user interface level presentation of the *ruleData* fields. KEW uses the *ruleData* fields to determine where a given document would be routed according to the associated rule:

```
public List<Row> getRoutingDataRows() {
    List<Row> rows = new ArrayList<Row>();
    List<Field> fields = new ArrayList<Field>();
    fields.add(new Field("Traveler username", "", Field.TEXT, false, USERID_FORM_FIELDNAME, "", false, false,
    null, null));
    rows.add(new Row(fields));
    return rows;
}
```

getDocContent() - Returns a string containing this Attribute's *routingData* values, formatted as a series of XML tags:

```
public String getDocContent() {
    String docContent = "";

    if (!StringUtils.isBlank(_uuid)) {
        String uuidContent = XmlUtils.encapsulate(UUID_PARAMETER_TAGNAME, _uuid);

        docContent = _attributeParser.wrapAttributeContent(uuidContent);
    }

    return docContent;
}
```

validateRoutingData() - Validates *routingData* values in the incoming map and returns a list of errors from the routing data. (The user interface calls **validateRoutingData()** during rule creation.):

```
public List validateRoutingData(Map paramMap) {
    List errors = new ArrayList();

    String principalName = StringUtils.trim((String) paramMap.get(PRINCIPAL_NAME_FORM_FIELDNAME));
    if (isRequired() && StringUtils.isBlank(principalName)) {
        errors.add(new WorkflowServiceErrorImpl("principalName is required",
    "accountattribute.principalName.required"));
    }

    if (!StringUtils.isBlank(principalName)) {
        KimPrincipalInfo principal =
    KIMServiceLocator.getIdentityService().getPrincipalByPrincipalName(principalName);
        if (principal == null) {
            errors.add(new WorkflowServiceErrorImpl("unable to retrieve user for principalName '" +
    principalName + "'", "accountattribute.principalName.invalid"));
        }
    }
    if ( errors.size() == 0 ) {
        _principalName = principalName;
    }
    return errors;
}
```

Build PostProcessor and Services

The PostProcessor class should implement the interface:

```
org.kuali.rice.kew.postprocessor.PostProcessorRemote
```

You should use this interface for business logic that should execute when the document transitions to a new status or when actions are taken on the document. The PostProcessor for the Travel Request Client is the class:

```
org.kuali.rice.kns.workflow.postprocessor.KualiPostProcessor
```

that implements the `doRouteStatusChange()` method to update the status of the travel document in the Travel database. The `KualiPostProcessor` in this case is the standard `PostProcessor` used on all documents that are built on the KNS framework.

Package PlugIn

Depending on how the application has been developed (i.e. embedded workflow engine vs. using the engine as a remote service) it may be necessary to package components like the `PostProcessor` into a plugin. See the [Workflow PlugIn Guide](#) for details on how to do this.

Client Web Application Steps

Build the Web Application

Begin to build a Kuali Enterprise Workflow the same as you build any other Java-enabled web application. You build it with all the business logic for the application and, for example, communication to the workflow engine using web services.

As an example, the Travel Request Client Web Application uses Struts, Spring, and OJB.

Build the Service that Connects to the Workflow Engine

For the rest of this section, this guide refers to the Java application communicating with the Kuali Enterprise Workflow as the *Client Application*. The Client Application needs a service that will interact with the workflow system. This service will perform actions such as locating a document in the workflow system and routing the document.

Below are examples from the Travel Request Sample Client. The methods in the `TravelDocumentService` class find a `TravelDocument` in the workflow system, save and route a `TravelDocument`, and validate a `TravelDocument`.

findByDocHeaderId() - Finds a Document in the workflow engine:

```
public TravelDocument findByDocHeaderId(Long docHeaderId, String principalId) {
    if (docHeaderId == null) {
        throw new IllegalArgumentException("invalid (null) docHeaderId");
    }
    TravelDocument result = travelDocumentDao.findByDocHeaderId(docHeaderId);
    if (result != null) {
        // convert DocAccountJoins into FinancialAccounts
        ArrayList accounts = new ArrayList();
        for (Iterator joins = result.getDocAccountJoins().iterator(); joins.hasNext();) {
            DocumentAccountJoin join = (DocumentAccountJoin) joins.next();

            FinancialAccount account = financialAccountService.findByAccountNumber(join.getAccountNumber());

            accounts.add(account);
        }
        result.setFinancialAccounts(accounts);
    }
    try {
```

```
        WorkflowDocument document = new WorkflowDocument( principalId, result.getDocHeaderId());
    } catch (WorkflowException e) {
        LOG.error("caught WorkflowException: ", e);
        throw new RuntimeException(e);
    }
}
return result;
}
```

The **TravelDocumentServiceImpl** class populates the attribute values on the workflow document (Employee, Account) that will be used for future routing. It does this by calling its *getEmployeeAttribute()* and *getAccountAttribute()* methods and adding the results to the workflow document by calling the *addAttributeDefinition()* method.

```
private WorkflowAttributeDefinitionVO getEmployeeAttribute(TravelDocument travelDocument) {
    WorkflowAttributeDefinitionDTO attrDef = new
    WorkflowAttributeDefinitionDTO("edu.sampleu.travel.workflow.EmployeeAttribute");
    String principalName = travelDocument.getTravelerUsername();
    attrDef.addConstructorParameter(principalName);
    return attrDef;
}

private List getAccountAttributes(TravelDocument travelDocument) {
    List accounts = travelDocument.getFinancialAccounts();
    List accountAttributes = new ArrayList();
    for (Iterator accountIterator = accounts.iterator(); accountIterator.hasNext();) {
        WorkflowAttributeDefinitionDTO attrDef = new
        WorkflowAttributeDefinitionDTO("edu.sampleu.travel.workflow.AccountAttribute");
        FinancialAccount account = (FinancialAccount)accountIterator.next();
        attrDef.addConstructorParameter(account.getAccountNumber());
        accountAttributes.add(attrDef);
    }
    return accountAttributes;
}
```

Build the Action Class with Workflow Lifecycle Methods

In the Travel Request Sample Client, the **WorkflowDocHandlerAction** struts action class calls the workflow lifecycle methods (approve, acknowledge, etc.) on the workflow document.

WorkflowDocHandlerAction - Take approve action. (Each workflow action - acknowledge, complete, etc. - is like this):

```
public ActionForward approve(ActionMapping mapping, ActionForm form, HttpServletRequest request,
    HttpServletResponse response) throws Exception {
    LOG.info("entering approve() method ...");
    DocHandlerForm docHandlerForm = (DocHandlerForm) form;
    WorkflowDocument document = docHandlerForm.getWorkflowDocument()
    document.approve(docHandlerForm.getAnnotation());
    saveDocumentActionMessage("general.routing.approved", request);
    LOG.info("forwarding to actionTaken from approve()");
    return mapping.findForward("actionTaken");
}
```

Set up the **WorkflowDocument** in the **initializeBaseFormState()** method of the **DispatchActionBase** from which the Struts action classes inherit. Obtain the workflow document with this line of code:

```
String principalId = getUserSession(request).getPrincipalId();
WorkflowDocument document = new WorkflowDocument(principalId, docId);
```

Package the Web Application

Package the Client Application (client web application) for deployment the way you normally package web applications. The Travel Request Sample Web Application does this with an Ant build script. The *dist* step of the *build.xml* script builds the *SampleWorkflowClient.war* file.

Final Steps

Deploy the Plugin

Deploy the plugin to your workflow installation. Copy the plugin directory structure to your application plugins directory. Please see the Workflow Plugin Guide for more information.

Deploy the Client Web Application

Deploy the Client Web Application to your Application server the way you normally deploy web applications.

Chapter 9. KEW Configuration

KEW Integration Options

The following integration options are available to applications integrating with KEW:

- **Embedded** - The KEW engine is embedded into a Java application. The Standalone Rice Server is required.
- **Bundled** - Same as Embedded mode except that the entire KEW web application is also embedded into the Java application. The Standalone Rice Server is not required.
- **Remote Java Client** – A Java client is used which relies on the service bus to communicate with a Standalone Rice Server's KEW services.
- **Thin Java Client** - A thin Java client is used which communicates with a Standalone Rice Server over remote service calls.
- **Web Services** - Interacts directly with web services on a Standalone Rice Server.

Table 9.1. Advantages/Disadvantages of KEW Integration Options

Integration Option	Advantages	Disadvantages
Embedded	<ul style="list-style-type: none"> • Integration of database transactions between client application and embedded KEW (via JTA) • Performance - Embedded client talks directly to database • No need for application plug-ins on the server • Great for Enterprise deployment, there is still a single shared Standalone Rice web application but scalability is increased because of multiple Workflow Engines 	<ul style="list-style-type: none"> • Can only be used by Java clients • More library dependencies than the Thin Client model • Requires client application to establish connections to Kuala Rice database
Bundled	<ul style="list-style-type: none"> • All the advantages of Embedded Mode • No need to deploy a standalone Rice server • Ideal for development or "quickstart" applications • Application can be bundled with Rice for ease of development/distribution • Can switch to Embedded Mode for deployment in an Enterprise environment 	<ul style="list-style-type: none"> • Not desirable for Enterprise deployment where more than one application is integrated with Rice and KEW • More library dependencies than the Thin Client model and Embedded Mode (additional web libraries)
Remote Java Client	<ul style="list-style-type: none"> • Relatively simple configuration • Client can access more external KEW services from the Standalone Rice Server than the Thin Java Client, and yet the client does not need to have an embedded KEW engine 	<ul style="list-style-type: none"> • Requires client application to be KSB-enabled, unlike the Thin Java Client • Cannot be used by KNS-enabled client applications
Thin Java Client	<ul style="list-style-type: none"> • Relatively simple configuration • Fewer Library Dependencies 	<ul style="list-style-type: none"> • No transactional integration between client and server • Plug-ins must be deployed to the server if custom routing components are needed
Web Services	<ul style="list-style-type: none"> • Any language which supports web services can be used 	<ul style="list-style-type: none"> • No transactional integration between client and server • Plug-ins must be deployed to the server if custom routing components are needed

Integration Option	Advantages	Disadvantages
		<ul style="list-style-type: none"> • Web Services can be slower than other integration options

Standalone Server

To effectively use any of the KEW integration modes besides bundled, a Standalone Rice Server will need to be deployed.

Embedded Deployment Diagram

Here is a diagram illustrating what a sample embedded deployment might look look.

Figure 9.1. Embedded Deployment Diagram example



Bundling the KEW Application

web.xml

Bundled mode is the same as *embedded* mode except that the client application embeds the entire Kuali Rice system within it (including the web application). The embedding of the web application portion is accomplished by utilizing Struts Modules.

Configuration is the same as *embedded* mode, with the exception of loading the web application portions in the **web.xml**:

```
<filter>
  <filter-name>UserLoginFilter</filter-name>
  <filter-class>org.kuali.rice.krad.web.filter.UserLoginFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>UserLoginFilter</filter-name>
  <servlet-name>action</servlet-name>
</filter-mapping>

<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  .. other struts configuration if applicable
  <init-param>
    <param-name>config/en</param-name>
    <param-value>/en/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <load-on-startup>0</load-on-startup>
</servlet>

<servlet>
  <servlet-name>remoting</servlet-name>
  <servlet-class>org.kuali.rice.kws.messaging.servlet.KSBDispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet>
  <servlet-name>export</servlet-name>
  <servlet-class>org.kuali.rice.kew.export.web.ExportServlet</servlet-class>
</servlet>

<servlet>
  <servlet-name>attachment</servlet-name>
  <servlet-class>org.kuali.rice.kew.notes.web.AttachmentServlet</servlet-class>
</servlet>

<servlet>
  <servlet-name>edoclite</servlet-name>
  <servlet-class>org.kuali.rice.kew.edl.EDLServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>remoting</servlet-name>
  <url-pattern>/remoting/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>

<servlet-mapping>

  <servlet-name>export</servlet-name>
  <url-pattern>/export/*</url-pattern>
</servlet-mapping>
```

```

<servlet-mapping>
    <servlet-name>attachment</servlet-name>
    <url-pattern>/en/attachment/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>edoclite</servlet-name>
    <url-pattern>/en/EDocLite</url-pattern>
</servlet-mapping>

```

org.kuali.rice.krad.web.filter.UserLoginFilter – This filter is used to assist the KEW bundled web application in determining who the authenticated user is. Specifically, the login filter invokes the KIM identity management service to determine the identity of the authenticated user.

Typically, a previously executed filter will challenge the user on entry to a Rice web page for their authentication credentials using CAS or some other form of single sign on (SSO) authentication system.

For development and testing purposes, Rice provides a simple filter implementation that will present a simple sign on screen. This screen displays only a single login entry field and submit button. The user can enter their username (no password) and press the submit button, and the system authenticates the user for entry into the system.

This can be configured as follows in the **web.xml**:

```

<filter>
    <filter-name>LoginFilter</filter-name>
    <filter-class>org.kuali.rice.krad.web.filter.UserLoginFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <servlet-name>action</servlet-name>
</filter-mapping>

```

and in the **rice-config.xml**:

```

<param name="filter.login.class">org.kuali.rice.krad.web.filter.DummyLoginFilter</param>
<param name="filtermapping.login.1"/*</param>

```

org.apache.struts.action.ActionServlet - The Struts servlet which loads the KEW Struts module. The module name should be 'en'. Struts only allows a single Action Servlet so if you are using Struts in your application, all of your Struts modules will need to be configured using the init-param elements in this servlet definition.

org.kuali.rice.ksb.messaging.servlet.KSBDispatcherServlet - A servlet which dispatches http requests for the Kuali Service Bus (see KSB documentation for more details). The servlet mapping here should correspond to the **serviceServletUrl** configuration parameter for the KSBConfigurer.

org.kuali.rice.kew.export.web.ExportServlet - serves exports of lookup results as XML files

org.kuali.rice.kew.notes.web.AttachmentServlet - serves attachments that have been attached to documents using the KEW Notes and Attachments framework

org.kuali.rice.kew.edl.EDLServlet - The servlet used to interact with eDocLite documents. See eDocLite documentation for more information.

Bundled Deployment Diagram

Figure 9.2. Bundled deployment diagram



Using the Remote Java Client

Along with the previous embedded configurations, KEW also allows for Remote Java Clients, which communicate with KEW services that are available on the service bus. Configuration of the remote client is similar to that of the embedded client, except that no embedded KEW engine gets set up; instead, the client relies on the service bus for accessing the KEW services of the Standalone Rice Server.

Caution

Limitations of Remote KEW Java Clients:

At present, KNS-enabled Java clients cannot be used as Remote KEW Java Clients.

Using the Thin Java Client

In addition to the embedded configurations discussed previously, KEW also provides a thin java client which can be used to talk directly to two KEW services exposed on the service bus.

These KEW services are:

- WorkflowDocumentService - provides methods for creating, loading, approving and querying documents

- `WorkflowUtilityService` - provides methods for querying for various pieces of information about the KEW system

Additionally, access to two KIM services is required, as Principal and Group information is needed to use many of the methods in the KEW services above.

These KIM services are:

- `kimIdentityService` - provides methods to query for Principal and Entity information
- `kimGroupService` - provides methods to query for Group information

Of course, this configuration requires Standalone Rice Server deployment. The workflow engine deployed within Standalone Rice Server is used for processing documents that integrate using a thin client.

These services are exposed on the KSB as Java services, meaning they use Java Serialization over HTTP to communicate. Optionally, the KEW services can also be secured to provide access to only those callers with authorized digital signatures (note that secure access is required for the KIM services). In order to configure the thin client, the following configuration properties need to be defined.

Required Thin Client Configuration Properties

Table 9.2. Required Thin Client Configuration Properties

Property	Description
<code>encryption.key</code>	The secret key used by the encryption service; Must match the setting on the standalone server
<code>keystore.alias</code>	Alias of the application's key within the keystore
<code>keystore.file</code>	Path to the application's keystore file
<code>keystore.password</code>	Password to the keystore and the key with the configured alias
<code>workflowdocument.jvaservice.endpoint</code>	Endpoint URL for the Workflow Document service
<code>workflowutility.jvaservice.endpoint</code>	Endpoint URL for the Workflow Utility service
<code>identity.jvaservice.endpoint</code>	Endpoint URL for the KIM identity service
<code>group.jvaservice.endpoint</code>	Endpoint URL for the KIM group service

Note

It is simplest to use an identical keystore file and configuration in your thin client application to that on your standalone server.

Optional Thin Client Configuration Properties

Table 9.3. Optional Thin Client Configuration Properties

Property	Description
<code>secure.workflowdocument.jvaservice.endpoint</code>	true/false value indicating if endpoint is secured (defaults to true); Must match the setting on the standalone server
<code>secure.workflowutility.jvaservice.endpoint</code>	true/false value indicating if endpoint is secured (defaults to true); Must match the setting on the standalone server

Thin Client Spring Configuration

Here is the Spring configuration for a thin client in `ThinClientSpring.xml`:

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
  <!-- point Rice to the file containing your configuration params -->
  <!-- which should include a parameter setting kew.mode to "THIN" -->
  <bean id="config" class="org.kuali.rice.core.config.spring.ConfigFactoryBean">
    <property name="configLocations">
      <list>
        <value>classpath: yourThinClientApp-config.xml</value>
      </list>
    </property>
  </bean>
  <!-- Pull your configuration params out as Properties -->
  <bean id="configProperties"
class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
    <property name="targetObject" ref="config" />
    <property name="targetMethod" value="getProperties" />
  </bean>
  <!-- expose configuration params to Spring -->
  <bean class=
"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="properties" ref="configProperties" />
  </bean>
  <!-- The RiceConfigurer that sets up thin client mode -->
  <bean id="rice" class="org.kuali.rice.kew.config.KEWConfigurer">
    <!-- inject the "config" bean into our configurer -->
    <property name="rootConfig" ref="config" />
  </bean>
</beans>
```

For more details on configuring Rice for a thin client, please see the Intro to Rice Guide in the Implementation Details section, the [Thin Client Implementation](#) sub-section.

Endpoint URLs

Since KEW and KIM use the KSB to expose their services, the endpoint URLs are the same as those exported by the KSB.

An example configuration for these might be:

```
<param name=
"workflowdocument.javaservice.endpoint">http://yourlocalip/kr-dev/remoting/WorkflowDocumentActionsService</
param>
<param name=
"workflowutility.javaservice.endpoint">http://yourlocalip/kr-dev/remoting/WorkflowUtilityService</param>
<param name=
"identity.javaservice.endpoint">http://yourlocalip/kr-dev/remoting/kimIdentityService</param>
<param name=
"group.javaservice.endpoint">http://yourlocalip/kr-dev/remoting/kimGroupService</param>
```

Thin Client Deployment Diagram

Here is a diagram showing what a thin client deployment might look like.

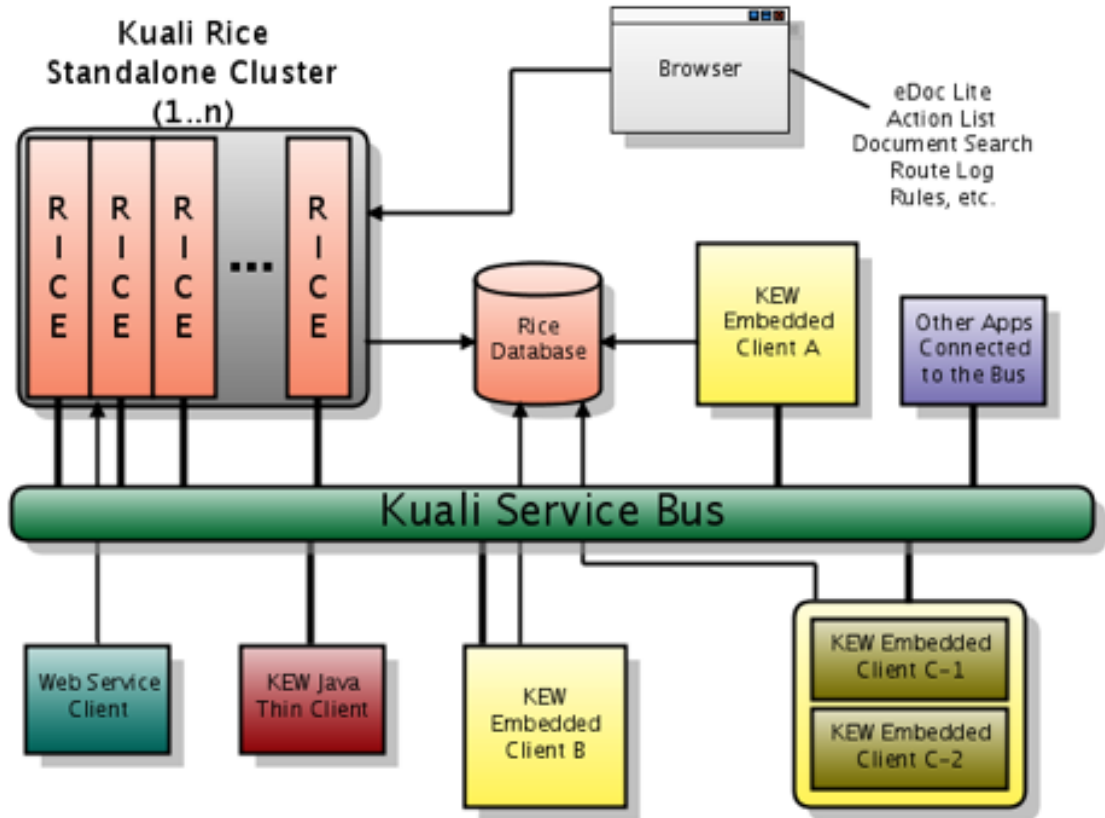
Figure 9.3. Thin client deployment diagram

Picture of an Enterprise Deployment

As can be seen from the various integration options described, a KEW Enterprise Deployment (and Kuali Rice in general) might very well be a distributed environment with multiple systems communicating with each other.

The diagram below shows what a typical Enterprise deployment of Kuali Rice might look like.

Figure 9.4. Typical enterprise deployment of Kuali Rice



KEW Core Parameters

The display below includes those basic set of parameters for **rice-config.xml** as the minimal parameters to startup the Rice software. These parameters are a beginning reference to you for modification to your **rice-config.xml**.

Note

Please verify that your application.url and database username/password are set correctly.

Table 9.4. KEW Core Parameters

Core	Description	Examples\Values
workflow.url	URL to the KEW web module (i.e., \${application.url}/en)	
plugin.dir	Directory from which plugins will be loaded	
attachment.dir.location	Directory where attachments will be stored	

As a minimum, you must enable the dummy login filter by adding these lines to the rice-config.xml file for default login screen:

```
<param name="filter.login.class">org.kuali.rice.krad.web.filter.DummyLoginFilter</param>
<param name="filtermapping.login.1">*/</param>
```


KEW Configuration Properties

Table 9.5. KEW Configuration Properties

Property	Description	Default
actionlist.outbox	Determines if the KEW actionlist "outbox" (i.e., the actions already completed) will be viewable by users of the Rice application.	false
actionlist.outbox.default.preference.on	Determines if the KEW actionlist "outbox" is the default mode for viewing the action list.	false
base.url	Base URL under which Action List and other KEW screens can be found	Example: if your action list URL is http://yourlocalip/en/ActionList.do , set this property to http://yourlocalip/
client.protocol	Same as clientProtocol property on KEWConfigurer, this property can be configured in either place	embedded
data.xml.root.location	The temporary location of files being processed by the KEW XmlPollingService	/tmp/\${environment}/kew/xml
document.lock.timeout	Used by the Oracle database platform to determine how long database locks on the document header are used	
email.reminder.lifecycle.enabled	If true, turns on timed job to send out regular e-mails to remind users of actions still waiting in their action list	
extra.classes.dir	Directory where classes for KEW plugins are located	
extra.lib.dir	Directory where libraries for KEW plugins are located	
kew.mode	The mode that KEW will run in; choices are "local", "embedded", "remote", or "thin"	local
kew.url	The base URL of KEW services and pages	\${application.url}/kew
plugin.dir	Directory to load plugins from if the Plugin Registry is enabled	
plugin.registry.enabled	If set to true, then the Plugin Registry will be enabled and any available plugins will be loaded (see Workflow Plugin Guide)	false
attachment.dir.location	When using the attachments system, this is the directory where attachments will be stored	
data.xml.loaded.location	Directory path where the XML Loader will store successfully loaded XML files	
data.xml.pending.location	Directory path where the XML Loader will look for files to ingest	
data.xml.pollIntervalSecs	Interval in seconds that the XML Loader will poll the pending directory for new XML files to load	
data.xml.problem.location	Directory path where the XML Loader will put XML files it failed to load	
datasource.platform	The fully qualified class name of an implementation of the org.kuali.rice.core.database.platform.Platform interface	
default.note.class	The fully qualified class name of the default implementation of org.kuali.rice.kew.notes.CustomNoteAttribute to use for the Notes system	org.kuali.rice.kew.notes.CustomNoteAttributeImpl
edl.config.loc	Location to load the EDocLite component configuration from	classpath:META-INF/EDLConfig.xml
embedded.server	Indicates if an embedded instance is supposed to behave like a standalone server. See additional notes below under embedded.server	false
Identity.useRemoteServices	Configuration parameter that governs whether a number of common identity services (user and	

Property	Description	Default
	group service) are exported or retrieved via the bus. If this flag is set to true then: 1. user and group service will NOT be published the bus, and 2. CoreResourceLoader will short-circuit the resource loader stack lookup and go directly to the bus to obtain these services, circumventing any beans that may be defined by local modules.	
initialDelaySecs	Delay in seconds after system starts up to begin the XML Loader polling	
rice.kew.enableKENNotification	Determines if KCB notifications should be sent for KEW events when Action Item events occur	true
rice.kew.struts.config.files	The struts-config.xml configuration file that the KEW portion of the Rice application will use	/kew/WEB-INF/struts-config.xml
workflow.documentsearch.base.url	The URL for the document search page	`\${workflow.url}/DocumentSearch.do?docFormKey=8888888&returnLocation=\${application.url}/portal.do&hideReturnLink=true`
xml.pipeline.lifecycle.enabled	If set to true, will poll a directory for new Rice configuration XML and ingest any new XML placed in that directory	false

The 'embedded.server' Parameter

If embedded.server parameter is enabled (set to true), then two additional features will be loaded when KEW is started:

1. XML Loader
2. Email Reminders

The XML Loader will poll a directory for XML files to ingest into the system (as configured by the data.xml.* properties).

The Email Reminders will handle sending Daily and Weekly batch emails for users that have their preferences set accordingly.

The 'datasource.platform' Parameter

KEW requires and uses the database platform implementation in order to function. These may be implemented differently for each support database management system.

The current functional implementations of this platform are:

- org.kuali.rice.core.framework.persistence.platform.OraclePlatform
- org.kuali.rice.core.framework.persistence.platform.Oracle9iPlatform (deprecated and just an alias for the OraclePlatform)
- org.kuali.rice.core.framework.persistence.platform.MySQLDatabasePlatform

Custom Servlet Filters

When running a Standalone Rice Server, you may want to implement your own filters for authentication purposes. The system comes with a special filter that will read filter definitions and mappings from the configuration system.

The Bootstrap Filter is a generic filter that is applied to all web requests, which then delegates to any filters and are setup through the default configuration. This mechanism allows registration of institution-specific filters without the necessity of modifying the web application configuration file (/WEB-INF/web.xml) within the standalone webapp.

Filter syntax is as follows:

```
<param name="filter.filter name.class">class name of filter</param>
```

filter name is an arbitrary name for your filter:

```
<param name="filter.myfilter.class">edu.institution.organization.MyFilter</param>
```

Any number of configuration parameters may be defined for a given filter as follows:

```
<param name="filter.filter name.filter param name">filter param value</param>
```

For example:

```
<param name="filter.myfilter.color">red</param>
<param name="filter.myfilter.shape">square</param>
```

For custom filters to be invoked, they must first be mapped to requests. That is done via the filter mapping parameter:

```
<param name="filtermapping.filter name.optional order index">path matching expression</param>
```

filter name is the name of your previously defined filter, *optional order index* is an optional integer used to specify the position of the filter in the invocation order, and *path matching expression* is a Servlet-specification-compatible url pattern.

```
<param name="filtermapping.myfilter.1">/special/path/</param>
```

If an order index is not specified, it is assumed to be 0. Filters with equivalent order are ordered arbitrarily with relation to each other (not in order of filter or mapping definition). A full example follows:

```
<param name="filter.myfilter.class">edu.institution.organization.MyFilter</param>
<param name="filter.myfilter.color">red</param>
<param name="filter.myfilter.shape">square</param>
<param name="filter.securityfilter.class">edu.institution.organization.SecurityFilter</param>
<param name="filter.securityfilter.secretKey">abracadabra</param>
<param name="filter.compressionfilter.class">edu.institution.organization.CompressionFilter</param>
<param name="filter.compressionfilter.compressLevel">5</param>
<param name="filtermapping.securityfilter.1">/secure/</param>
<param name="filtermapping.myfilter.2">/special/path/</param>
```

```
<param name="filtermapping.compressionfilter.3">*/</param>
```

Email Configuration

KEW can send emails to notify users about items in their Action List (depending on user preferences). Email in KEW uses the JavaMail library. In order to configure email, you will need to configure the appropriate JavaMail properties. A list of those properties can be found at the end of the page at the following url: <http://java.sun.com/products/javamail/javadocs/javax/mail/package-summary.html>

In addition to these standard JavaMail properties, you can also set the following optional properties to configure simple SMTP authentication.

Table 9.6. Optional Properties to Configure Simple SMTP Authentication

Property	Description	Examples/Values
mail.transport.protocol	The protocol used to sending mail	smtp
mail.smtp.host	This is the host name of the SMTP	smtp.secureserver.net
mail.smtp.username	The username used for access to the SMTP server	
mail.smtp.password	The password used for access to the SMTP server	

Of course, if the authentication required by your mail server is beyond the abilities of the above configuration, it is possible to override the *enEmailService* loaded by the KEW module and implement a custom email service.

In order for KEW to send out emails, several steps need to be done. In order to have KEW send out any emails, the "SEND_EMAIL_NOTIFICATION_IND" KNS System Parameter needs to be set to 'Y'. For emails to real people, the environment code must be set to 'prd'. If this is not set to 'prd', an email can still be sent out to a test address. This test address is set by the KNS System Parameter, "EMAIL_NOTIFICATION_TEST_ADDRESS". Emails sent in a test system will only be sent to the address specified by the EMAIL_NOTIFICATION_TEST_ADDRESS. The "from" address may also be set with a System Parameter. To do this, set the "FROM_ADDRESS" System Parameter to the email address you want the KEW emails sent from. If the FROM_ADDRESS parameter doesn't exist or isn't set, it will default to "admin@localhost".

Periodic Email Reminders

KEW can send emails on a nightly or weekly basis to remind users about items in their Action List (depending on user preferences). The following set of parameters configures whether the processes to send these reminders will run, and at what time(s) of day they will do so.

Table 9.7. Configuration Parameters for Email Reminders

Property	Description	Examples/Values
email.reminder.lifecycle.enabled	Enable periodic KEW reminder emails	true
dailyEmail.active	Enable daily reminder emails	true
dailyEmail.cronExpression	Configures the schedule on which the daily reminder emails are sent – see <code>org.quartz.CronExpression</code> , <code>org.quartz.CronTrigger</code> for information about the format for this parameter	0 0 1 * * ?
weeklyEmail.active	Enable weekly reminder emails	true

Property	Description	Examples/Values
weeklyEmail.cronExpression	Configures the schedule on which the weekly reminder emails are sent – see <code>org.quartz.CronExpression</code> , <code>org.quartz.CronTrigger</code> for information about the format for this parameter	0 0 2 ? * 2

Email Customization

KEW provides default email template for Action List notification messages that are sent. However, it is also possible to customize this either globally or on a Document Type by Document Type basis.

There are two ways to customize Action List emails:

1. Configure a `CustomEmailAttribute`
2. Creating a custom XSLT Stylesheet

To accomplish this, you must write a `CustomEmailAttribute` and configure it on the appropriate `DocumentType`.

Configure a CustomEmailAttribute

The `CustomEmailAttribute` interface provides two methods for adding custom content to both the subject and the body.

```
public String getCustomEmailSubject();
public String getCustomEmailBody();
```

Note that each of these values is appended to the end of either the subject or the body of the email. The rest of the email still uses the standard email content.

Also, when implementing one of these components, the document is made available to you as a `RouteHeaderDTO` and the action request related to the notification is made available as an `ActionRequestDTO`.

Once you have implemented the `CustomEmailAttribute`, you need to make it available to the KEW engine (either deployed in a plugin or available on the classpath when running embedded KEW).

Document Type Configuration

Once you make the email attribute component available to KEW, you need to configure it on the Document Type.

First, define the attribute:

```
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <ruleAttributes xmlns="ns:workflow/RuleAttribute" xsi:schemaLocation="ns:workflow/RuleAttribute
resource:RuleAttribute">
    <ruleAttribute>
      <name>MyCustomEmailAttribute</name>
      <className>my.package.MyCustomEmailAttribute</className>
      <label>MyCustomEmailAttribute</label>
```

```

        <description>My Custom Email Attribute</description>
        <type>EmailAttribute</type>
    </ruleAttribute>
</ruleAttributes>
</data>

```

Next, update the Document Type definition to include the attribute:

```

<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <documentTypes xmlns="ns:workflow/DocumentType" xsi:schemaLocation="ns:workflow/DocumentType
resource:DocumentType">
    <documentType>
      <name>MyDocType</name>
      <label>My Document Type</label>
      <postProcessorName>...</postProcessorName>
      <attributes>
        <attribute>
          <name>MyCustomEmailAttribute</name>
        </attribute>
      </attributes>
      <routePaths>
        ...
      </routePaths>
      <routeNodes>
        ...
      </routeNodes>
    </documentType>
  </documentTypes>
</data>

```

These should be ingested using the XML Ingester. See [Importing Files to KEW](#) for more information on using the XML Ingester.

Create a Custom XSLT Style Sheet

Global Email Customization

A more convenient way to customize email content declaratively is to replace the global email XSLT style sheet in Rice. Do this by ingesting an XSLT style sheet with the name *kew.email.style*. This style sheet should take input of this format for reminder emails:

```

<!-- root element sent depends on email content requested by the system -->
<immediateReminder|dailyReminder|weeklyReminder actionListUrl="url to ActionList" preferencesUrl="url to
Preferences"
applicationEmailAddress="configured KEW email address" env="KEW environment string (dev/test/prd)">

  <user> <!-- the principal who received the request -->
    <name>...</name>
    <principalName>...</principalName>
    <principalId>...</principalId>
    <firstName>...</firstName>
    <lastName>...</lastName>
    <emailAddress>...</emailAddress>
    ...
  </user>
  <actionItem>
    <!-- one top-level actionItem element sent for each ActionItem; for immediate email reminders, there
will only ever be one; for daily and weekly reminders, there may be several -->

    <!-- custom subject content produced by the CustomEmailAttribute associated with the DocumentType of
this ActionItem, if any -->
    <customSubject>...</customSubject>

    <!-- custom body content produced by the CustomEmailAttribute associated with the DocumentType of this
ActionItem, if any -->

```

```

<customBody>...</customBody>

<actionItem> <!-- the actual ActionItem -->
  <principalId>...</principalId>
  <groupId>...</groupId>
  <routeHeaderId>...</routeHeaderId>
  <actionRequestId>...</actionRequestId>
  <docTitle>...</docTitle>
  <actionItemId>...</actionItemId>
  <roleName>...</roleName>
  <dateAssigned>...</dateAssigned>
  <actionRequestCd>...</actionRequestCd>
  <docHandlerURL>...</docHandlerURL>
  <recipientTypeCode>...</recipientTypeCode>
  <actionRequestLabel>...</actionRequestLabel>
  <delegationType>...</delegationType>
  <docName>...</docName>
  <docLabel>...</docLabel>
</actionItem>
<actionItemPerson> <!-- see "user" element at the top, simliar content -->
...
</actionItemPerson>
<actionItemPrincipalId>...</actionItemPrincipalId>
<actionItemPrincipalName>...</actionItemPrincipalName>

<doc> <!-- the RouteHeader associated with this ActionItem -->
  <routeHeaderId>...</routeHeaderId>
  <docTitle>...</docTitle>
  <docContent>...</docContent>
  <initiatorWorkflowId>...</initiatorWorkflowId>
  <documentTypeId>...</documentTypeId>
  <docRouteStatusLabel>...</docRouteStatusLabel>
  <docRouteStatus>...</docRouteStatus>
  <createDate>...</createDate>
  ...
</doc>
<docInitiator>
  <principalName>...</principalName>
  <principalId>...</principalId>
  <entityId>...</entityId>
</docInitiator>
<documentType> <!-- DocumentType -->
  <name>...</name>
  <label>...</label>
  <description>...</description>
  <serviceNamespace>...</serviceNamespace>
  <notificationFromAddress>...</notificationFromAddress>
  <docHandlerUrl>...</docHandlerUrl>
  <documentTypeId>...</documentTypeId>
  ...
</actionItem>

</immediateReminder|dailyReminder|weeklyReminder>

```

This format is used for feedback emails:

```

<!-- feedback form -->
<feedback actionListUrl="url to ActionList" preferencesUrl="url to Preferences"
applicationEmailAddress="configured KEW email address" env="KEW environment string (dev/test/prd)">
  <networkId>...</networkId>
  <lastName>...</lastName>
  <routeHeaderId>...</routeHeaderId>
  <documentType>...</documentType>
  <userEmail>...</userEmail>
  <phone>...</phone>
  <timeDate>...</timeDate>
  <edenCategory>...</edenCategory>
  <comments>
    ...
  </comments>
  <pageUrl>...</pageUrl>
  <firstName>...</firstName>
  <exception>...</exception>
  <userName>...</userName>

```

```
</feedback>
```

In both cases, the output generated by the style sheet must be like this:

```
<email>
  <subject>... subject here ...</subject>
  <body>... body here ...</body>
</email>
```

You must then upload the custom style sheet into the style service using the standard KEW XML ingestion mechanism:

```
<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
<styles xmlns="ns:workflow/Style" xsi:schemaLocation="ns:workflow/Style resource:Style">
  <style name="kew.email.style">
    <!-- A custom global email reminder stylesheet -->
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:strip-space elements="*" />
      <xsl:template match="immediateReminder">
        ...
      </xsl:template>
      <xsl:template match="dailyReminder">
        ...
      </xsl:template>
      <xsl:template match="weeklyReminder">
        ...
      </xsl:template>
      <xsl:template match="feedback">
        ...
      </xsl:template>
    </xsl:stylesheet>
  </style>
</styles>
```

The global style sheet should handle all email content requests. You can use the standard *include* syntax to import an existing style sheet that may implement defaults.

DocumentType-Specific Email Customization

You can also customize immediate reminder email content on a per-DocumentType basis. To do so, define a custom email style sheet name on the DocumentType definition:

```
...
<documentType>
  <name>SomeDoc</name>
  <description>a document with customized reminder email</description>
  ...
  <emailStylesheet>somedoc.custom.email.style</emailStylesheet>
  ...
</documentType>
...
```

Then, upload a corresponding style sheet with a matching name, as above.

Workflow Preferences Configuration

Workflow users have the ability to update their preferences by going to the "User Preferences" page. The default values for many of these preferences can now be configured.

For example, institutions will commonly override the default action list email preference. By default it's set to "immediate," but it can be configured to "no", "daily", "weekly", or "immediate." The user will still be able to override the defaults on their User Preferences screen.

Here a list of workflow preferences that can be configured:

```
<!-- Default Option for Action List User Preferences. -->
<param name="userOptions.default.color">white</param>
<!-- email options: no, daily, weekly, immediate -->
<param name="userOptions.default.email" >immediate</param>
<param name="userOptions.default.notifyPrimary" >yes</param>
<param name="userOptions.default.notifySecondary" >no</param>
<param name="userOptions.default.openNewWindow" >yes</param>
<param name="userOptions.default.actionListSize" >10</param>
<param name="userOptions.default.refreshRate" >15</param>
<param name="userOptions.default.showActionRequired" >yes</param>

<param name="userOptions.default.showDateCreated" >yes</param>
<param name="userOptions.default.showDocumentType" >yes</param>
<param name="userOptions.default.showDocumentStatus" >yes</param>
<param name="userOptions.default.showInitiator" >yes</param>
<param name="userOptions.default.showDelegator" >yes</param>
<param name="userOptions.default.showTitle" >yes</param>
<param name="userOptions.default.showWorkgroupRequest" >yes</param>
<param name="userOptions.default.showClearFYI" >yes</param>
<param name="userOptions.default.showLastApprovedDate" >no</param>
<param name="userOptions.default.showCurrentNode" >no</param>
<param name="userOptions.default.useOutBox" >yes</param>
<!-- delegatorFilterOnActionList: "Secondary Delegators on Action List Page" or "Secondary Delegators only on
Filter Page" -->
<param name="userOptions.default.delegatorFilterOnActionList" >Secondary Delegators on Action List Page</param>
<param name="userOptions.default.primaryDelegatorFilterOnActionList" >Primary Delegates on Action List Page</
param>
```

Outbox Configuration

The **Outbox** is a standard feature on the **Action List** and is visible to the user in the UI by default. When the Outbox is turned on, users can access it from the Outbox hyperlink at the top of the Action List.

The Outbox is implemented by heavily leveraging existing Action List code. When an **Action Item** is deleted from the Action Item table as the result of a user action, the item is stored in the **KEW_OUT_BOX_ITM_T** table, using the **org.kuali.rice.kew.actionitem.OutboxItemActionListExtension** object. This object is an extension of the **ActionItemActionListExtension**. The separate object exists to provide a bean for OJB mapping.

The Workflow Preferences determine if the Outbox is visible and functioning for each user. The preference is called **Use Outbox**. In addition, you can configure the Outbox at the KEW level using the parameter tag:

```
<param name="actionlist.outbox">true</param>
```

When the Outbox is set to *false*, the preference for individual users to configure the Outbox is turned off. By default, the Outbox is set to true at the KEW level. You can turn the Outbox off (to hide it from users) by setting the property below to *false*:

```
<param name="actionlist.outbox.default.preference.on">false</param>
```

This provides backwards compatibility with applications that used earlier versions of KEW.

Notes on the Outbox:

- Actions on saved documents are not displayed in the Outbox.
- The Outbox responds to all saved Filters and Action List Preferences.
- A unique instance of a document only exists in the Outbox. If a user has a document in the Outbox and that user takes action on the document, then the original instance of that document remains in the Outbox.

Implementing KEW at your institution

In addition to the previous discussion of KEW configuration, there are a few other aspects relevant to implementing KEW at your institution.

Bootstrap data

Because the operation of parts of KEW is dependent on a set of Document Types and Attributes being available within the system, there is some bootstrap XML that you will want to import. The easiest way to do this is to import the files in the following locations using the XML Ingester:

- `kns/src/main/config/xml/RiceSampleAppWorkflowBootstrap.xml`
- `kew/src/main/config/bootstrap/edlstyle.xml`
- `kew/src/main/config/bootstrap/widgets.xml`

These files include the following:

- Application constants: cluster-wide configuration settings
- Core document types and rules: a few primordial document types and rules are required for the system to function
- Default "eDocLite" styles: these are required if you wish to use eDocLite
- Default admin user and workgroup: these are depended upon (at the moment) by the core document types and rules, as well as referred to by the default application constants

Application constants you may want to change:

- `Config.Application.AdminUserList`: this should be set to a space-delimited set of administrative user names
- `Workflow.AdminWorkgroup`: this should be set to an institutional admin workgroup; if the default KEW workgroup service is used, this can be left to the default, `WorkflowAdmin`
- `Config.Mailer.FromAddress`: this should be changed to an address specific to your institution, e.g. `kew@your-university.edu`
- `HelpDeskActionList.helpDeskActionListName`: set to an workgroup at your institution
- `ApplicationContext`: set to the context path of the KEW application, if it differs from the environment default, e.g. "en-prod" instead of "en-prd"

In the core document types and rules config, you will need to change:

- `superUserWorkgroupName`, `blanketApproveWorkgroupName`, and `exceptionWorkgroup`: should be set to the administrative group at your institution. If you are using the default workgroup service, this can be left as `WorkgroupAdmin`
- ensure all `docHandler` elements, if they specify a URL, specify: `"${base.url}/en-dev/Workgroup.do?methodToCall=docHandler"`, and ensure that the `base.url` config parameter is specified in your configuration (as mentioned above)

KEW Routing Components and Configuration Guide

KEW has several components that you can use to configure routing. Typically a single application will write a set of these components for reuse across multiple Document Types. These components are wired together using an XML configuration file that you need to import into KEW. See [Importing XML Files to KEW](#) for more information.

This document looks at defining the routing components available in KEW and how to use these components to make a cohesive routing setup.

- **RouteModule** - The most basic module; it allows KEW to generate Action Requests
- **RuleAttribute** - A component that fits into KEW's rule system. These rules are used to build routing paths for documents. They function for users across the organization and for multiple applications.
- **XML RuleAttribute** – Similar in functionality to a `RuleAttribute` but built using XML only
- **RoleAttribute** - A component that fits into KEW's rule system, but which is a pointer to outside data. See [Built-in Roles and Nodes](#) for more information on implementing a `RoleAttribute`.
- **PostProcessor** - A component that gets called throughout the routing process and handles a set of standard events that all eDocs (electronic documents) go through.
- **DocumentType Authorizer** - A component that gets called during the routing process to perform authorization checks. Applications can customize this component on a per-doctype basis.

These components are contained in a Document Type that is defined in XML. A Document Type is the prototype for eDocs. Below is the Document Type configuration that explains how KEW uses the eDoc rule:

```
<?XML version="1.0" encoding="UTF-8"?>
<data XMLns="ns:workflow" XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <documentTypes XMLns="ns:workflow/DocumentType" xsi:schemaLocation="ns:workflow/DocumentType
resource:DocumentType">
    <documentType>
      <name>YOURSERVICE-DOCS.RuleDocument</name>
      <parent>YOURSERVICE-DOCS</parent>
      <description>Add/Modify Workflow rules</description>
      <label>Add/Modify Workflow rules</label>
      <postProcessorName>your.package.routetemplate.RulePostProcessor</postProcessorName>
      <authorizer>your.package.CustomDocumentTypeAuthorizer</authorizer>
      <superUserGroupName>WorkflowAdmin</superUserGroupName>
      <blanketApproveGroupName>IU-WORKFLOW-RULE-BLANKET-APPROVERS</blanketApproveGroupName>
      <defaultExceptionGroupName>YOUR_EXCEPTION_TEAM</defaultExceptionGroupName>
      <docHandler>https://yourlocalIP/en-prd/Rule.do?methodToCall=docHandler</docHandler>
      <notificationFromAddress>...@yourEmailServerIP.edu</notificationFromAddress>
      <active>true</active>
      <routingVersion>1</routingVersion>
    </documentType>
  </documentTypes>
</data>
```

```

<routePaths>
  <routePath>
    <start name="Adhoc Routing" nextNode="Rule routing Route Level" />
    <requests name="Rule routing Route Level" />
  </routePath>
</routePaths>
<routeNodes>
  <start name="Adhoc Routing">
    <activationType>S</activationType>
    <mandatoryRoute>>false</mandatoryRoute>
    <finalApproval>>false</finalApproval>
  </start>
  <requests name="Rule routing Route Level">
    <activationType>S</activationType>
    <ruleTemplate>RuleRoutingTemplate</ruleTemplate>
    <mandatoryRoute>>true</mandatoryRoute>
    <finalApproval>>false</finalApproval>
  </requests>
</routeNodes>
</documentType>
</documentTypes>
</data>

```

Configuration Steps

Let's go through the configuration step-by-step and explain what all the pieces mean:

DocumentTypeName Definition

```

<name>YOURSERVICE-DOCS.RuleDocument</name>
<parent>YOURSERVICE-DOCS</parent>
<description>Add/Modify Workflow rules</description>
<label>Add/Modify Workflow rules</label>

```

The section above defines the Document Type's name, its **parent**, **description**, and **label**. The **name** is used by the client application's API to communicate with KEW. Here is a sample of code from the client application's API communicating with KEW:

```

WorkflowDocument document = new WorkflowDocument(new NetworkIdVO("username"), "DocumentTypeName");
document.routeDocument("user inputted annotation");

```

The above code will route a document in KEW.

- The string **DocumentTypeName** exists in KEW and you define it using the **<name>** element.
- The **parent** element gives the Document Type a parent Document Type. Use this for inheritance of routing configuration and policies.
- **Description** is defined as shown. The document's *Description* is displayed on the Document Type report.
- **Label** is typically the forward-facing name for the Document Type. The label is displayed to the user when an eDoc is in their Action List and they use it when they search for an eDoc using DocSearch.

PostProcessor Class

```

<postProcessorName>your.package.routetemplate.RulePostProcessor</postProcessorName>

```

The element above determines which class to use for the PostProcessor for this particular Document Type. This component receives event notifications as eDocs travel through routing.

DocumentTypeAuthorizer Class

```
<authorizer>your.package.CustomDocumentTypeAuthorizer</authorizer>
```

The element above determines which class to use for the DocumentType Authorizer for this particular Document Type. This component performs authorization checks as the eDoc travels through routing.

Managed Workgroups

```
<superUserWorkgroupName>WorkflowAdmin</superUserWorkgroupName>
<blanketApproveWorkgroupName>WorkgroupBlanketApprovers</blanketApproveWorkgroupName>
<defaultExceptionWorkgroupName>WorkflowAdmin</defaultExceptionWorkgroupName>
```

This section sets KEW managed workgroups in several roles in the Document Type.

- **SuperUserWorkgroupName** defines the workgroup that determines whether a person is allowed to take Super User Actions on a document through the Super User interface.
- The content of element **blanketApproveWorkgroupName** determines which people have access to blanket approve a document.
- **defaultExceptionWorkgroup** determines to which workgroup to send an eDoc of this type if it goes into exception routing. This is an optional element. You can also define Exception Workgroups with a route node.

docHandler

```
<docHandler>https://yourlocalIP/en-prd/Rule.do?methodToCall=docHandler</docHandler>
```

The docHandler tells KEW where to forward users when they click an eDoc link. See Document Search for more information.

notificationFromAddress

```
<notificationFromAddress>...@yourEmailServerIP</notificationFromAddress>
```

When KEW sends an email notification to a user regarding a document of this type, the From address on the message is the address specified here. This is helpful because users will often reply to the messages they receive from KEW, and this allows their responses to go to an appropriate address for the Document Type. This is an optional element. If it is not defined here, KEW uses the default From address. See the Installation Guide for more detail.

active

```
<active>true</active>
```

Use active to define the activeness of a Document Type. KEW does not allow anyone to create eDocs of an inactive Document Type.

routePaths

```
<routePaths>
  <routePath>
    <start name="Adhoc Routing" nextNode="Rule routing Route Level" />
    <requests name="Rule routing Route Level" />
  </routePath>
</routePaths>
```

The above defines the path an eDoc will travel as it progresses through its life. **Start** and **Requests** are some of the standard node types used. There is only one stop each eDoc must make as it travels through workflow. The eDoc starts at the step **Adhoc Routing** and then progresses to the request node named **Rule routing Route Level**.

Additionally, an automatic progression of the application document status may be configured to occur on route node transition with the addition of the nextAppDocStatus attribute in the elements of a routePath:

```
<routePaths>
  <routePath>
    <start name="Initiated" nextNode="DestinationApproval" nextAppDocStatus="Approval in Progress"/>
    <requests name="DestinationApproval" nextNode="TravelerApproval" nextAppDocStatus="Submitted"/>
    <requests name="TravelerApproval" nextNode="SupervisorApproval" />
    <requests name="SupervisorApproval" nextNode="AccountApproval" />
    <requests name="AccountApproval" />
  </routePath>
</routePaths>
```

This section only defines the path the eDocs will travel, and optionally the application document status transitions. The nodes themselves are defined below.

Node Definition XML

```
<routeNodes>
  <start name="Adhoc Routing">
    <activationType>S</activationType>
    <mandatoryRoute>>false</mandatoryRoute>
    <finalApproval>>false</finalApproval>
  </start>
  <requests name="Rule routing Route Level">
    <activationType>S</activationType>
    <ruleTemplate>RuleRoutingTemplate</ruleTemplate>
    <mandatoryRoute>>true</mandatoryRoute>
    <finalApproval>>false</finalApproval>
  </requests>
</routeNodes>
```

This is the node definition XML. This determines certain behaviors each node can have.

Activation Type determines if Approve requests are activated all at once or one at a time. Any given requests node can generate multiple rules that can then generate multiple requests. The ActivationType value specifies if *all* action requests generated for *all* fired rules are activated immediately (*P = parallel* activation), or if the set of action requests generated by each rule are activated one after the other, according to rule order (*S = sequential* activation). However, to activate requests starting with those with the smallest priority *and* to activate all those requests in parallel the activation type of (*R = priority-parallel* activation). Once all requests are approved, then the next priority will be activated. This is essentially a hybrid of the

traditional sequential and parallel activation types. Activation type is only relevant when multiple rules are generated.

Figure 9.5. Parallel and Sequential Activation Types

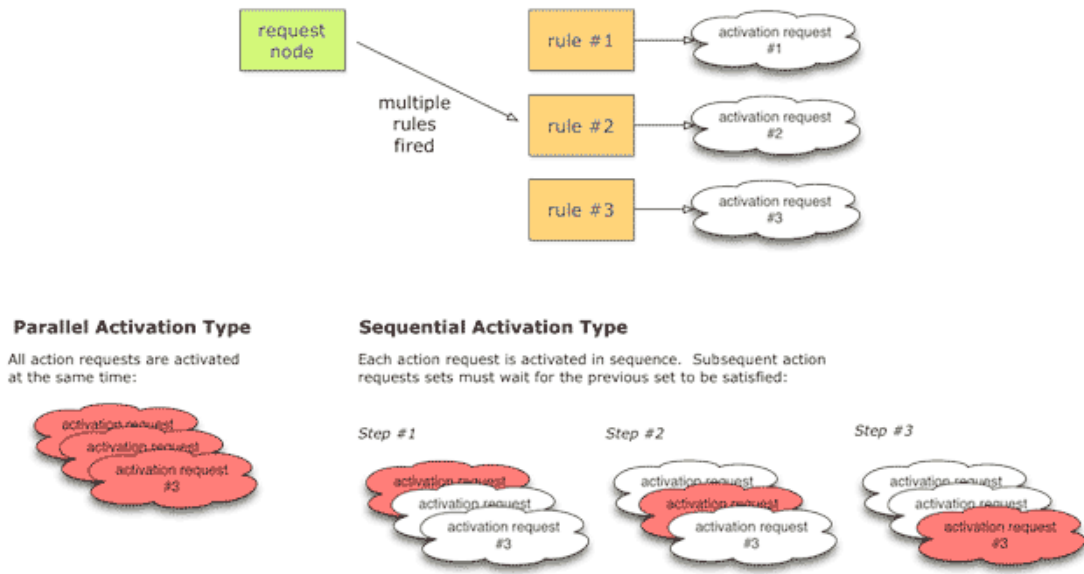
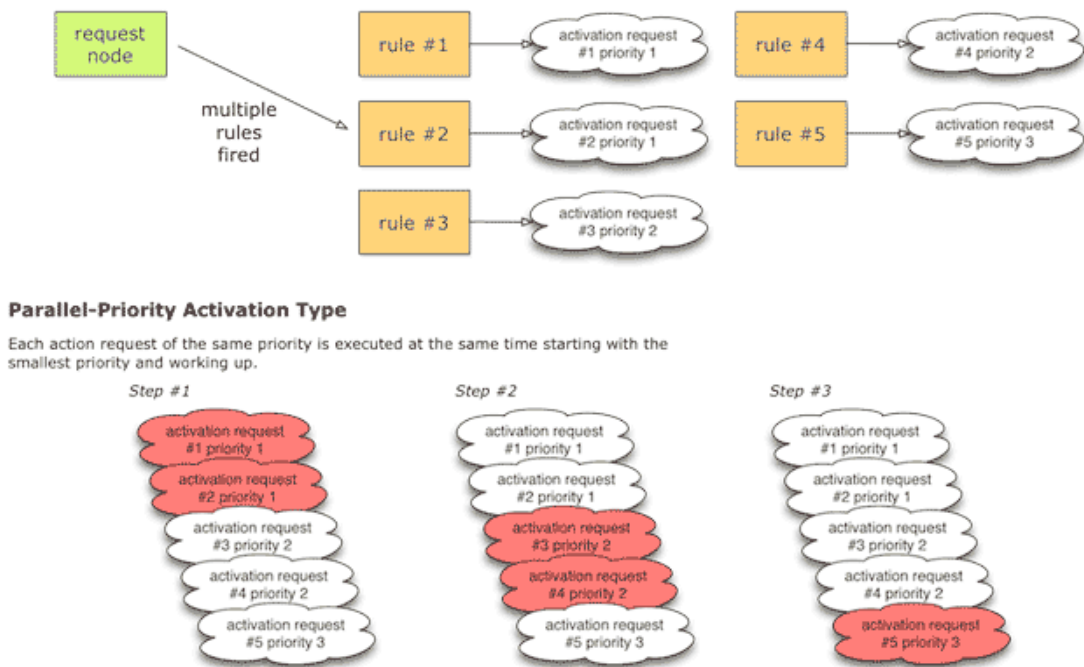


Figure 9.6. Parallel-Priority Activation Type



The **mandatoryRoute** key determines if it's mandatory to generate approval requests. If a route node is *mandatory* and it doesn't generate an *approve* request, the document is put in exception routing.

The **finalapproval** key determines if this node should be the last node that has an *approve* request. If approvals are generated after this step, the document is thrown into exception routing.

Finally, there is a request node named *Rule routing Route Level* with a key called **ruleTemplate**. This is our hook into the rule system for KEW:

```
<ruleTemplate>RuleRoutingTemplate</ruleTemplate>
```

And this is our hook into a route module:

```
<routeModule>package.your.ARouteModule</routeModule>
```

KEW contacts the route module when the document enters that route node and the route module returns Action Requests for KEW to deliver.

Rule Attributes

If the application integrating with KEW is using Rules to contain the routing data and **RuleAttributes** for document evaluation, then the routing configuration requires more XML. Below is an XML snippet that defines **RuleAttribute**; this is written in Java.

```
<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <ruleAttributes xmlns="ns:workflow/RuleAttribute" xsi:schemaLocation="ns:workflow/RuleAttribute
resource:RuleAttribute">
    <ruleAttribute>
      <name>RuleRoutingAttribute</name>
      <className>org.kuali.rice.kew.rule.RuleRoutingAttribute</className>
      <label>RuleRoutingAttribute</label>
      <description>RuleRoutingAttribute</description>
      <type>RuleAttribute</type>
    </ruleAttribute>
  </ruleAttributes>
</data>
```

The above defines a **RuleAttribute** called *RuleRoutingAttribute*. *RuleRoutingAttribute* maps to the Java class **org.kuali.rice.kew.rule.RuleRoutingAttribute**. The *type* of this attribute is a **RuleAttribute**; essentially this means the RuleAttribute's behavior is determined in a Java class. There are also RuleAttributes made entirely from XML, but programming attributes is outside the scope of this Guide.

Rule Templates

Finally, we need to tie the **RuleAttribute** to the Document Type. This is done using the **RuleTemplate** and it is defined using XML. The **RuleTemplate** schema below provides further explanation:

```
<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <ruleTemplates xmlns="ns:workflow/RuleTemplate" xsi:schemaLocation="ns:workflow/RuleTemplate
resource:RuleTemplate">
    <ruleTemplate>
      <name>RuleRoutingTemplate</name>
      <description>RuleRoutingTemplate</description>
      <attributes>
        <attribute>
          <name>RuleRoutingAttribute</name>
          <required>true</required>
        </attribute>
      </attributes>
    </ruleTemplate>
  </ruleTemplates>
</data>
```



```
</data>
```

Note

Notice that the name of this RuleTemplate, *RuleRoutingTemplate*, matches the name given in the **ruleTemplate** element in the Document Type route node declaration. Also, notice that the **RuleAttribute** made above is referenced in the **RuleTemplate** above in the *attributes* section.

```
<attributes>
  <attribute>
    <name>RuleRoutingAttribute</name>
    <required>true</required>
  </attribute>
</attributes>
```

The **RuleTemplate** is the join between **RuleAttributes** and Document Types. In this way, we can reuse the same attribute declaration (and therefore Java logic) across Document Types.

Once the XML, condensed into a single file, is uploaded into KEW, eDocs of this type can be created and routed from a client application.

All the content in the code examples above is aggregated into a single file below with a single surrounding data tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <ruleAttributes xmlns="ns:workflow/RuleAttribute" xsi:schemaLocation="ns:workflow/RuleAttribute
resource:RuleAttribute">
    <ruleAttribute>
      <name>RuleRoutingAttribute</name>
      <className>org.kuali.rice.kew.rule.RuleRoutingAttribute</className>
      <label>foo</label>
      <description>foo</description>
      <type>RuleAttribute</type>
    </ruleAttribute>
  </ruleAttributes>
  <ruleTemplates xmlns="ns:workflow/RuleTemplate" xsi:schemaLocation="ns:workflow/RuleTemplate
resource:RuleTemplate">
    <ruleTemplate>
      <name>RuleRoutingTemplate</name>
      <description>RuleRoutingTemplate</description>
      <attributes>
        <attribute>
          <name>RuleRoutingAttribute</name>
          <required>true</required>
        </attribute>
      </attributes>
    </ruleTemplate>
  </ruleTemplates>
  <documentTypes xmlns="ns:workflow/DocumentType" xsi:schemaLocation="ns:workflow/DocumentType
resource:DocumentType">
    <documentType>
      <name>EDENSERVICE-DOCS.RuleDocument</name>
      <parent>EDENSERVICE-DOCS</parent>
      <description>Add/Modify Workflow rules</description>
      <label>Add/Modify Workflow rules</label>
      <postProcessorName>org.kuali.rice.kew.postprocessor.RulePostProcessor</postProcessorName>
      <superUserGroupName namespace=KR-WKFLW">WorkflowAdmin</superUserGroupName>
      <blanketApproveGroupName namespace=KR-WKFLW">WorkflowAdmin</blanketApproveGroupName>
      <defaultExceptionGroupName></defaultExceptionGroupName>
      <docHandler>https://yourlocalIP/en-prd/Rule.do?methodToCall=docHandler</docHandler>
      <active>true</active>
      <routingVersion>1</routingVersion>
      <routePaths>
        <routePath>
```

```

        <start name="Adhoc Routing" nextNode="Rule routing Route Level" />
        <requests name="Rule routing Route Level" />
    </routePath>
</routePaths>
<routeNodes>
    <start name="Adhoc Routing">
        <activationType>S</activationType>
        <mandatoryRoute>>false</mandatoryRoute>
    </start>
    <requests name="Workflow Document Routing">
        <activationType>S</activationType>
        <ruleTemplate>RuleRoutingTemplate</ruleTemplate>
        <mandatoryRoute>>true</mandatoryRoute>
    </requests>
</routeNodes>
</documentType>
</documentTypes>
</data>

```

Routing Rules

There is a separate User Guide on how to use the Rule UI. This will show you how to create a Rule as well as modify and delete.

InitiatorRoleAttribute

InitiatorRoleAttribute is a RoleAttribute that exposes an INITIATOR abstract role that resolves to the initiator of the document.

Table 9.8. InitiatorRoleAttribute

Name	Address
Class	InitiatorRoleAttribute
Package	org.kuali.rice.kew.rule
Full	org.kuali.rice.kew.rule.InitiatorRoleAttribute

RoutedByUserRoleAttribute

RoutedByUserRoleAttribute is a RoleAttribute that exposes the user who routed the document.

Table 9.9. RoutedByUserRoleAttribute

Name	Address
Class	RoutedByUserRoleAttribute
Package	org.kuali.rice.kew.rule
Full	org.kuali.rice.kew.rule.RoutedByUserRoleAttribute

NoOpNode

NoOpNode is a SimpleNode implementation that is a code structure example, but has no functionality.

Table 9.10. NoOpNode

Name	Address
Class	NoOpNode
Package	org.kuali.rice.kew.engine.node
Full	org.kuali.rice.kew.engine.node.NoOpNode

RequestActivationNode

RequestActivationNode is a *SimpleNode* that activates any requests on it. It returns true when there are no more requests that require activation.

In *RequestActivationNode*, the *activateRequests* method activates the Action Requests that are pending at this route level of the document. The requests are processed by Priority and then by Request ID. The requests are activated implicitly according to the route level.

Acknowledgement Requests do not cause processing to stop. Only Action Requests for Approval or Completion cause processing to stop at the current document's route level. Inactive requests at a lower level cause a routing exception.

Table 9.11. RequestActivationNode

Name	Address
Class	RequestActivationNode
Package	org.kuali.rice.kew.engine.node
Full	org.kuali.rice.kew.engine.node.RequestActivationNode

NetworkIdRoleAttribute

NetworkIdRoleAttribute is a *RoleAttribute* that routes the request to a NetworkID specified in the document content.

Table 9.12. NetworkIdRoleAttribute

Name	Address
Class	NetworkIdRoleAttribute
Package	org.kuali.rice.kew.engine.node
Full	org.kuali.rice.kew.engine.node.NetworkIdRoleAttribute

Using NetworkIdRoleAttribute for Dynamic Routing

The *RoleAttribute* component in KEW allows for routing to a dynamically generated list of principals or groups. However, in order to do this a Java class which implements the *RoleAttribute* interface must be created. This is fairly simple for Java applications, but it can be painful when integrating with non-Java applications.

Thankfully, the *NetworkIdRoleAttribute* class is an implementation of *RoleAttribute* which is provided out of the box. This allows users to specify in XML format who the document should be routed to.

Setting up a Document Type to use NetworkIdRoleAttribute

There is some KEW setup involved in order to utilize this functionality. The following steps assume an existing Document Type is already in place:

1. Create a "Rule Attribute" in KEW which uses the `org.kuali.rice.kew.rule.NetworkIdRoleAttribute` class.

```
<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
  resource:WorkflowData">
  <ruleAttributes xmlns="ns:workflow/RuleAttribute" xsi:schemaLocation="ns:workflow/RuleAttribute
    resource:RuleAttribute">
    <ruleAttribute>
      <name>Name.Of.My.NetworkIdRoleAttribute</name>
```

```

<className>org.kuali.rice.kew.rule.NetworkIdRoleAttribute</className>
<label>My Label</label>
<description>My Description</description>
<type>RuleXmlAttribute</type>
<configuration>
  <xmlElementLabel>myNetworkId</xmlElementLabel>
  <roleNameLabel>My Role Name</roleNameLabel>
  <groupTogether>true</groupTogether>
</configuration>
</ruleAttribute>
</ruleAttributes>
</data>

```

There are a few areas where the configuration could be changed in this rule attribute

- **name** - The attribute should be named so that it can be identified as belonging to this particular application.
- **label and description** - This can be anything to better describe the rule attribute
- **xmlElementLabel** - This will tell the attribute how it can match XML for the incoming document to determine which xml elements represent the people the document should be routed to
- **roleNameLabel** - This will show up as a label in the route log next to the requests that have been generated
- **groupTogether** - Indicates whether or not all people should be grouped together for the purpose of a "first approve" role. This defaults to false.

2. **Create a "Rule Template" in KEW which uses the attribute previously created.** As with rule attribute, the name and description can be customized as desired.

```

<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <ruleTemplates xmlns="ns:workflow/RuleTemplate" xsi:schemaLocation="ns:workflow/RuleTemplate
resource:RuleTemplate">
    <ruleTemplate>
      <name>Name.Of.My.NetworkIdRuleTemplate</name>
      <description>My Description</description>
      <attributes>
        <attribute>
          <name>Name.Of.My.NetworkIdRoleAttribute</name>
          <required>true</required>
        </attribute>
      </attributes>
    </ruleTemplate>
  </ruleTemplates>
</data>

```

3. **Create a "Routing Rule" in KEW using the Rule Template that was created in the previous step.**

```

<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <rules xmlns="ns:workflow/Rule" xsi:schemaLocation="ns:workflow/Rule resource:Rule">
    <rule>
      <documentType>My.DocumentType.Name</documentType>
      <ruleTemplate>Name.Of.My.NetworkIdRuleTemplate</ruleTemplate>
      <description>My Network Id routing rule</description>
      <responsibilities>
        <responsibility>
          <role>org.kuali.rice.kew.rule.NetworkIdRoleAttribute!networkId</role>
          <approvePolicy>F</approvePolicy>
          <actionRequested>A</actionRequested>
        </responsibility>
      </responsibilities>
    </rule>
  </rules>
</data>

```

```

        <priority>1</priority>
    </responsibility>
</responsibilities>
</rule>
</rules>
</data>

```

There are a few areas where the configuration could be changed in this rule:

- **documentType** - This is the document type the rule should apply to.
- **ruleTemplate** - Use the name of the rule template previously created
- **description** - This can be anything to better describe the rule
- **approvePolicy** - 'F' if only one member of the role should approve or 'A' if all members of the role to approve
- **actionRequested** - 'A' is Approve, 'K' is Acknowledge, and 'F' is FYI

4. **Update the Document Type definition to add the "Route Node" configured to point at this node.**
Here is an example document type:

```

<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="ns:workflow
    resource:WorkflowData">
    <documentTypes xmlns="ns:workflow/DocumentType" xsi:schemaLocation="ns:workflow/DocumentType
        resource:DocumentType">
        <documentType>
            <name>My.DocumentType.Name</name>
            ...
            <routePaths>
                <routePath>
                    <start name="AdHoc" nextNode="My.NetworkId.Node" />
                    <requests name="My.NetworkId.Node" />
                </routePath>
            </routePaths>
            <routeNodes>
                <start name="AdHoc">
                    <activationType>P</activationType>
                </start>
                <requests name="My.NetworkId.Node">
                    <activationType>P</activationType>
                    <ruleTemplate>Name.Of.My.NetworkIdRuleTemplate</ruleTemplate>
                </requests>
            </routeNodes>
        </documentType>
    </documentTypes>
</data>

```

There are a few areas where the configuration could be changed in this document type:

- **name** - This is the name of the document type
- **nextNode and requests name** - Set this to something that describes who the document will be routed to (i.e. supervisor, administrator, etc.
- **ruleTemplate** - Use the name of the rule template previously created

Using the KEW API to Route the Document

Once the document type has been set up, the KEW api can be used to set the appropriate XML on the document. Many of the operations include the ability to pass a DocumentContentUpdate. In this example,

the 'applicationContent' should be set to look like the following. In this case myNetworkId is the name of the xmlElementLabel that was defined in the first step of creating the rule attribute.

```
<NetworkIdRoleAttribute>
  <myNetworkId>dev1</myNetworkId>
  <myNetworkId>dev2</myNetworkId>
  <myNetworkId>dev3</myNetworkId>
  ...
</NetworkIdRoleAttribute>
```

UniversityIdRoleAttribute

UniversityIdRoleAttribute is a *RoleAttribute* that routes requests to an Empl ID specified in the document content.

Table 9.13. UniversityIdRoleAttribute

Name	Address
Class	UniversityIdRoleAttribute
Package	org.kuali.rice.kew.engine.node
Full	org.kuali.rice.kew.engine.node.UniversityIdRoleAttribute

SetVarNode

SetVarNode is a *SimpleNode* that allows you to set document variables.

The definition of *SetVarnode* takes these configuration parameter elements:

- **Name:** The name of the variable to set
- **Value:** The value to which to set the variable. This value is parsed according to *Property/PropertyScheme* syntax. The default *PropertyScheme* is *LiteralScheme*, which evaluates the value simply as a literal; it won't do anything but return the value.

Table 9.14. SetVarNode

Name	Address
Class	SetVarNode
Package	org.kuali.rice.kew.engine.node.var
Full	org.kuali.rice.kew.engine.node.var.SetVarNode

Routing Configuration using KIM Responsibilities

In addition to routing workflow based on users and workgroups using routing rules, you can also route workflow based on KIM responsibilities. This allows you to utilize group membership and role assignments to manage who is permitted to perform approvals.

Route Node Definition

In review, you define a rule-based routing node with XML similar to:

```
<requests name="Rule routing Route Level">
  <activationType>S</activationType>
  <ruleTemplate>RuleRoutingTemplate</ruleTemplate>
  <mandatoryRoute>true</mandatoryRoute>
```

```
<finalApproval>>false</finalApproval>
</requests>
```

A routing node that uses KIM responsibilities can replace a rule-based routing node. You define it with XML similar to:

```
<role name="Purchasing">
  <qualifierResolverClass>
    org.kuali.rice.kns.workflow.attribute.DataDictionaryQualifierResolver
  </qualifierResolverClass>
  <activationType>P</activationType>
</role>
```

Node Name

You name the routing node with the name attribute, just like for a rule-based routing node.

Qualifier Resolver

The qualifier resolver finds any qualifiers that need to be used while matching the responsibility. You can specify it in either of two ways:

- `<qualifierResolver>name</qualifierResolver>` names a rule attribute which identifies the class to use
- `<qualifierResolverClass>class.name</qualifierResolverClass>` provides the fully-qualified name of the Java class to use

Other Options

You can specify `<responsibilityTemplateName>name</responsibilityTemplateName>` to identify the responsibility template to use. This option is not usually used since all of the responsibilities provided with KIM use a template named **Review**.

You can specify `<namespace>name</namespace>` to identify the name space for the responsibility. This option is usually not used since all of the responsibilities provided with KIM use a name space of **KR-WKFLW**.

Matching Routing Nodes to Responsibilities

The KIM responsibility template **Review** defines two details:

- The name of the document type
- The name of the routing node

When you define a responsibility in KIM using this template, you specify a value for each of these details. When a document is routed using responsibility-based routing nodes, KIM receives the type of the document being routed and the name of the node; it then locates any responsibilities which have the same routing node name and either the same document type name or the name of a parent document type (all the way up to the top of the hierarchy). The list of people who gets the request consists of anyone who has been assigned a role with any of the matching responsibilities.

Chapter 10. KEW Administration Guide

This guide provides information on administering a Quali Enterprise Workflow (KEW) installation. Out of the box, KEW comes with a default setup that works well in development and test environments. However, when moving to a production environment, this setup requires adjustments. This document discusses basic administration as well as instructions for working with some of KEW's administration tools.

Configuration Overview

You configure KEW primarily through the **workflow.xml** file. Please see the KEW Configuration Parameters guide for more information on initial configuration of a KEW installation.

Application Constants

Application Constants are the configuration elements in KEW. Each constant is modifiable at system runtime; any changes take effect immediately in KEW. Application Constants are stored in a cluster-safe cache and propagated across all machines when change occurs. For more information about Application Constants, please refer to Application Constants.

Production Environments

When rolling KEW out into a production environment, there are application constants which you may need to change:

- **ActionList.sendEmailNotification** - This is usually set to false in test environments so emails aren't generated during testing. Usually, this is set to true in a production environment to allow email notifications. You also need to ensure that your email service is configured properly to allow KEW to send notifications.
- **ApplicationContext** - In a production environment, this is usually something like en-prd. You must set this value correctly so that KEW's email notifications contain valid links.
- **Backdoor.ShowbackDoorLogin** - The backdoor login allows users to masquerade as other users for testing purposes. It is recommended that you set this value to false in a production environment.
- **RouteManagerPool.numWorkers** - The appropriate value for this depends on the capabilities of your production hardware. If it's set too high, KEW may use so much of the CPU that other applications running on the same machine are adversely impacted.
- **RouteManagerQueue.waitTime** - In test environments, users tend to be more sensitive to immediate feedback since they may be testing processes over the course of a couple minutes that, in practice, occur over a number of days. In test environments, it is recommended that you keep this value low. In a production environment, you can reasonably increase this value without affecting the speed at which documents are routed. This reduces thrashing on the route queue.
- **RouteQueue.isRoutingByIPNumber** - If you are running your production KEW system in a clustered environment, set this value to false. This allows processing of documents in the queue to be distributed across the entire cluster, which enhances routing performance and facilitates load balancing.
- **RouteQueue.maxRetryAttempts** - As with the *RouteManagerQueue.waitTime* constant, in a test environment it is important to find out as quickly as possible if a document is going to go into exception routing (usually indicating a problem in that document's routing setup). In a production environment, it

may make sense to allow a longer period before a document goes into exception routing. This constant, in combination with the *RouteQueue.timeIncrement* constant, determines how long it takes a document to be put into exception routing.

- **RouteQueue.timeIncrement** - Increasing this value results in a longer time before a document goes into exception routing.

XML Ingestion

KEW relies on XML for data population and routing configuration. **XML Ingestor** 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, go to *Ingestor UI* and select the XML file that you want to import:

Figure 10.1. Ingestor



After upload, notice the red arrow and the statement, *Ingested xml doc: <name of file>*:

Figure 10.2. Ingestion Complete



Message Queue Administration

The Message Queue is the main scheduling mechanism in KEW. You use it to schedule documents for asynchronous routing and to queue arbitrary units of work. When KEW places a document into exception routing, it may become stuck after a series of failed attempts. You can use the Route Queue UI to resolve this issue, as well as to fix new entries, if needed.

Examining the Message Queue

The main Message Queue screen:

Figure 10.3. Message Queue Screen

Current Node Info
IP Address: 129.79.211.85
Service Namespace: RICE
message.persistance: true
message.delivery: async
message.off: false

Message ID:
Service Name:
Service Namespace:
IP Number:
Queue Status:
App Specific Value 1:
App Specific Value 2:

Documents currently in route queue: 20
20 items retrieved, displaying all items.

Message Queue Id	Service Name	Service Namespace	IP Number	Queue Status	Queue Priority	Queue Date	Expiration Date	Retry Count	App Specific Value 1	App Specific Value 2	Actions
53317554	(TK)ActionInvocationProcessor	RICE	129.79.211.87	ROUTING	10	04:00 PM 01/09/2012	04:00 PM 01/09/2012	0	9767012		View Edit ReQueue
53317553	(TK)ActionInvocationProcessor	RICE	129.79.211.87	ROUTING	10	04:00 PM 01/09/2012	04:00 PM 01/09/2012	0	9753106		View Edit ReQueue
53317552	(TK)ActionInvocationProcessor	RICE	129.79.211.87	ROUTING	10	04:00 PM 01/09/2012	04:00 PM 01/09/2012	0	9753857		View Edit ReQueue
53317551	(TK)ActionInvocationProcessor	RICE	129.79.211.87	ROUTING	10	04:00 PM 01/09/2012	04:00 PM 01/09/2012	0	9756456		View Edit ReQueue
53317550	(TK)ActionInvocationProcessor	RICE	129.79.211.87	ROUTING	10	04:00 PM 01/09/2012	04:00 PM 01/09/2012	0	9749536		View Edit ReQueue
53317549	(RICE)DocumentRoutingService	RICE	129.79.211.86	ROUTING	5	04:00 PM 01/09/2012	04:00 PM 01/09/2012	0	9828162		View Edit ReQueue
53316971	OSCacheNotificationService	RICE	129.79.211.87	ROUTING	15	03:56 PM 01/09/2012	03:56 PM 01/09/2012	0			View Edit ReQueue

Examining this sample screen, we see there are 20 entries in the message queue, one on each row. The columns display information about each entry:

- **Message Queue Id** - The primary key of this route queue entry in the data store
- **Service Name**
- **Service Namespace**
- **IP Number** - The IP address of the machine on which the entry was created. In the environment pictured, we have three machines in our cluster. The IP number shows from which machine each entry was queued up.
- **Queue Status** – The entry can be in a state of QUEUED, ROUTING, or EXCEPTION:
 - A QUEUED entry is waiting for a worker thread to pick it up.
 - A ROUTING entry currently has a worker working on it.
 - An EXCEPTION entry has a problem and the route manager cannot access it. An administrator manually sets an EXCEPTION status to suspend a route queue entry until a problem can be diagnosed.
- **Queue Priority** - The priority of the entry in the queue, where entries with the lowest number are processed first

- **Queue Date** - The date that KEW should process this queue entry. If the queue checker runs and discovers the queue date for an entry is equal to or earlier than the current time, it processes that entry.
- **Expiration Date**
- **Retry Count** - The number of times KEW has attempted to process the entry
- **App Specific Value 1** - The parameters to be passed to the Route Queue processor such as document ID
- **App Specific Value 2** - The parameters to be passed to the Route Queue processor
- **Action** - The Edit link in the Action column allows you to edit the route queue entry.

Once a message entry has been successfully processed, it is deleted from the queue.

Diagnosing and Fixing Problems

Sometimes it is necessary to manually edit a route queue entry that is *halted inside of the queue*. This situation might happen when:

- KEW encounters an error trying to put the document into exception routing. This could occur if there is a database error or the document's *PostProcessor* throws an exception when it's notified of a status change
- KEW is improperly shut down in the middle of an entry being processed
- The database goes down while an entry is being processed

In all cases, the status of the entry is **ROUTING**, but there is no longer a worker thread processing the entry. Currently, KEW doesn't implement any auto-detection of failure cases. To put one of these entries in a state where it can be picked up by the route manager again, simply click the **Edit** link and set the entry's status back to **QUEUED**. Here's a screen shot of the **Route Queue Entry - Edit** screen:

Figure 10.4. Route Queue Entry Edit Screen

The screenshot shows the 'Route Queue Entry Edit Screen' with a table comparing 'Existing Route Queue Values' and 'New Route Queue Values'. The 'Queue Status' is set to 'ROUTING' and the 'Retry Count' is set to '1', both of which are circled in red. Below the table are buttons for 'queue document', 'delete', 'reset', and 'clear'.

	Existing Route Queue Values	New Route Queue Values
Route Queue Id:	1357006	1357006 ?
Document Id:	550176	550176 ?
Queue Priority:	21	0 ?
Queue Status:	R	ROUTING ?
Queue Date:	07/12/2006	07/12/2006 ?
Retry Count:	1	1 ?
IP Number:	129.79.210.179	129.79.210.179 ?
Processor Class Name:		? ?
Processor Value:		? ?

queue document delete reset clear

Use the Queue Status dropdown list to change the status of the entry. You may also want to set the Retry Count to zero to allow you to diagnose the problem before the document goes into exception routing.

Chapter 11. KEW System Parameters

System Parameters Covered

Table 11.1. KEW System Parameters

Name	Value	Description
MAX_MEMBERS_PER_PAGE	20	The maximum number of role or group members to display at once on their documents. If the number is above this value, the document will switch into a paging mode with only this many rows displayed at a time.
PREFIXES	Ms;Mrs;Mr;Dr	
SUFFIXES	Jr;Sr;Mr;Md	
CHECK_ENCRYPTION_SERVICE_OVERRIDE_IND	Y	Flag for enabling/disabling (Y/N) the demonstration encryption check.
DATE_TO_STRING_FORMAT_FOR_FILE_NAME	yyyyMMdd	A single date format string that the DateTimeService will use to format dates to be used in a file name when DateTimeServiceImpl.toStringForFilename(Date) is called. For a more technical description of how characters in the parameter value will be interpreted, please consult the Java Documentation for java.text.SimpleDateFormat. Any changes will be applied when the application is restarted.
DATE_TO_STRING_FORMAT_FOR_USER_INTERFACE	MM/dd/yyyy	A single date format string that the DateTimeService will use to format a date to be displayed on a web page. For a more technical description of how characters in the parameter value will be interpreted, please consult the Java Documentation for java.text.SimpleDateFormat. Any changes will be applied when the application is restarted.
DEFAULT_COUNTRY	US	Used as the default country code when relating records that do not have a country code to records that do have a country code, e.g. validating a zip code where the country is not collected.
ENABLE_DIRECT_INQUIRIES_IND	Y	Flag for enabling/disabling direct inquiries on screens that are drawn by the nervous system (i.e. lookups and maintenance documents)
ENABLE_FIELD_LEVEL_HELP_IND	N	Indicates whether field level help links are enabled on lookup pages and documents.
MAX_FILE_SIZE_DEFAULT_UPLOAD	5M	Maximum file upload size for the application. Must be an integer, optionally followed by "K", "M", or "G". Only used if no other upload limits are in effect.
SENSITIVE_DATA_PATTERNS	[0-9]{9};[0-9]{3}-[0-9]{2}-[0-9]{4}	A semi-colon delimited list of regular expressions that identify potentially sensitive data in strings. These patterns will be matched against notes, document explanations, and routing annotations.
STRING_TO_DATE_FORMATS	MM/dd/yy;MM-dd-yy;MMMM dd, yyyy;MMddy	A semi-colon delimited list of strings representing date formats that the DateTimeService will use to parse dates when DateTimeServiceImpl.convertToSqlDate(String) or DateTimeServiceImpl.convertToDate(String) is called. Note that patterns will be applied in the order listed (and the first applicable one will be used). For a more technical description of how characters in the parameter value will be interpreted, please consult the Java Documentation for java.text.SimpleDateFormat. Any changes will be applied when the application is restarted.
STRING_TO_TIMESTAMP_FORMATS	MM/dd/yyyy hh:mm a	A semi-colon delimited list of strings representing date formats that the DateTimeService will use to parse date and times when DateTimeServiceImpl.convertToDateTime(String) or DateTimeServiceImpl.convertToSqlTimestamp(String) is called. Note that patterns will be applied in the order listed (and the first applicable one will be used). For a more technical description of how characters in the parameter value will be interpreted, please consult the

KEW System Parameters

Name	Value	Description
		Java Documentation for java.text.SimpleDateFormat. Any changes will be applied when the application is restarted.
TIMESTAMP_TO_STRING_FORMAT_FOR_FILE_NAME	yyyyMMdd-HH-mm-ss-S	A single date format string that the DateTimeService will use to format a date and time string to be used in a file name when DateTimeServiceImpl.toDateTimeStringForFilename(Date) is called. For a more technical description of how characters in the parameter value will be interpreted, please consult the Java Documentation for java.text.SimpleDateFormat. Any changes will be applied when the application is restarted.
TIMESTAMP_TO_STRING_FORMAT_FOR_USER_INTERFACE	MM/dd/yyyy hh:mm a	A single date format string that the DateTimeService will use to format a date and time to be displayed on a web page. For a more technical description of how characters in the parameter value will be interpreted, please consult the Java Documentation for java.text.SimpleDateFormat. Any changes will be applied when the application is restarted.
ACTIVE_FILE_TYPES	collectorInputFileType; procurementCardInputFileType; enterpriseFeederFileSetType; assetBarcodeInventoryInputFileType customerLoadInputFileType	Batch file types that are active options for the file upload screen.
SCHEDULE_ADMIN_GROUP	KR-WKFLW:WorkflowAdmin	The workgroup to which a user must be assigned to modify batch jobs.
DEFAULT_CAN_PERFORM_ROUTE_REPORT_IND	N	If Y, the Route Report button will be displayed on the document actions bar if the document is using the default DocumentAuthorizerBase.getDocumentActionFlags to set the canPerformRouteReport property of the returned DocumentActionFlags instance.
EXCEPTION_GROUP	KR-WKFLW:WorkflowAdmin	The workgroup to which a user must be assigned to perform actions on documents in exception routing status.
MAX_FILE_SIZE_ATTACHMENT	5M	Maximum attachment uploads size for the application. Used by KualiDocumentFormBase. Must be an integer, optionally followed by "K", "M", or "G".
PESSIMISTIC_LOCK_ADMIN_GROUP	KFS:KUALI_ROLE_SUPERVISOR	Workgroup which can perform admin deletion and lookup functions for Pessimistic Locks.
SEND_NOTE_WORKFLOW_NOTIFICATION_ACTIONS	K	Some documents provide the functionality to send notes to another user using a workflow FYI or acknowledge functionality. This parameter specifies the default action that will be used when sending notes. This parameter should be one of the following 2 values: "K" for acknowledge or "F" for "fyi". Depending on the notes and workflow service implementation, other values may be possible.
SESSION_TIMEOUT_WARNING_MESSAGE_TIME	5	The number of minutes before a session expires. That user should be warned when a document uses pessimistic locking.
SUPERVISOR_GROUP	KR-WKFLW:WorkflowAdmin	Workgroup which can perform almost any function within Kuali.
MULTIPLE_VALUE_RESULTS_EXPIRATION_SECONDS	86400	Lookup results may continue to be persisted in the DB long after they are needed. This parameter represents the maximum amount of time, in seconds, that the results will be allowed to persist in the DB before they are deleted from the DB.
MULTIPLE_VALUE_RESULTS_PER_PAGE	100	Maximum number of rows that will be displayed on a lookup results screen.
RESULTS_DEFAULT_MAX_COLUMN_LENGTH	70	If a maxLength attribute has not been set on a lookup result field in the data dictionary, then the result column's max length will be the value of this parameter. Set this parameter to 0 for an unlimited default length or a positive value (i.e. greater than 0) for a finite max length.
RESULTS_LIMIT	200	Maximum number of results returned in a look-up query.
MAX_AGE	86400	Pending attachments are attachments that do not yet have a permanent link with the associated Business Object (BO). These pending attachments are stored in the attachments.pending.directory (defined in the configuration service). If the BO is never persisted, then this attachment will become orphaned (i.e. not associated with any BO), but will remain in this directory. The

KEW System Parameters

Name	Value	Description
		PurgePendingAttachmentsStep batch step deletes these pending attachment files that are older than the value of this parameter. The unit of this value is seconds. Do not set this value too short, as this will cause problems attaching files to BOs.
NUMBER_OF_DAYS_SINCE_LAST_UPDATE	1	Determines the age of the session document records that the step will operate on, e.g. if this parameter is set to 4, the rows with a last update timestamp older than 4 days prior to when the job is running will be deleted.
CUTOFF_TIME	02:00:00:AM	Controls when the daily batch schedule should terminate. The scheduler service implementation compares the start time of the schedule job from quartz with this time on day after the schedule job started running.
CUTOFF_TIME_NEXT_DAY_IND	Y	Controls whether when the system is comparing the schedule start day & time with the scheduleStep_CUTOFF_TIME parameter, it considers the specified time to apply to the day after the schedule starts.
STATUS_CHECK_INTERVAL	30000	Time in milliseconds that the scheduleStep should wait between iterations.
ACTION_LIST_DOCUMENT_POPUP_IND	Y	Flag to specify if clicking on a Document ID from the Action List will load the Document in a new window.
ACTION_LIST_ROUTE_LOG_POPUP_IND	N	Flag to specify if clicking on a Route Log from the Action List will load the Route Log in a new window.
EMAIL_NOTIFICATION_TEST_ADDRESS		Default email address used for testing.
HELP_DESK_NAME_GROUP	KR-WKFLW:WorkflowAdmin	The name of the group who has access to the "Help Desk" feature on the Action List.
PAGE_SIZE_THROTTLE		Throttles the number of results returned on all users Action Lists, regardless of their user preferences. This is intended to be used in a situation where excessively large Action Lists are causing performance issues.
SEND_EMAIL_NOTIFICATION_IND	N	Flag to determine whether or not to send email notification.
KIM_PRIORITY_ON_DOC_TYP_PERMS_IND	N	Flag for enabling/disabling document type permission checks to use KIM Permissions as priority over Document Type policies.
MAXIMUM_NODES_BEFORE_RUNAWAY		The maximum number of nodes the workflow engine will process before it determines the process is a runaway process. This is to prevent infinite "loops" in the workflow engine.
SHOW_ATTACHMENTS_IND	Y	Flag to specify whether or not a file upload box is displayed for KEW notes which allows for uploading of an attachment with the note.
SHOW_BACK_DOOR_LOGIN_IND	Y	Flag to show the backdoor login.
TARGET_FRAME_NAME	iframe_51148	Defines the target iframe name that the KEW internal portal uses for its menu links.
DOCUMENT_SEARCH_POPUP_IND	Y	Flag to specify if clicking on a Document ID from Document Search will load the Document in a new window.
DOCUMENT_SEARCH_ROUTE_LOG_POPUP_IND	N	Flag to specify if clicking on a Route Log from Document Search will load the Route Log in a new window.
FETCH_MORE_ITERATION_LIMIT		Limit of fetch more iteration for document searches.
RESULT_CAP		Maximum number of documents to return from a search.
DOCUMENT_TYPE_SEARCH_INSTRUCTION	Enter document type information below and click search.	Instructions for searching document types.
DEBUG_TRANSFORM_IND	N	Defines whether the debug transform is enabled for eDocLite.
USE_XSLTC_IND	N	Defines whether XSLTC is used for eDocLite.
IS_LAST_APPROVER_ACTIVATE_FIRST_IND		A flag to specify whether the WorkflowInfo.isLastApproverAtNode(...) API method attempts to active requests first, prior to execution.
REPLACE_INSTRUCTION	Enter the reviewer to replace.	Instructions for replacing a reviewer.
FROM_ADDRESS	rice.test@kulai.org	Default from email address for notifications. If not set, this value defaults to admin@localhost.

KEW System Parameters

Name	Value	Description
NOTE_CREATE_NEW_INSTRUCTION	Create or modify note information.	Instructions for creating a new note.
RESTRICT_DOCUMENT_TYPES		Comma separated list of Document Types to exclude from the Rule Quicklinks.
CUSTOM_DOCUMENT_TYPES		Defines custom Document Type processes to use for certain types of routing rules.
DELEGATE_LIMIT	20	Specifies that maximum number of delegation rules that will be displayed on a Rule inquiry before the screen shows a count of delegate rules and provides a link for the user to show them.
GENERATE_ACTION_REQUESTS_IND	Y	Flag to determine whether or not a change to a routing rule should be applied retroactively to existing documents.
ROUTE_LOG_POPUP_IND	F	Flag to specify if clicking on a Route Log from a Routing Rule inquiry will load the Route Log in a new window.
RULE_CACHE_REQUEUE_DELAY	5000	Amount of time after a rule change is made before the rule cache update message is sent.
RULE_CREATE_NEW_INSTRUCTION	Please select a rule template and document type.	Instructions for creating a new rule.
RULE_LOCKING_ON_IND	Y	Defines whether rule locking is enabled.
RULE_SEARCH_INSTRUCTION	Use fields below to search for rules.	Instructions for the rule search.
RULE_TEMPLATE_CREATE_NEW_INSTRUCTION	Enter a rule template name and description. Please select all necessary rule attributes for this template.	Instructions for creating new rule templates.
RULE_TEMPLATE_SEARCH_INSTRUCTION	Use fields below to search for rule templates.	Instructions for the rule template search.
NOTIFY_EXCLUDED_USERS_IND		<p>Defines a group name (in the format "namespace:name") which contains members who should never receive notification action requests from KEW. Notification requests in KEW are generated when someone disapproves or blanket approves or blanket approves are exist to notify other approvers that these actions have taken place.</p> <p>The most common use for this is in the case of "system" users who participate in workflow transactions. In these cases, since they aren't actual users who would be checking their action list, it doesn't make sense to send them requests since they won't ever be fulfilled.</p>

Chapter 12. Defining Workflow Processes

Defining Workflow Processes Using Document Types

A Document Type is an object that brings workflow components together into a cohesive unit (routing configuration). One of its primary responsibilities is to define the routing path for a document. The routing path is the process definition for the document. It can consist of various types of nodes that perform certain actions, such as sending action requests to responsible parties, transmitting emails, or splitting the route path into parallel branches.

In addition to the routing path, it contains the Post Processor which receives event callbacks from the engine, the DocHandler which is the access point into the client application from the Action List and Access Control for certain actions. It can also define various policies that control how documents of that type are processed by the workflow engine.

This document has four parts:

1. A detailed explanation of the common fields in the **Document Type** XML definition
2. An example of each Document Type with a description of each field in it
3. Descriptions of the Document Type *policies*
4. A description of inheritance as applied to Document Types

There are some common attributes in every **Document Type**, but each **Document Type** can be customized to provide different functions.

- Document Types
- Document Type Policies
- Inheritance

Common Fields in Document Type XML Definition

Table 12.1. Common Fields in Document Type XML Definition

Field	Description
name	The name of the Document Type
parent	The parent Document Type of this Document Type. Each Child Document Type inherits the attributes of its parent Document Type.
description	The description of the Document Type; its primary responsibilities.
label	The label of the Document Type, how it's recognized
postProcessorName	The name of the postProcessor that takes charge of the routing for this Document Type
postprocessor	A component that gets called throughout the routing process and handles a set of standard events that all eDocs (electronic documents) go through.
authorizer	A component that gets called during the routing process to perform authorization checks. Applications can customize this component on a per-doctype basis.

Defining Workflow Processes

Field	Description
superGroupName	The name of a workgroup whose members are the super users of this Document Type. Super users of this Document Type can execute a super user document search on this Document Type.
blanketApproveGroupName	The name of a workgroup whose members have the blanketapprove rights over this Document Type.
defaultExceptionGroupName	The name of the workgroup whose members receive an exception notice when a document of this Document Type encounters an exception in its routing.
docHandler	The DocHandler that handles the routing of this Document Type
active	A true or false indicator for the active status of this document
validApplicationStatuses	The set of valid application document statuses for this document type. If this optional configuration is set, the application document status will only allow the specified values to be set.
policies	The policies that apply to this Document Type
policy	The policy that applies to this Document Type. Use this when there is only one policy for the Document Type. value: A true or false indicator for whether the action for the policy will be taken
routingVersion	<i>This field exists only for backward compatibility with older versions of KEW. Originally, KEW only supported sequential routing paths (as opposed to those with splits and joins). The KEW <code>getDocRouteLevel()</code> API returns an integer that represents the numerical step in the routing process. This number only has meaning for those documents that define sequential routing.</i> <ul style="list-style-type: none"> • A document with a routingVersion of "1" will keep track of the route level number. • A document with a routingVersion of "2" (the default, unless explicitly defined in the Document Type configuration) will <i>NOT</i> keep track of the route level number and an exception will be thrown if code attempts to access that value. <i>New Document Type definitions do NOT need, and should NOT have, this flag defined.</i>
routePaths	The routing paths for this Document Type
routePath	The routing path for this Document Type. Use this field when there is just one routing path for this Document Type.
routeNode	A point or node on the routing path of this Document Type
routeModule	The most basic module; it allows KEW to generate Action Requests
start	The starting node of this Document Type during routing
requests	The requested next node in the routing of this Document Type
activationType	The activation type of the next node that is requested by this Document Type. There are two activation types: <ul style="list-style-type: none"> • P: Parallel: Multiple nodes in the routing process are activated at the same time • S: Serial or Sequential: The nodes in the routing process are activated one at a time • R: Priority-Parallel: The multiple nodes with the same priority are activated at the same time before moving to the next priority
ruleTemplate	The ruleTemplate that applies to the routing node in this Document Type
split	The routing path splits into branches and can continue on any of them at a split.
branch	One of the branches in the routing path.
join	The point in the routing path where the split branches join together.
process	There is a sub-process in the routing path; in other words, some nodes in the routing path will activate a sub-process.
simple	A new node in the routing path <ul style="list-style-type: none"> • type: The type of the new routing node • value: The value of the new routing node

Field	Description
	<ul style="list-style-type: none"> • message: The message associated with the new routing node • level: The routing level of the new routing node • log: The log name of the new routing node
dynamic	This changes the node to dynamic when it transitions to the next node in the routing path; therefore, the routing path is dynamic rather than static.

Document Types

Document Type Examples

BlanketApproveTest

```
<documentType>
  <name>BlanketApproveTest</name>
  <description>BlanketApproveTest</description>
  <label>BlanketApproveTest</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <policies>
    <policy>
      <name>DEFAULT_APPROVE</name>
      <value>>false</value>
    </policy>
  </policies>
</documentType>
```

- **name:** This is the Document Type for Blanket Approve Test.
- **description:** This Document Type is used to test the Blanket Approve function.
- **label:** This Document Type is recognized as the BlanketApproveTest type.
- **postProcessorName:** The postProcessor for this Document Type is org.kuali.rice.kew.postprocessor.DefaultPostProcessor.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is _blank.
- **active:** This Document Type is currently Active. In other words, it is in use.
- **Policies** for this Document Type contains two policies: The DEFAULT_APPROVE policy is set false by default. In other words, the default approve action on this type of document is NOT to approve it.

BlanketApproveSequentialTest

```

<documentType>
  <name>BlanketApproveSequentialTest</name>
  <parent>BlanketApproveTest</parent>
  <description>BlanketApproveSequentialTest</description>
  <label>BlanketApproveSequentialTest</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW">WorkflowAdmin</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW">WorkflowAdmin</defaultExceptionGroupName>

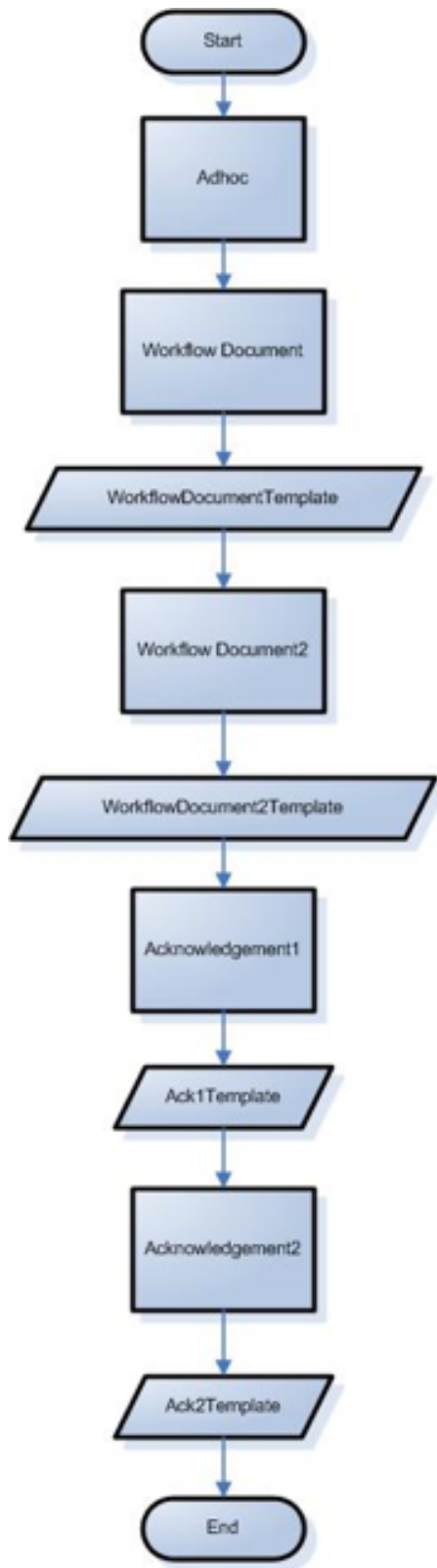
  <docHandler>_blank</docHandler>
  <active>true</active>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="WorkflowDocument" />
      <requests name="WorkflowDocument" nextNode="WorkflowDocument2" />
      <requests name="WorkflowDocument2" nextNode="Acknowledge1" />
      <requests name="Acknowledge1" nextNode="Acknowledge2" />
      <requests name="Acknowledge2" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
    </start>
    <requests name="WorkflowDocument">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
    </requests>
    <requests name="WorkflowDocument2">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocument2Template</ruleTemplate>
    </requests>
    <requests name="Acknowledge1">
      <activationType>P</activationType>
      <ruleTemplate>Ack1Template</ruleTemplate>
    </requests>
    <requests name="Acknowledge2">
      <activationType>P</activationType>
      <ruleTemplate>Ack2Template</ruleTemplate>
    </requests>
  </routeNodes>
</documentType>

```

- **name:** This is the Document Type for Blanket Approve Sequential Test. There is a sequence of routing nodes, and no routing node can be skipped.
- **parent:** The parent Document Type is BlanketApproveTest. This Document Type inherits the policies that BlanketApproveTest has.
- **description:** This Document Type is used to test the Blanket Approve Sequential function.
- **label:** This Document Type is recognized as the blanketApproveSequentialTest type.
- **postProcessorName:** The postProcessor for this Document Type is org.kuali.rice.kew.postprocessor.DefaultPostProcessor.
- **superUserGroupName:** The super users for this Document Type are members of the WorkflowAdmin.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApprove right on this type of document.
- **defaultExceptionGroupName:** The members of the WorkflowAdmin will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is _blank.

- **active:** This Document Type is currently active. In other words, it is in use.
- **routePath:** The routing path for this Document Type is: AdHoc -> WorkflowDocument -> WorkflowDocument2 -> Acknowledge1 -> Acknowledge2.
- **routeNode:** Based on the routePath, there are five nodes in the routing of this Document Type:
 - The starting node for this Document Type is AdHoc. On the initiation of a document of this type, the postProcessor in Quali Enterprise Workflow (KEW) activates the node, AdHoc.
 - The next node in the routing for this Document Type is WorkflowDocument. On request, the node is activated and applies the rules in rule template, WorkflowDocumentTemplate.
 - The next node in the routing for this Document Type is WorkflowDocument2. On request, the node is activated and applies the rules in rule template, WorkflowDocument2Template.
 - The next node in the routing for this Document Type is Acknowledge1. On request, the node is activated and applies the rules in rule template, Ack1Template.
 - The next node in the routing for this Document Type is Acknowledge2. On request, the node is activated and applies the rules in rule template, Ack2Template.

Figure 12.1. BlanketApproveSequentialTest Workflow



BlanketApproveParallelTest

```

<documentType>
  <name>BlanketApproveParallelTest</name>
  <parent>BlanketApproveTest</parent>
  <description>BlanketApproveParallelTest</description>
  <label>BlanketApproveParallelTest</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="WorkflowDocument" />
      <requests name="WorkflowDocument" nextNode="Split" />
      <split name="Split" nextNode="WorkflowDocumentFinal">
        <branch name="B1">
          <requests name="WorkflowDocument2-B1" nextNode="WorkflowDocument3-B1" />
          <requests name="WorkflowDocument3-B1" nextNode="Join" />
        </branch>
        <branch name="B2">
          <requests name="WorkflowDocument3-B2" nextNode="WorkflowDocument2-B2" />
          <requests name="WorkflowDocument2-B2" nextNode="Join" />
        </branch>
        <branch name="B3">
          <requests name="WorkflowDocument4-B3" nextNode="Join" />
        </branch>
        <join name="Join" />
      </split>
      <requests name="WorkflowDocumentFinal" nextNode="Acknowledge1" />
      <requests name="Acknowledge1" nextNode="Acknowledge2" />
      <requests name="Acknowledge2" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
    </start>
    <requests name="WorkflowDocument">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
    </requests>
    <split name="Split" />
    <requests name="WorkflowDocument2-B1">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocument2Template</ruleTemplate>
    </requests>
    <requests name="WorkflowDocument2-B2">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocument2Template</ruleTemplate>
    </requests>
    <requests name="WorkflowDocument3-B1">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocument3Template</ruleTemplate>
    </requests>
    <requests name="WorkflowDocument3-B2">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocument3Template</ruleTemplate>
    </requests>
    <requests name="WorkflowDocument4-B3">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocument4Template</ruleTemplate>
    </requests>
    <join name="Join" />
    <requests name="WorkflowDocumentFinal">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocumentFinalTemplate</ruleTemplate>
    </requests>
    <requests name="Acknowledge1">
      <activationType>P</activationType>
      <ruleTemplate>Ack1Template</ruleTemplate>
    </requests>
    <requests name="Acknowledge2">
      <activationType>P</activationType>
      <ruleTemplate>Ack2Template</ruleTemplate>
    </requests>
  </routeNodes>

```

```
</requests>
</routeNodes>
</documentType>
```

- **name:** This is the Document Type for Blanket Approve Parallel Test. At some point in the routing, the route path may split and a node can be skipped if another parallel node takes action on the document.
- **Parent:** The parent Document Type is BlanketApproveTest. This Document Type inherits the routing that exists for BlanketApproveTest.
- **description:** This Document Type is used to test the Blanket Approve Parallel function.
- **label:** This Document Type is recognized as the blanketApproveParallelTest type.
- **postProcessorName:** The postProcessor for this Document Type is *org.kuali.rice.kew.postprocessor.DefaultPostProcessor*.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is *_blank*.
- **active:** This Document Type is currently active. In other words, it is in use.
- **routePath:** The routing path for this Document Type is: AdHoc -> WorkflowDocument -> split -> B1\B2\B3 -> Join -> WorkflowDocumentFinal -> Acknowledge1 -> Acknowledge2.
- **routeNode:** Based on the routePath, there are six nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node, **AdHoc**.
 - The next node in the routing for this Document Type is **WorkflowDocument**. On request, the node is activated and applies the rules in rule template, **WorkflowDocumentTemplate**. Then, the routing path splits into three branches for the next node.
 - One branch is B1. On request, the node **WorkflowDocument2-B1** is activated and applies the **WorkflowDocument2Template**. The next node in this branch is **WorkflowDocument3-B1**. On request, the node is activated and applies the **WorkflowDocument3Template**.
 - One branch is B2. On request, the node **WorkflowDocument3-B2** is activated and applies the **WorkflowDocument3Template**. The next node in this branch is **WorkflowDocument2-B2**. On request, the node is activated and applies the **WorkflowDocument2Template**.
 - One branch is B3. On request, the node **WorkflowDocument4-B3** is activated and applies the **WorkflowDocument4Template**.
 - Then, the routing path joins and the route merges back together into one route.
 - The next node in the routing for this Document Type is **WorkflowDocumentFinal**. On request, the node is activated and applies the rules in rule template, **WorkflowDocumentFinalTemplate**.

- The next node in the routing for this Document Type is **Acknowledge1**. On request, the node is activated and applies the rules in rule template, **Ack1Template**.
- The next node in the routing for this Document Type is **Acknowledge2**. On request, the node is activated and applies the rules in rule template, **Ack2Template**.

Figure 12.2. BlanketApproveParallelTest Workflow



NotificationTest


```

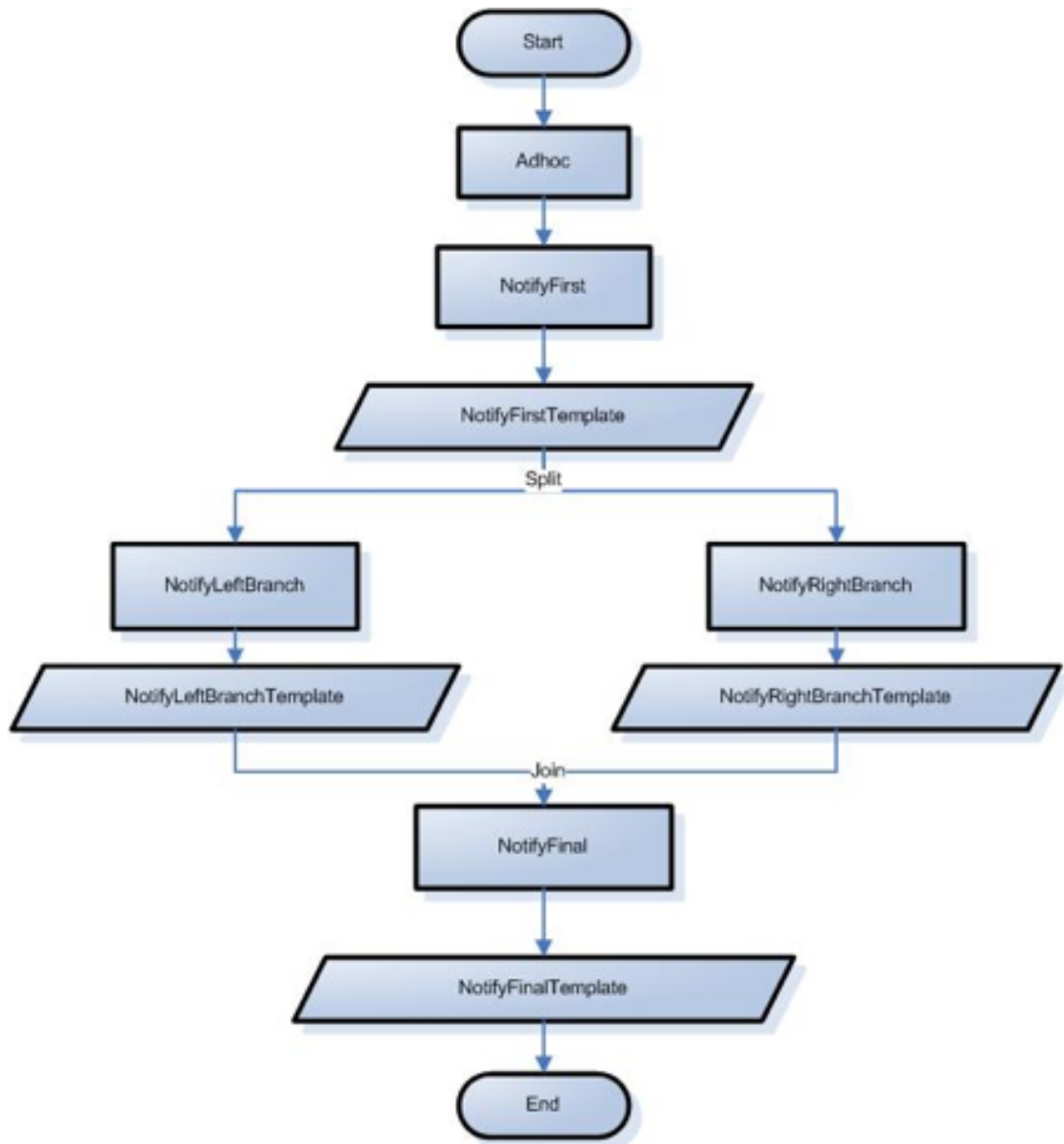
<documentType>
  <name>NotificationTest</name>
  <description>NotificationTest</description>
  <label>NotificationTest</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superGroupName namespace="KR-WKFLW" >TestWorkgroup</superGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="NotifyFirst" />
      <requests name="NotifyFirst" nextNode="Split" />
      <split name="Split" nextNode="NotifyFinal">
        <branch name="LeftBranch">
          <requests name="NotifyLeftBranch" nextNode="Join" />
        </branch>
        <branch name="RightBranch">
          <requests name="NotifyRightBranch" nextNode="Join" />
        </branch>
        <join name="Join" />
      </split>
      <requests name="NotifyFinal" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
    </start>
    <requests name="NotifyFirst">
      <activationType>P</activationType>
      <ruleTemplate>NotifyFirstTemplate</ruleTemplate>
    </requests>
    <split name="Split" />
    <requests name="NotifyLeftBranch">
      <activationType>P</activationType>
      <ruleTemplate>NotifyLeftBranchTemplate</ruleTemplate>
    </requests>
    <requests name="NotifyRightBranch">
      <activationType>P</activationType>
      <ruleTemplate>NotifyRightBranchTemplate</ruleTemplate>
    </requests>
    <join name="Join" />
    <requests name="NotifyFinal">
      <activationType>P</activationType>
      <ruleTemplate>NotifyFinalTemplate</ruleTemplate>
    </requests>
  </routeNodes>
</documentType>

```

- **name:** This is the Document Type for Notification Test. At some point in the routing, the route path may split, and a node can be skipped if another notification node takes action on the document.
- **description:** This Document Type is used to test the notification function.
- **label:** This Document Type is recognized as the NotificationTest type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.

- **docHandler:** The Doc Handler for this type of document is **_blank**.
- **active:** This Document Type is currently active. In other words, it is in use.
- **routePath:** The routing path for this Document Type is: AdHoc -> NotifyFirst -> split -> LeftBranch \RightBranch -> Join -> NotifyFinal.
- **routeNode:** Based on the routePath, there are four nodes in the routing of this Document Type:
 - o The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node, **AdHoc**.
 - The next node in the routing for this Document Type is **NotifyFirst**. On request, the node is activated and applies the rules in rule template, **NotifyFirstTemplate**. Then the routing path splits into two branches for the next node.
 - One branch is **LeftBranch**. On request, the node is activated and applies the **NotifyLeftBranchTemplate**.
 - One branch is **RightBranch**. On request, the node is activated and applies the **NotifyRightBranchTemplate**.
 - Then the routing path joins together again.
 - The next node in the routing for this Document Type is **NotifyFinal**. On request, the node is activated and applies the rules in rule template, **NotifyFinalTemplate**

Figure 12.3. NotificationTest Workflow



NotificationTestChild

```

<documentType>
  <name>NotificationTestChild</name>
  <parent>NotificationTest</parent>
  <description>NotificationTest</description>
  <label>NotificationTest</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW" > TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <policies>
    <policy>
      <name>SEND_NOTIFICATION_ON_SU_APPROVE</name>
      <value>true</value>
    </policy>
  </policies>
</documentType>
    
```

```
</policies>
</documentType>
```

- **name:** This is the Document Type for Notification Test Child.
- **Parent:** The parent Document Type is NotificationTest. This Document Type inherits the routing that NotificationTest has.
- **description:** This Document Type is used to test the Notification function.
- **label:** This Document Type is recognized as the NotificationTest type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is **_blank**.
- **active:** This Document Type is currently active. In other words, it is in use.
- **Policy:** There is only one policy that applies to this Document Type: SEND_NOTIFICATION_ON_SU_APPROVE. This policy currently applies to this Document Type. In other words, a notification will be sent to the designated two users when a SuperUser approves a document of this type.

BlanketApproveMandatoryNodeTest

```
<documentType>
  <name>BlanketApproveMandatoryNodeTest</name>
  <parent>BlanketApproveTest</parent>
  <description>BlanketApproveMandatoryNodeTest</description>
  <label>BlanketApproveMandatoryNodeTest</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="WorkflowDocument" />
      <requests name="WorkflowDocument" nextNode="WorkflowDocument2" />
      <requests name="WorkflowDocument2" nextNode="Acknowledge1" />
      <requests name="Acknowledge1" nextNode="Acknowledge2" />
      <requests name="Acknowledge2" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
    </start>
    <requests name="WorkflowDocument">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
      <mandatoryRoute>true</mandatoryRoute>
    </requests>
    <requests name="WorkflowDocument2">
      <activationType>P</activationType>
```

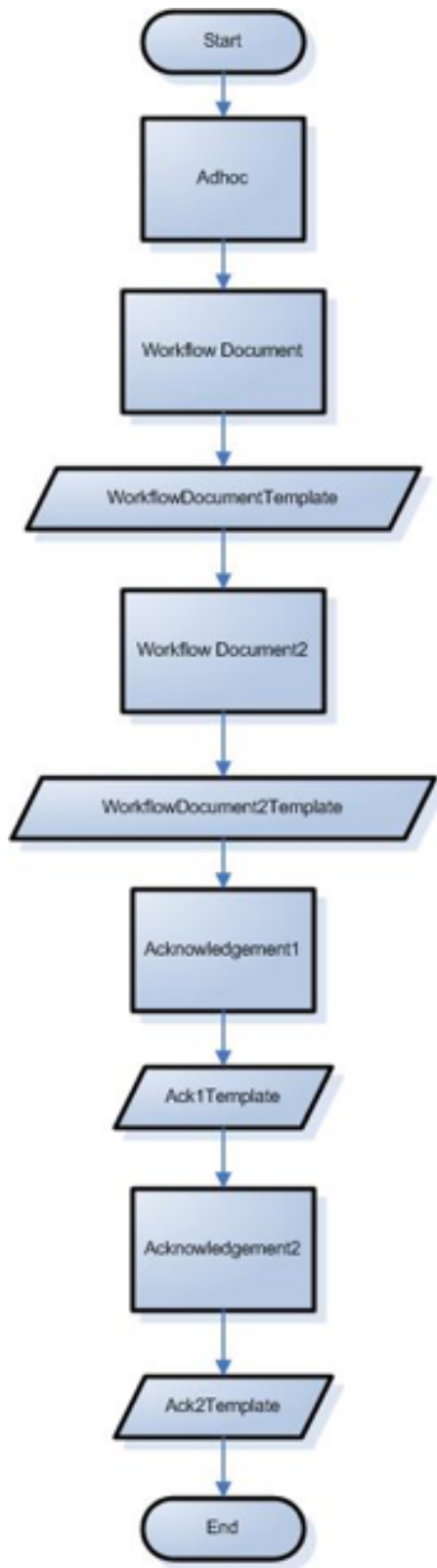
```

<ruleTemplate>WorkflowDocument2Template</ruleTemplate>
<mandatoryRoute>true</mandatoryRoute>
<finalApproval>true</finalApproval>
</requests>
<requests name="Acknowledge1">
  <activationType>P</activationType>
  <ruleTemplate>Ack1Template</ruleTemplate>
</requests>
<requests name="Acknowledge2">
  <activationType>P</activationType>
  <ruleTemplate>Ack2Template</ruleTemplate>
</requests>
</routeNodes>
</documentType>

```

- **name:** This is the Document Type for Blanket Approve Mandatory Node Test.
- **Parent:** The parent Document Type is BlanketApproveTest. This Document Type inherits the policies that NotificationTest has.
- **description:** This Document Type is used to test the Blanket Approve Mandatory Node.
- **label:** This Document Type is recognized as the BlanketApproveMandatoryNodeTest type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is **_blank**.
- **active:** This Document Type is currently active. In other words, it is in use.
- **routePath:** The routing path for this Document Type is: AdHoc -> WorkflowDocument -> WorkflowDocument2 -> Acknowledge1 -> Acknowledge2.
- **routeNode:** Based on the routePath, there are five nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node **AdHoc**.
 - The next node in the routing for this Document Type is **WorkflowDocument**. On request, the node is activated, applies the rules in rule template, **WorkflowDocumentTemplate**, and sets the mandatory route as true. In other words, the document must route through this node.
 - The next node in the routing for this Document Type is **WorkflowDocument2**. On request, the node is activated, applies the rules in rule template, **WorkflowDocument2Template**, and sets the mandatory route as **true**. In other words, the document must route through this node.
 - The next node in the routing for this Document Type is **Acknowledge1**. On request, the node is activated and applies the rules in rule template, **Ack1Template**.
 - The next node in the routing for this Document Type is **Acknowledge2**. On request, the node is activated and applies the rules in rule template, **Ack2Template**.

Figure 12.4. Blanket Approve Mandatory Test



SaveActionEventTest

```

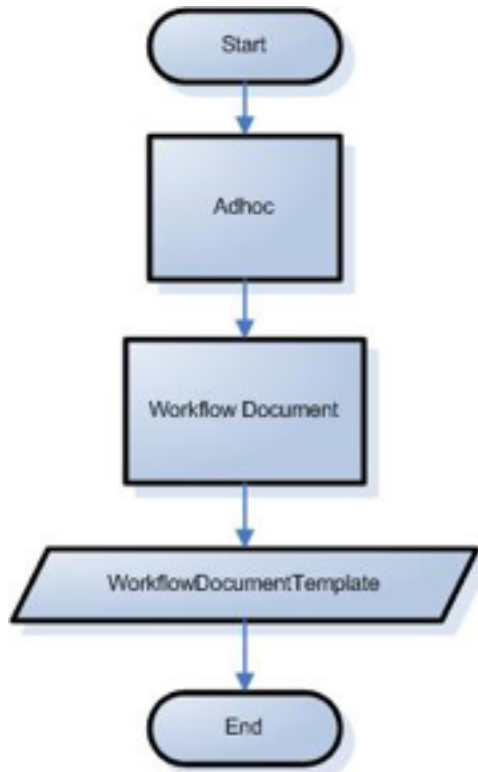
<documentType>
  <name>SaveActionEventTest</name>
  <description>SaveActionEventTest</description>
  <label>SaveActionEventTest</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superGroupName namespace="KR-WKFLW" >TestWorkgroup</superGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <policies>
    <policy>
      <name>DEFAULT_APPROVE</name>
      <value>>false</value>
    </policy>
  </policies>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="WorkflowDocument" />
      <requests name="WorkflowDocument" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
    </start>
    <requests name="WorkflowDocument">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
    </requests>
  </routeNodes>
</documentType>

```

- **name:** This is the Document Type for Save Action Event Test.
- **description:** This Document Type is used to test the Blanket Approve Mandatory Node.
- **label:** This Document Type is recognized as the SaveActionEventTest type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is **_blank**.
- **active:** This Document Type is currently active. In other words, it is in use.
- **Policies** for this Document Type: The DEFAULT_APPROVE policy is set **false** by default. In other words, the default approve action on this type of document is NOT to approve it.
- **routePath:** The routing path for this Document Type is: AdHoc -> WorkflowDocument.
- **routeNode:** Based on the routePath, there are two nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node **AdHoc**.

- The next node in the routing for this Document Type is **WorkflowDocument**. On request, the node is activated and applies the rules in rule template **WorkflowDocumentTemplate**.

Figure 12.5. Save Action Event Test



SaveActionEventTestNonInitiator

```

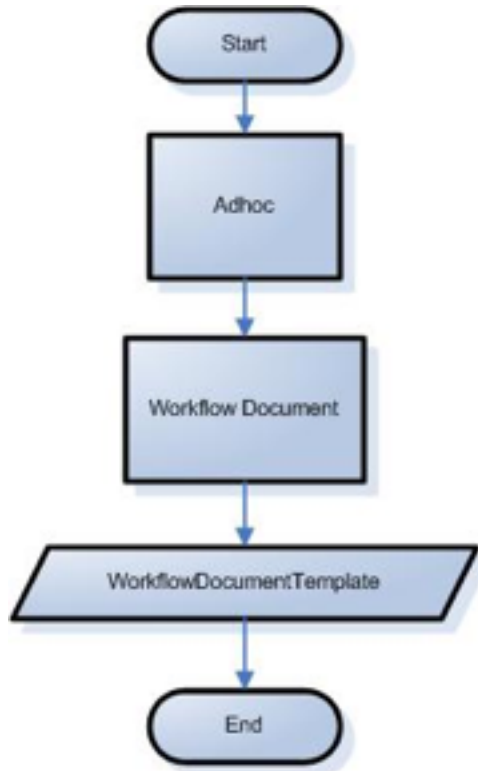
<documentType>
  <name>SaveActionEventTestNonInitiator</name>
  <description>SaveActionEventTest With No Initiator Only Save Required</description>
  <label>SaveActionEventTestNonInitiator</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <policies>
    <policy>
      <name>DEFAULT_APPROVE</name>
      <value>>false</value>
    </policy>
    <policy>
      <name>INITIATOR_MUST_SAVE</name>
      <value>>false</value>
    </policy>
  </policies>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="WorkflowDocument" />
      <requests name="WorkflowDocument" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
  
```



```
</start>
<requests name="WorkflowDocument">
  <activationType>P</activationType>
  <ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
</requests>
</routeNodes>
</documentType>
```

- **name:** This is the Document Type for Save Action Event Test Non Initiator.
- **description:** This Document Type is used to test the saving of an action event by non-initiator.
- **label:** This Document Type is recognized as the SaveActionEventTestNonInitiator type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is **_blank**.
- **active:** This Document Type is currently active. In other words, it is in use.
- **Policies** for this Document Type:
 - The DEFAULT_APPROVE policy is set **false** by default. In other words, the default approve action on this type of document is NOT to approve it.
 - The INITIATOR_MUST_SAVE policy is set **false** by default. In other words, the initiator does NOT have to save the document for the non-initiator to save the actions on it.
- **routePath:** The routing path for this Document Type is: AdHoc -> WorkflowDocument.
- **routeNode:** Based on the routePath, there are two nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node **AdHoc**.
 - The next node in the routing for this Document Type is **WorkflowDocument**. On request, the node is activated and applies the rules in rule template, **WorkflowDocumentTemplate**.

Figure 12.6. Save Action Even Test: Non-Initiator



TakeWorkgroupAuthorityDoc

```

<documentType>
  <name>TakeWorkgroupAuthorityDoc</name>
  <description>TakeWorkgroupAuthority Action Test</description>
  <label>TakeWorkgroupAuthorityDoc</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW" >TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW" > TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <policies>
    <policy>
      <name>DEFAULT_APPROVE</name>
      <value>>false</value>
    </policy>
  </policies>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="WorkgroupByDocument" />
      <requests name="WorkgroupByDocument" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
    </start>
    <requests name="WorkgroupByDocument">
      <activationType>P</activationType>
      <ruleTemplate>WorkgroupByDocument</ruleTemplate>
    </requests>
  </routeNodes>
</documentType>
  
```

- **name:** This is the Document Type for Take Workgroup Authority Doc.
- **description:** This Document Type is used to decide authorized workgroups by Document Type.
- **label:** This Document Type is recognized as the TakeWorkgroupAuthorityDoc type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is **_blank**.
- **active:** This Document Type is currently active. In other words, it is in use.
- **Policies** for this Document Type: The DEFAULT_APPROVE policy is set **false** by default. In other words, the default approve action on this type of document is NOT to approve it.
- **routePath:** The routing path for this Document Type is: AdHoc -> WorkflowDocument.
- **routeNode:** Based on the routePath, there are two nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node **AdHoc**.
 - The next node in the routing for this Document Type is **WorkflowDocument**. On request, the node is activated and applies the rules in rule template, **WorkflowDocumentTemplate**.

Figure 12.7. Take Workgroup Authority



MoveSequentialTest

```

<documentType>
  <name>MoveSequentialTest</name>
  <parent>BlanketApproveTest</parent>
  <description>Move Sequential Test</description>
  <label>Move Sequential Test</label>

  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="WorkflowDocument" />
      <requests name="WorkflowDocument" nextNode="WorkflowDocument2" />
      <requests name="WorkflowDocument2" nextNode="Acknowledge1" />
      <requests name="Acknowledge1" nextNode="Acknowledge2" />
      <requests name="Acknowledge2" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
    </start>
    <requests name="WorkflowDocument">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
    </requests>
    <requests name="WorkflowDocument2">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocument2Template</ruleTemplate>
    </requests>
    <requests name="Acknowledge1">
  
```

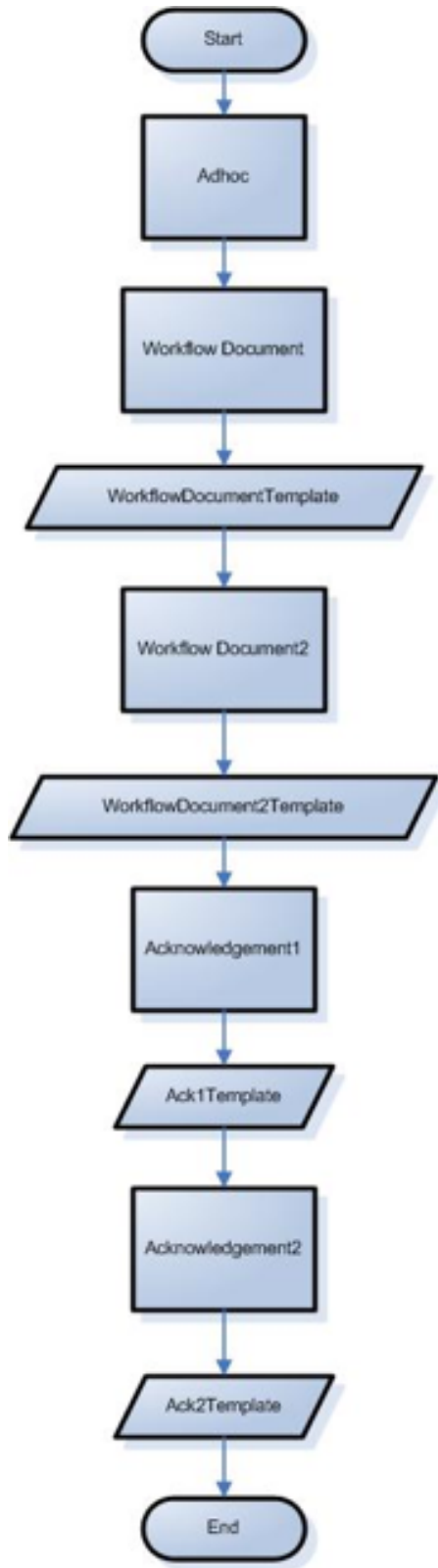
```

    <activationType>P</activationType>
    <ruleTemplate>Ack1Template</ruleTemplate>
  </requests>
  <requests name="Acknowledge2">
    <activationType>P</activationType>
    <ruleTemplate>Ack2Template</ruleTemplate>
  </requests>
</routeNodes>
</documentType>

```

- **name:** This is the Document Type for Move Sequential Test.
- **Parent:** The parent Document Type is BlanketApproveTest. This Document Type inherits the policies that BlanketApproveTest has.
- **description:** This Document Type is used to test Move Sequence.
- **label:** This Document Type is recognized as MoveSequentialTest type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is **_blank**.
- **active:** This Document Type is currently active. In other words, it is in use.
- **routePath:** The routing path for this Document Type is: AdHoc -> WorkflowDocument -> WorkflowDocument2 -> Acknowledge1 -> Acknowledge2.
- **routeNode:** Based on the routePath, there are five nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node **AdHoc**.
 - The next node in the routing for this Document Type is **WorkflowDocument**. On request, the node is activated, applies the rules in rule template, **WorkflowDocumentTemplate**, and sets the mandatory route as **true**. In other words, the document must route through this node.
 - The next node in the routing for this Document Type is **WorkflowDocument2**. On request, the node is activated, applies the rules in rule template, **WorkflowDocument2Template**, and sets the mandatory route as **true**. In other words, the document must route through this node.
 - The next node in the routing for this Document Type is **Acknowledge1**. On request, the node is activated and applies the rules in rule template, **Ack1Template**.
 - The next node in the routing for this Document Type is **Acknowledge2**. On request, the node is activated and applies the rules in rule template, **Ack2Template**.

Figure 12.8. Move Sequential Test



MoveInProcessTest

```

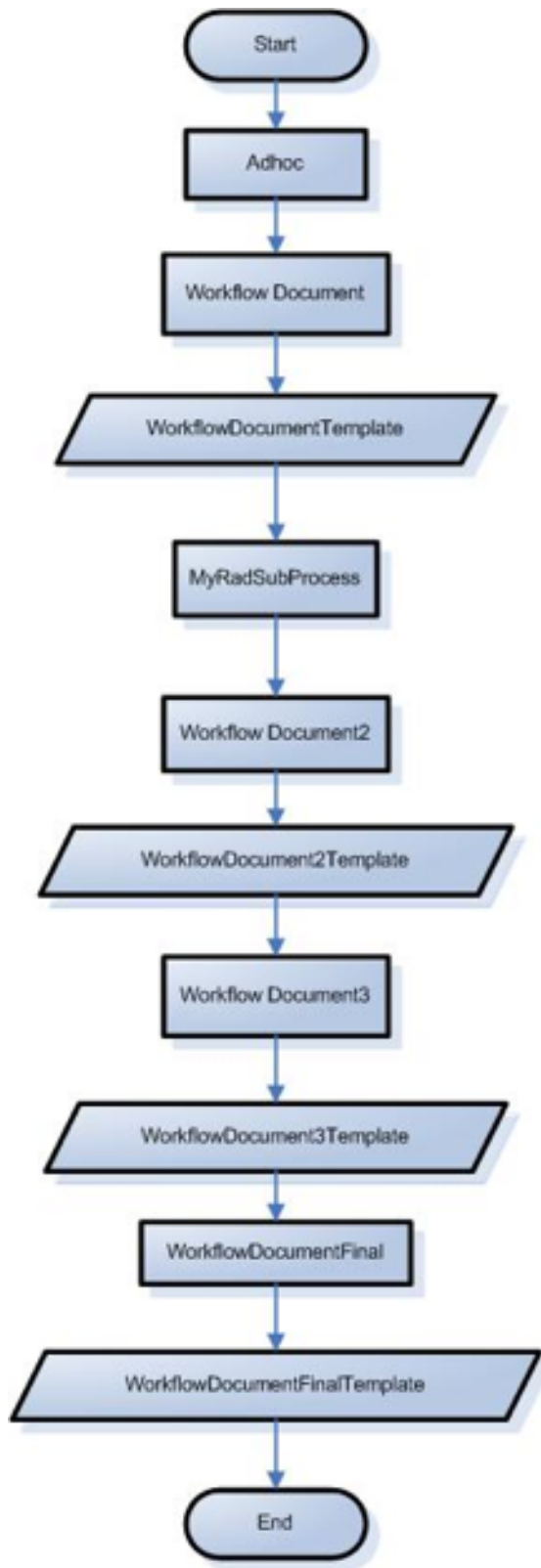
<documentType>
  <name>MoveInProcessTest</name>
  <parent>BlanketApproveTest</parent>
  <description>Move In Process Test</description>
  <label>Move In Process Test</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="WorkflowDocument" />
      <requests name="WorkflowDocument" nextNode="MyRadSubProcess" />
      <process name="MyRadSubProcess" nextNode="WorkflowDocumentFinal" />
      <requests name="WorkflowDocumentFinal" />
    </routePath>
    <routePath processName="MyRadSubProcess" initialNode="WorkflowDocument2">
      <requests name="WorkflowDocument2" nextNode="WorkflowDocument3" />
      <requests name="WorkflowDocument3" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
    </start>
    <requests name="WorkflowDocument">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
    </requests>
    <requests name="MyRadSubProcess" />
    <requests name="WorkflowDocument2">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocument2Template</ruleTemplate>
    </requests>
    <requests name="WorkflowDocument3">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocument3Template</ruleTemplate>
    </requests>
    <requests name="WorkflowDocumentFinal">
      <activationType>P</activationType>
      <ruleTemplate>WorkflowDocumentFinalTemplate</ruleTemplate>
    </requests>
  </routeNodes>
</documentType>

```

- **name:** This is the Document Type for Move In Process Test.
- **Parent:** The parent Document Type for this Document Type is BlanketApproveTest. This Document Type inherits the policies that BlanketApproveTest has.
- **description:** This Document Type is used to test Move In Process.
- **label:** This Document Type is recognized as the MoveInProcessTest type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is **_blank**.

- **active:** This Document Type is currently active. In other words, it is in use.
- **routePath:** The routing path for this Document Type is: AdHoc -> WorkflowDocument -> MyRadSubProcess -> WorkflowDocument2 -> WorkflowDocument3 -> WorkflowDocumentFinal. There is a sub-process MyRadSubProcess in this path.
- **routeNode:** As can be seen from the routePath, there are five nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node **AdHoc**.
 - The next node in the routing for this Document Type is **WorkflowDocument**. On request, the node is activated, applies the rules in rule template, **WorkflowDocumentTemplate**, and initiates a sub process MyRadSubProcess.
 - The next node in MyRadSubProcess for this Document Type is **WorkflowDocument2**. On request, the node is activated and applies the rules in rule template, **WorkflowDocument2Template**.
 - The next node in MyRadSubProcess for this Document Type is **WorkflowDocument3**. On the request, the node is activated and applies the rules in rule template, **WorkflowDocument3Template**.
 - The next node in the routing for this Document Type is **WorkflowDocumentFinal**. On request, the node is activated and applies the rules in rule template **WorkflowDocumentFinalTemplate**.

Figure 12.9. Move In Process Test



AdhocRouteTest

```

<documentType>
<name>AdhocRouteTest</name>
<description>AdhocRouteTest</description>
<label>AdhocRouteTest</label>

<postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
<superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
<blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
<defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
<docHandler>_blank</docHandler>
<active>true</active>
<routePaths>
<routePath>
<start name="AdHoc" nextNode="One" />
<requests name="One" />
</routePath>
</routePaths>
<routeNodes>
<start name="AdHoc">
<activationType>P</activationType>
</start>
<requests name="One">
<activationType>S</activationType>
<ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
</requests>
</routeNodes>
</documentType>

```

- **name:** This is the Document Type for Adhoc Route Test.
- **description:** This Document Type is used to test Ad Hoc Route.
- **label:** This Document Type is recognized as the AdhocRouteTest type.
- **postProcessorName:** the postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is **_blank**.
- **active:** This Document Type is currently active. In other words, it is in use.
- **routePath:** The routing path for this Document Type is: AdHoc -> One.
- **routeNode:** Based on the routePath, there are two nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node **AdHoc**.
 - The next node in the routing for this Document Type is **One**. On request, the node is activated by the type **S** and applies the rules in rule template, **WorkflowDocumentTemplate**.

Figure 12.10. Adhoc Route Test**PreApprovalTest**

```

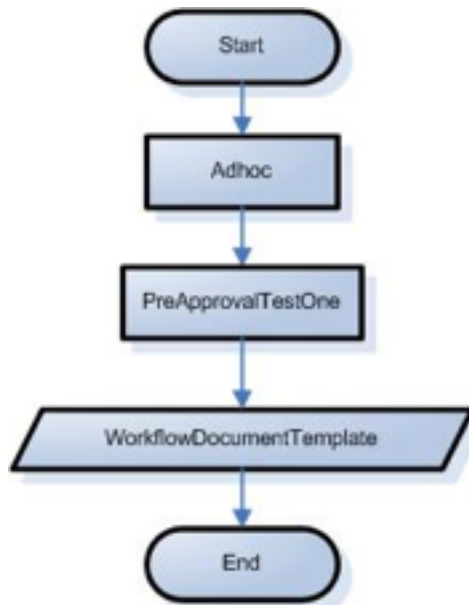
<documentType>
  <name>PreApprovalTest</name>
  <description>PreApprovalTest</description>
  <label>PreApprovalTest</label>
  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="PreApprovalTestOne" />
      <requests name="PreApprovalTestOne" />
    </routePath>
  </routePaths>
  <routeNodes>
    <start name="AdHoc">
      <activationType>P</activationType>
    </start>
    <requests name="PreApprovalTestOne">
      <activationType>S</activationType>
      <ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
    </requests>
  </routeNodes>
</documentType>

```

- **name:** This is the Document Type for PreApprovalTest.
- **description:** This Document Type is used to test Pre-Approval.
- **label:** This Document Type is recognized as the PreApprovalTest type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.

- **blanketApproveGroupName**: The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName**: The members of the TestWorkgroup will receive an exception notice for documents of this Document Type. • **docHandler**: The Doc Handler for this type of document is **_blank**.
- **active**: This Document Type is currently active. In other words, it is in use.
- **routePath**: The routing path for this Document Type is: AdHoc -> PreApprovalTestOne.
- **routeNode**: Based on the routePath, there are two nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node **AdHoc**.
 - The next node in the routing for this Document Type is **PreApprovalTestOne**. On request, the node is activated by the type **S** and applies the rules in rule template, **WorkflowDocumentTemplate**.

Figure 12.11. PreApproval Test



VariablesTest

```

<documentType>
  <name>VariablesTest</name>
  <description>VariablesTest</description>
  <label>VariablesTest</label>

  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <routePaths>
    <routePath>
      <start name="AdHoc" nextNode="setStartedVar" />
      <simple name="setStartedVar" nextNode="setCopiedVar" />
      <simple name="setCopiedVar" nextNode="PreApprovalTestOne" />
      <requests name="PreApprovalTestOne" nextNode="setEndedVar" />
    
```

```

        <simple name="setEndedVar" nextNode="setGoogleVar" />
        <simple name="setGoogleVar" nextNode="setXPathVar" />
        <simple name="setXPathVar" nextNode="resetStartedVar" />
        <simple name="resetStartedVar" nextNode="logNode" />
        <simple name="logNode" nextNode="logNode2" />
        <simple name="logNode2" />
    </routeProvider>
</routeProvider>
</routePaths>
<routeNodes>
    <start name="AdHoc">
        <activationType>P</activationType>
    </start>
    <simple name="setStartedVar">
        <type>org.kuali.rice.kew.engine.node.var.SetVarNode</type>
        <name>started</name>
        <value>startedVariableValue</value>
    </simple>
    <simple name="setCopiedVar">
        <type>org.kuali.rice.kew.engine.node.var.SetVarNode</type>
        <name>copiedVar</name>
        <value>var:started</value>
    </simple>
    <requests name="PreApprovalTestOne">
        <activationType>S</activationType>
        <ruleTemplate>WorkflowDocumentTemplate</ruleTemplate>
    </requests>
    <simple name="setEndedVar">
        <type>org.kuali.rice.kew.engine.node.var.SetVarNode</type>
        <name>ended</name>
        <value>endedVariableValue</value>
    </simple>
    <simple name="setGoogleVar">
        <type>org.kuali.rice.kew.engine.node.var.SetVarNode</type>
        <name>google</name>
        <value>url:http://google.com</value>
    </simple>
    <simple name="setXPathVar">
        <type>org.kuali.rice.kew.engine.node.var.SetVarNode</type>
        <name>xpath</name>
        <value>xpath:concat(local-name(//documentContent), $ended)</value>
    </simple>
    <simple name="resetStartedVar">
        <type>org.kuali.rice.kew.engine.node.var.SetVarNode</type>
        <name>started</name>
        <value>aNewStartedVariableValue</value>
    </simple>
    <simple name="logNode">
        <type>org.kuali.rice.kew.engine.node.LogNode</type>
        <message>var:xpath</message>
    </simple>
    <simple name="logNode2">
        <type>org.kuali.rice.kew.engine.node.LogNode</type>
        <level>Error</level>
        <log>Custom.Logger.Name</log>
        <message>THAT'S ALL FOLKS</message>
    </simple>
</routeNodes>
</documentType>

```

- **name:** This is the Document Type for VariablesTest.
- **description:** This Document Type is used to test Variables.
- **label:** This Document Type is recognized as the VariablesTest type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.

- **defaultExceptionGroupName**: The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler**: The Doc Handler for this type of document is **_blank**.
- **active**: This Document Type is currently active. In other words, it is in use.
- **routePath**: The routing path for this Document Type is: AdHoc -> setStartedVar -> setCopiedVar -> preApprovalTestOne -> setEndedVar -> setGoogleVar -> setXPathVar -> resetStartedVar -> logNode -> logNode2.
- **routeNode**: Based on the routePath, there are ten nodes in the routing of this Document Type:
 - The starting node for this Document Type is **AdHoc**. On the initiation of a document of this type, the postProcessor in KEW activates the node **AdHoc**.
 - The next node in the routing for this Document Type is **setStartedVar**.
 - Its type is **org.kuali.rice.kew.engine.node.var.SetVarNode**
 - Its name is **started**.
 - Its value is **startedVariableValue**.
 - The next node in the routing for this Document Type is **setCopiedVar**.
 - Its type is **org.kuali.rice.kew.engine.node.var.SetVarNode**.
 - Its name is **copiedVar**.
 - The value that it is copying is var:**started**.
 - The next node in the routing for this Document Type is **preApprovalTestOne**. On request, the node is activated by the type **S** and applies the rules in rule template **WorkflowDocumentTemplate**.
 - The next node in the routing for this Document Type is **setEndedVar**.
 - Its type is **org.kuali.rice.kew.engine.node.var.SetVarNode**.
 - Its name is **ended**.
 - Its value is **endedVariableValue**.
 - The next node in the routing for this Document Type is **setGoogleVar**.
 - Its type is **org.kuali.rice.kew.engine.node.var.SetVarNode**.
 - Its name is **google**. It links to **http://google.com**.
 - The next node in the routing for this Document Type is **setXPathVar**.
 - Its type is **org.kuali.rice.kew.engine.node.var.SetVarNode**.
 - Its name is **xpath**.
 - It adds **//documentContent** to the current path.

- The next node in the routing for this Document Type is **resetStartedVar**.
 - Its type is **org.kuali.rice.kew.engine.node.var.SetVarNode**.
 - Its name is **started**.
 - It resets the started node at a new node, **aNewStartedVariableValue**.
- The next node in the routing for this Document Type is **logNode**.
 - Its type is **org.kuali.rice.kew.engine.node.LogNode**.
 - It sends a message about the xpath of the variables at **var:xpath**.
- The next node in the routing for this Document Type is **logNode2**.
 - Its type is **org.kuali.rice.kew.engine.node.LogNode**.
 - Its level is **ErRoR**.
 - It opens the log **Custom.Logger.Name**.
 - It returns a message **THAT'S ALL FOLKS**.

Figure 12.12. Variables Test



SUApproveDocumentNotifications


```
<documentType>
  <name>SUApproveDocumentNotifications</name>
  <parent>SUApproveDocument</parent>
  <description>SUApproveDocumentNotifications</description>
  <label>SUApproveDocumentNotifications</label>

  <postProcessorName>org.kuali.rice.kew.postprocessor.DefaultPostProcessor</postProcessorName>
  <superUserGroupName namespace="KR-WKFLW" >TestWorkgroup</superUserGroupName>
  <blanketApproveGroupName namespace="KR-WKFLW">TestWorkgroup</blanketApproveGroupName>
  <defaultExceptionGroupName namespace="KR-WKFLW"> TestWorkgroup</defaultExceptionGroupName>
  <docHandler>_blank</docHandler>
  <active>true</active>
  <policies>
    <policy>
      <name>SEND_NOTIFICATION_ON_SU_APPROVE</name>
      <value>true</value>
    </policy>
  </policies>
</documentType>
```

- **name:** This is the Document Type for SuperUser Approve Document Notifications.
- **description:** This Document Type is used to test the SuperUser Approve Document Notifications.
- **label:** This Document Type is recognized as the SUApproveDocumentNotifications type.
- **postProcessorName:** The postProcessor for this Document Type is **org.kuali.rice.kew.postprocessor.DefaultPostProcessor**.
- **superUserGroupName:** The super users for this Document Type are members of the TestWorkgroup.
- **blanketApproveGroupName:** The members of the TestWorkgroup have blanketApproval right on this type of document.
- **defaultExceptionGroupName:** The members of the TestWorkgroup will receive an exception notice for documents of this Document Type.
- **docHandler:** The Doc Handler for this type of document is **_blank**.
- **active:** This Document Type is currently active. In other words, it is in use.
- There is just one **policy** for this Document Type: The SEND_NOTIFICATION_ON_SU_APPROVE policy is set true by default. In other words, notifications will be automatically sent on SuperUser's approval.

Document Type Authorizer

The Document Type Authorizer is a component that gets called during the routing process to perform authorization checks. Applications can customize this component, for example to introduce custom role qualifiers or permission details, on a per-doctype basis by registering a custom

```
org.kuali.rice.kew.framework.document.security.DocumentTypeAuthorizer
```

implementation.

The DocumentTypeAuthorizer will be called to make the following checks:

- canInitiate
- canBlanketApprove

- canCancel
- canRecall
- canSave
- canRoute
- canSuperUserApproveSingleActionRequest
- canSuperUserApproveDocument
- canSuperUserDisapproveDocument

Document Type Policies

Document Type Policies affect workflow routing behavior.

Current Document Type polices:

- DISAPPROVE
- DEFAULT_APPROVE
- DOCUMENT_STATUS_POLICY
- INITIATOR_MUST_ROUTE
- INITIATOR_MUST_SAVE
- INITIATOR_MUST_CANCEL
- INITIATOR_MUST_BLANKET_APPROVE
- LOOK_FUTURE
- SEND_NOTIFICATION_ON_SU_APPROVE
- SUPPORTS_QUICK_INITIATE
- NOTIFY_ON_SAVE
- blanketApprovePolicy
- ALLOW_SU_POST_PROCESSOR_OVERRIDE
- NOTIFY_COMPLETED_ON_RETURN
- NOTIFY_PENDING_ON_RETURN
- RECALL_NOTIFICATION
- ALLOW_SU_FINAL_APPROVAL
- SEND_NOTIFICATION_ON_SU_DISAPPROVE

Document Type Policies defined in the Document Type XML have this structure:

```
<documentType>
  <name>...</name>
  <policies>
    <policy>
      <name>DEFAULT_APPROVE</name>
      <value>>true</value>
    </policy>
    <policy>
      <name>LOOK_FUTURE</name>
      <value>>false</value>
    </policy>
    <policy>
      <name>DOCUMENT_STATUS_POLICY</name>
      <stringValue>APP</stringValue>
    </policy>
  </policies>
</documentType>
```

DISAPPROVE

The **DISAPPROVE** policy determines whether a document will discontinue routing (transactions). When a document has been disapproved, the document initiator and previous approvers will receive notice of this disapproval action.

DEFAULT_APPROVE

The **DEFAULT_APPROVE** policy determines whether a document will continue processing with or without any approval requests. If a document is set to have no approval requests, its put into exception routing. Then, the document will continue to route to the exception workgroup associated with the last route node or to the workgroup defined in the **defaultExceptionWorkgroupname**.

DOCUMENT_STATUS_POLICY

The **DOCUMENT_STATUS_POLICY** policy sets whether to display the KEW Route Status, the Application Document Status, or Both in the Route Log. Valid policy values are: **KEW**, **APP**, or **BOTH**.

The set of valid statuses for a given document type may be defined. If defined, only those values are allowed as valid statuses. These will also be used to populate a multi-select box on the doc search screen if this doc type is selected (see [Customizing Document Search: Application Document Status](#)). If not defined, any string with a length of up to 64 characters may be used, and a text input field is used on the doc search screen. An example configuration follows.

```
<validApplicationStatuses>
  <status>Initiated</status>
  <status>Validated</status>
  <status>Awaiting Content Approval</status>
  <status>Org Review</status>
  <status>Approved</status>
  ...
</validApplicationStatuses>
```

Additionally, the valid statuses may be grouped into named categories for display and search purposes. If defined, these categories will display in doc search (again, if the document type is selected) under the application document status multi-select as headings under which the individual statuses are grouped. These categories can be selected as well, which has an equivalent effect on the search to individually selecting all of the statuses within the category (see [Customizing Document Search: Application Document Status](#)). Note that not all statuses need be grouped within categories, as demonstrated below by the "Approved" status below.

```
<validApplicationStatuses>
  <category name="Pre-Submit">
    <status>Initiated</status>
    <status>Validated</status>
  </category>
  <category name="In Process">
    <status>Awaiting Content Approval</status>
    <status>Org Review</status>
  </category>
  <status>Approved</status>
  ...
</validApplicationStatuses>
```

In the process definition section, automatic status updates may be assigned to occur on route node transition. Please see the section on the routePaths in this guide.

INITIATOR_MUST_ROUTE

The **INITIATOR_MUST_ROUTE** policy sets the rule that the user who initiates the document must route it.

INITIATOR_MUST_SAVE

The **INITIATOR_MUST_SAVE** policy sets the rule that the user who initiated the document will be the only one authorized to **save** the document.

INITIATOR_MUST_CANCEL

The **INITIATOR_MUST_CANCEL** policy sets the rule that the user who initiated the document will be the only one authorized to **cancel** the document.

INITIATOR_MUST_BLANKET_APPROVE

The **INITIATOR_MUST_BLANKET_APPROVE** policy sets the rule that the user who initiated the document is the only one authorized to **blanket approve** the document.

LOOK_FUTURE

The **LOOK_FUTURE** policy determines whether the document can be brought into a simulated route from the route log. This policy simulates where the document would end up if it completed the route.

SEND_NOTIFICATION_ON_SU_APPROVE

The **SEND_NOTIFICATION_ON_SU_APPROVE** policy indicates to KEW that it is to send a notification on SuperUser approval.

SUPPORTS_QUICK_INITIATE

The **SUPPORTS_QUICK_INITIATE** policy indicates whether the Document Type is displayed on the Quick Links, so that users can quickly initiate instances of the document.

NOTIFY_ON_SAVE

The **NOTIFY_ON_SAVE** policy indicates whether a notification should be sent in when a save action is applied to this Document Type.

blanketApprovePolicy

The **blanketApprovePolicy** policy indicates who can **blanket approve** a workflow document. Its values are either ANY or NONE.

- ANY means that anybody can blanket approve the document.
- NONE means that no one can blanket approve the document.

Alternatively, the configuration of the document can be set up to specify a `blanketApproveWorkgroupName`. `blanketApproveWorkgroupName` indicates that members of that workgroup can blanket approve the document. You can specify either `blanketApprovePolicy` OR `blanketApproveWorkgroupName` in the Document Type.

Since the blanket approve policy is not a true/false policy (like the others), it is specified as an element in the Document Type XML:

```
<documentType>
  <name>...</name>
  .
  .
  .
  <blanketApprovePolicy>NONE</blanketApprovePolicy>
</documentType>
```

ALLOW_SU_POST_PROCESSOR_OVERRIDE

There is currently the ability to override the "Perform Post Processor Logic" on the "Super User Action on Action Requests" page. This functionality is configurable by document type and as such allows for inheritance.

By default, the `ALLOW_SU_POST_PROCESSOR_OVERRIDE` it's set to true. The checkbox appears on the super user screen as:

Figure 12.13. Super User Action on Requests

Super User Action on Action Requests

APPROVE Requested of employee, employee	
Request Date	04:17 PM 08/02/2010
Request Status	ACTIVE
Responsibility	Supervisor Routing
Annotation	employee
Route Level	TravelerApproval
Routing Priority	1
Responsibility Id	2024
Action Request Id	2377
Perform Post Processor Logic	<input checked="" type="checkbox"/>

In order to turn off the post processor check box, you would add this to the documentType definition:

```
<policies>
  <policy>
    <name>ALLOW_SU_POSTPROCESSOR_OVERRIDE</name>
    <value>>false</value>
  </policy>
```

```
</policies>
```

Recall From Routing

Three Document Type policies affect Recall behavior. These policies are defined in the respective the DocumentType XML. The following two policies apply to Return-To-Previous actions as well as Recall actions:

NOTIFY_COMPLETED_ON_RETURN - Default: false toggles whether to notify previous router log participants with FYIs when a document is recalled. This does not affect notifications to pending approvers which are always sent.

Example:

```
<policy>
  <name>NOTIFY_COMPLETED_ON_RETURN</name>
  <value>true</value>
</policy>
```

NOTIFY_PENDING_ON_RETURN - Default: true toggles whether to notify pending approvers with FYIs when a document is recalled. This does not affect notifications to prior approvers.

Example:

```
<policy>
  <name>NOTIFY_PENDING_ON_RETURN</name>
  <value>true</value>
</policy>
```

The following policy is Recall-specific:

RECALL_NOTIFICATION - Default: false/none

Example:

```
<policy>
  <name>RECALL_NOTIFICATION</name>
  <value>true</value>
  <recipients xmlns:r="ns:workflow/Rule" xsi:schemaLocation="ns:workflow/Rule resource:Rule"
  xmlns:dt="ns:workflow/DocumentType">
    <r:principalName>quickstart</r:principalName>
    <r:user>quickstart</r:user>
    <role namespace="KR-SYS" name="Technical Administrator"/>
  </recipients>
</policy>
```

ALLOW_SU_FINAL_APPROVAL

Setting this policy to false disallows Super User approval on final nodes of the document.

```
<policies>
  <policy>
```

```
<name>ALLOW_SU_FINAL_APPROVAL</name>
<value>>false</value>
</policy>
</policies>
```

SEND_NOTIFICATION_ON_SU_DISAPPROVE

By default, acknowledgements are not sent on Super User Disapproval like they are for normal Disapprove actions. This policy can be used to enable sending of acknowledgements upon Super User Disapproval.

```
<policies>
  <policy>
    <name>SEND_NOTIFICATION_ON_SU_DISAPPROVE</name>
    <value>>true</value>
  </policy>
</policies>
```

Inheritance

Document Types can specify a parent Document Type. This allows them to be included in a Document Type hierarchy from which certain behavior can be inherited from their parent Document Type.

Inheritable Fields

These fields are inherited:

- **superUserGroupName:** Indicates members of the workgroup who can perform SuperUser actions on the document
- **blanketApproveGroupName:** Indicates members of the workgroup that can blanket approve the document.
- **notificationFromAddress:** Sends a notice to the sender when the transfer of the document is completed.
- **messageEntity:** A head and body of the message.
- **policies:** Indicates a set of rule(s) applied in the document. For each policy, True means policy DOES apply, False means policy does NOT apply.
- **searchable attributes:** Constraint(s) assigned as the searchable criteria for a document.
- **route paths/route nodes:** Designated traveling points before the document reaches its destination in a routing process.

Special notes about inheritance:

1. **Policies:** In the Policies section, there are multiple Document Type policies (INITAIATOR_MUST_ROUTE, DEFAULT_APPROVE, etc). Each policy can be overridden on an **individual basis**. In contrast to the route path, there is no need to override the entire **policies** section for a Document Type. For more detailed information about Document Type policies, please see Document Type Policies (above) in this document.
2. **Route paths/ route nodes:** To override the route path and route node definitions of a parent Document Type, you must override ALL route node and route path definitions. You cannot inherit and use just part of a route path; it's all or nothing.

Document Type hierarchy and the Rules Engine

The Rules Engine follows these rules to determine its rule evaluation set for a Document Type at a particular node:

1. The Rules Engine looks at the Rule Template name of the current node and selects all rules with that template and that document's Document Type. It adds those rules to the rule evaluation set.
2. If the Document Type has a parent Document Type, it selects all rules with that template and that parent Document Type and adds those to the rule evaluation set.
3. Its repeats step two until it reaches the root of the Document Type hierarchy.
4. The final rule evaluation set includes all of these rules.

Defining Workflow Processes Using PeopleFlow - a new feature in KEW

PeopleFlow is our Kualu Rice instantiation of the "maps" concept included in the original Coeus. For all intents and purposes it's a prioritized list of people to send requests to. PeopleFlow gives you a new type of request activation strategy called "priority-parallel" to activate requests generated from a PeopleFlow in the appropriate order. Essentially, it's like a mini people-based workflow that doesn't require you to specify a KEW node in the document type for each individual who might need to approve or be notified. You can define "Stops" in a PeopleFlow, where everything in the same stop proceeds in parallel, but all must be done within the stop before proceeding to the next stop.

You can call/execute a PeopleFlow from within a KEW workflow node directly, or you can invoke the Kualu Rules Management System (KRMS) engine from an application and any PeopleFlows that get selected during rule execution, defined in a KRMS agenda, will be called. In this way, you can integrate business rules across applications and workflows.

The same PeopleFlow that defines a routing order among a set of persons, groups or roles can be called by KRMS rules, with the KRMS rules defining which type of request to pass to the PeopleFlow (for example, an "approval" routing action or a "notification").

KRMS is also a new feature in Rice 2.0. See the KRMS Technical Guide for more information on KRMS.

You can define a PeopleFlow (simple workflow) via a maintenance document. See the KEW Users' Guide for additional details on defining a PeopleFlow.

Technical Information about PeopleFlow

(decide what needs to go here -- architecture, data model, api, troubleshooting, etc.?)

Chapter 13. Using the Workflow Document API

Overview

This document explains features of the workflow document API. There are two interfaces in KEW that allow you to create a document for delivery through workflow. The **WorkflowDocument** interface is designed to create a new document in the workflow system once an action has been taken, such as sending ad hoc requests. The **WorkflowInfo** interface is actually a convenience class for client applications that query workflow. Both classes assist with implementing connections to KEW.

WorkflowDocument

The process for this section of the API involves creating the initial **WorkflowDocument** using a constructor to create a new routable document in KEW. Once the object is defined, it initializes by loading an existing *routeHeaderId* or by constructing an empty document of a specified *documentType*. A number of methods can be invoked once initialization is complete and details of how those methods would be invoked are outlined primarily in the Java Documentation at <https://test.kuali.org/rice/rice-api-1.0-javadocs/>.

Document content methods modify the properties of a document's content. A specific case is *addAttributeDefinition()*, where a *WorkflowAttribute* is used to generate attribute document content that will be appended to the existing document content. Another case is adding a searchable attribute definition with the *addSearchableDefinition()* method. More information on the various constructors, methods, and objects relating to the *WorkflowDocument* class is available in the Java documentation found at <https://test.kuali.org/rice/rice-api-1.0-javadocs/org/kuali/rice/kew/service/WorkflowDocument.html>.

WorkflowInfo

This class is the second client interface to KEW. The first time this object is initialized, the client configuration is accessed to determine how to connect to KEW. Methods invoked from this class can grab the routing header information based on the *principalId*, or return a set of Action Requests for a document that's in route based on the *routeHeaderId*, the *nodeName* and the *principalId*. More information on the various constructors, methods, and objects relating to the *WorkflowInfo* class is available in the Java documentation found at <https://test.kuali.org/rice/rice-api-1.0-javadocs/org/kuali/rice/kew/service/WorkflowInfo.html>.

Chapter 14. Creating an eDocLite Application

Overview

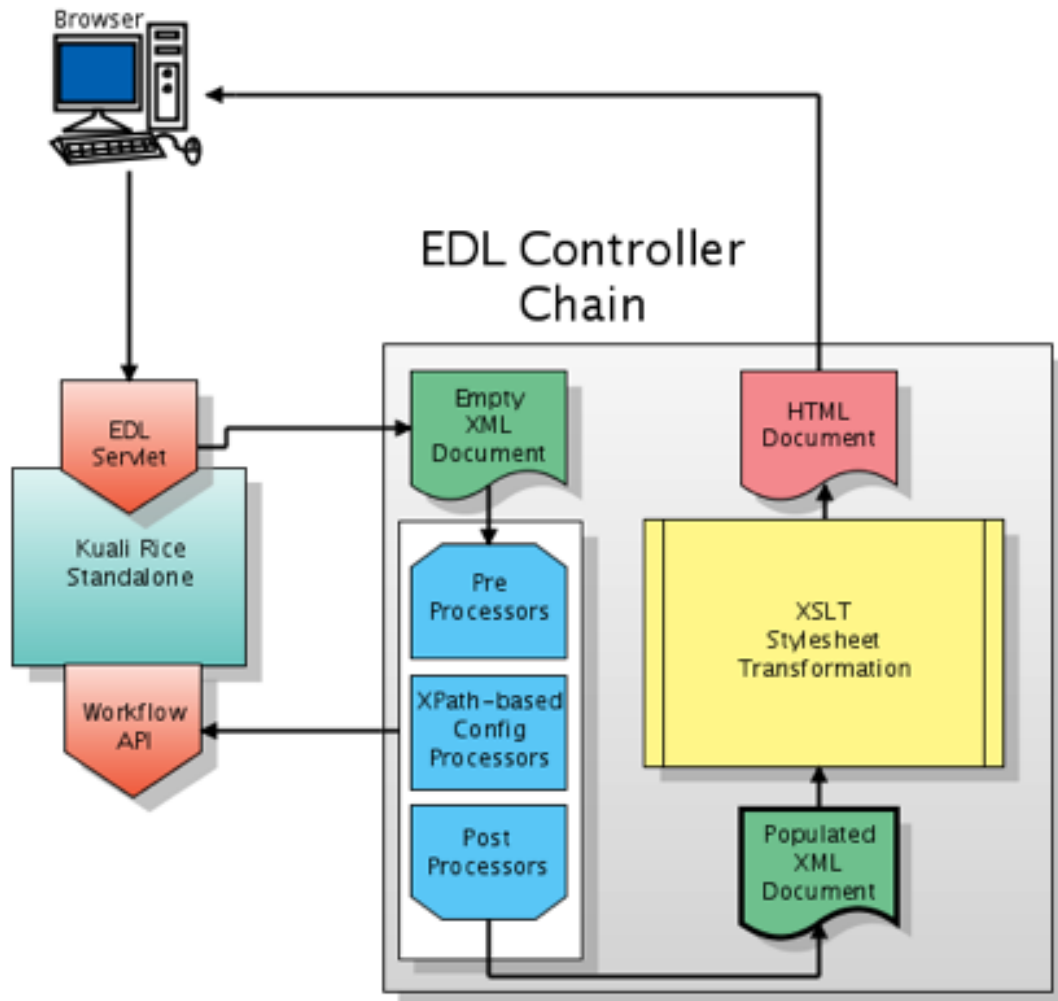
eDocLite is a simple, form-based system that is built into Quali Enterprise Workflow (KEW). It facilitates rapid development and implementation of simple documents and validation rules using XML. Use it for simple documents with simple route paths. You can integrate it with larger applications using a database layer post-processor component.

eDocLite uses an XSLT style sheet for custom presentation and XML to define form fields. The actual form display is called an EDL. This diagram shows how these objects are related:

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'.

Figure 14.1. EDL Controller Chain



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.

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

  <edl name="eDoc.Example1.Form" title="Example 1">
    <security />
    <createInstructions>** Questions with an asterisk are required.</createInstructions>
    <instructions>** Questions with an asterisk are required.</instructions>
    <validations />
    <attributes />
    <fieldDef name="userName" title="Full Name">
      <display>
        <type>text</type>
        <meta>
          <name>size</name>
          <value>40</value>
        </meta>
      </display>
      <validation required="true">
        <message>Please enter your full name</message>
      </validation>
    </fieldDef>
    <fieldDef name="rqstDate" title="Requested Date of Implementation:">
      <display>
        <type>text</type>
      </display>
      <validation required="true">
        <regex>^[0-1]?[0-9](/|-)[0-3]?[0-9](/|-)[1-2][0-9][0-9][0-9]$</regex>
        <message>Enter a valid date in the format mm/dd/yyyy.</message>
      </validation>
    </fieldDef>
    <fieldDef name="requestType" title="Request Type:">
      <display>
        <type>radio</type>
        <values title="New">New</values>
        <values title="Modification">Modification</values>
      </display>
      <validation required="true">
        <message>Please select a request type.</message>
      </validation>
    </fieldDef>
    <fieldDef attributeName="EDL.Campus.Example" name="campus" title="Campus:">
      <display>
        <type>select</type>
        <values title="IUB">IUB</values>
        <values title="IUPUI">IUPUI</values>
      </display>
      <validation required="true">
        <message>Please select a campus.</message>
      </validation>
    </fieldDef>
    <fieldDef name="description" title="Description of Request:">
      <display>
        <type>textarea</type>
        <meta>
          <name>rows</name>
          <value>5</value>
        </meta>
        <meta>
          <name>cols</name>
          <value>60</value>
        </meta>
      </display>
    </fieldDef>
  </edl>
</edoclite>
</data>
```

```

        <meta>
            <name>wrap</name>
            <value>hard</value>
        </meta>
    </display>
    <validation required="false" />
</fieldDef>
<fieldDef name="fundedBy" title="My research/sponsored program work is funded by NIH or NSF.">
    <display>
        <type>checkbox</type>
        <values title="My research/sponsored program work is funded by NIH or NSF.">nihnsf</values>
    </display>
</fieldDef>
<fieldDef name="researchHumans" title="My research/sponsored program work involves human subjects.">
    <display>
        <type>checkbox</type>
        <values title="My research/sponsored program work involves human subjects.">humans</values>
    </display>
</fieldDef>
</edl>
</eDocLite>
</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 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.

```

<!-- widgets is simply more xslt that contains common functionality that greatly simplifies html rendering

```

Creating an eDocLite Application

```
It is somewhat complicated but does not require changes or full understanding unless enhancements are required.
-->
<xsl:include href="widgets" />
<xsl:output indent="yes" method="html" omit-xml-declaration="yes" version="4.01" />

<!-- 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
values contained in workflow xml. Not all of these are used in this form but are shown because often they can
be useful
The ones prefixed with my-class are methods that are exposed by workflow to eDocLite .-->
<xsl:variable name="actionable" select="//documentContent/documentState/actionable" />
<xsl:variable name="docHeaderId" select="//documentContent/documentState/docId" />
<xsl:variable name="editable" select="//documentContent/documentState/editable" />
<xsl:variable name="globalReadOnly" select="//documentContent/documentState/editable != 'true'" />
<xsl:variable name="docStatus" select="//documentState/workflowDocumentState/status" />
<xsl:variable name="isAtNodeInitiated" select="my-class:isAtNode($docHeaderId, 'Initiated')" />
<xsl:variable name="isPastInitiated" select="my-class:isNodeInPreviousNodeList('Initiated', $docHeaderId)" />
<xsl:variable name="isUserInitiator" select="my-class:isUserInitiator($docHeaderId)" />
<!-- <xsl:variable name="workflowUser" select="my-class:getWorkflowUser().authenticationUserId().id()" /> This
has a unique implementation at IU -->
<xsl:param name="overrideMain" select="'true'" />

<!-- mainForm begins here. Execution of stylesheet begins here. It calls other templates which can call other
templates.
Position of templates beyond this point do not matter. -->
<xsl:template name="mainForm">
  <html xmlns="">
    <head>
      <script language="javascript" />
      <xsl:call-template name="htmlHead" />
    </head>
    <body onload="onPageLoad()">
      <xsl:call-template name="errors" />
      <!-- the header is useful because it tells the user whether they are in 'Editing' mode or 'Read
Only' mode. -->
      <xsl:call-template name="header" />
      <xsl:call-template name="instructions" />
      <xsl:variable name="formTarget" select="'eDocLite '" />
      <!-- validateOnSubmit is a javascript function (file: edoclitel.js) which supports edoclite forms
and can be somewhat complicated
but does not
require modification unless enhancements are required. -->
      <form action="{formTarget}" enctype="multipart/form-data" id="edoclite" method="post"
onsubmit="return validateOnSubmit(this)">
        <xsl:call-template name="hidden-params" />
        <xsl:call-template name="mainBody" />
        <xsl:call-template name="notes" />
        <br />
        <xsl:call-template name="buttons" />
        <br />
      </form>
      <xsl:call-template name="footer" />
    </body>
  </html>
</xsl:template>

<!-- mainBody template begins here. It calls other templates which can call other templates. Position of
templates do not matter. -->
<xsl:template name="mainBody">
  <!-- to debug, or see values of previously created variables, one can use the following format.
for example, uncomment the following line to see value of $docStatus. It will be rendered at the top
of the main body form. -->
  <!-- $docStatus=<xsl:value-of select="$docStatus" /> -->
  <!-- rest of this all is within the form table -->
  <table xmlns="" align="center" border="0" cellpadding="0" cellspacing="0" class="bord-r-t" width="80%">
    <tr>
      <td align="left" border="3" class="thnormal" colspan="1">
<br />
<h3>
My Page
<br />
EDL EDoclite Example
</h3>
<br />
</td>
      <td align="center" border="3" class="thnormal" colspan="2">
<br />

```

```

<h2>eDocLite Example 1 Form</h2></td>
</tr>
<tr>
  <td class="headercell15" colspan="100%">
<b>User Information</b>
</td>
</tr>
<tr>
  <td class="thnormal">
    <xsl:call-template name="widget_render">
      <xsl:with-param name="fieldName" select="'userName'" />
      <xsl:with-param name="renderCmd" select="'title'" />
    </xsl:call-template>
    <font color="#ff0000">*</font>
  </td>
  <td class="datacell">
    <xsl:call-template name="widget_render">
      <xsl:with-param name="fieldName" select="'userName'" />
      <xsl:with-param name="renderCmd" select="'input'" />
      <xsl:with-param name="readOnly" select="'$isPastInitiated'" />
    </xsl:call-template>
  </td>
</tr>
<tr>
  <td class="headercell15" colspan="100%">
<b>Other Information</b>
</td>
</tr>
<tr>
  <td class="thnormal">
    <xsl:call-template name="widget_render">
      <xsl:with-param name="fieldName" select="'rqstDate'" />
      <xsl:with-param name="renderCmd" select="'title'" />
    </xsl:call-template>
    <font color="#ff0000">*</font>
  </td>
  <td class="datacell">
    <xsl:call-template name="widget_render">
      <xsl:with-param name="fieldName" select="'rqstDate'" />
      <xsl:with-param name="renderCmd" select="'input'" />
      <xsl:with-param name="readOnly" select="'$isPastInitiated'" />
    </xsl:call-template>
  </td>
</tr>
<tr>
  <td class="thnormal">
    <xsl:call-template name="widget_render">
      <xsl:with-param name="fieldName" select="'campus'" />
      <xsl:with-param name="renderCmd" select="'title'" />
    </xsl:call-template>
    <font color="#ff0000">*</font>
  </td>
  <td class="datacell">
    <xsl:call-template name="widget_render">
      <xsl:with-param name="fieldName" select="'campus'" />
      <xsl:with-param name="renderCmd" select="'input'" />
      <xsl:with-param name="readOnly" select="'$isPastInitiated'" />
    </xsl:call-template>
  </td>
</tr>
<tr>
  <td class="thnormal">
    <xsl:call-template name="widget_render">
      <xsl:with-param name="fieldName" select="'description'" />
      <xsl:with-param name="renderCmd" select="'title'" />
    </xsl:call-template>
  </td>
  <td class="datacell">
    <xsl:call-template name="widget_render">
      <xsl:with-param name="fieldName" select="'description'" />
      <xsl:with-param name="renderCmd" select="'input'" />
      <xsl:with-param name="readOnly" select="'$isPastInitiated'" />
    </xsl:call-template>
  </td>
</tr>
<tr>
  <td class="thnormal" colspan="2">

```

```

<b>(Check all that apply)</b>
</td>
  </tr>
  <tr>
    <td class="datacell" colspan="2">
      <xsl:call-template name="widget_render">
        <xsl:with-param name="fieldName" select="'fundedBy'" />
        <xsl:with-param name="renderCmd" select="'input'" />
        <xsl:with-param name="readOnly" select="'$isPastInitiated'" />
      </xsl:call-template>
      <br />
      <xsl:call-template name="widget_render">
        <xsl:with-param name="fieldName" select="'researchHumans'" />
        <xsl:with-param name="renderCmd" select="'input'" />
        <xsl:with-param name="readOnly" select="'$isPastInitiated'" />
      </xsl:call-template>
      <br />
    </td>
  </tr>
  <tr>
    <td class="headercell1" colspan="100%">
<b>Supporting Materials</b></td>
  </tr>
  <tr>
    <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>
  </tr>
</table>
<br xmlns="" />
</xsl:template>
<xsl:template name="nbsp">
  <xsl:text disable-output-escaping="yes">&nbsp;&nbsp;&nbsp;</xsl:text>
</xsl:template>
</xsl:stylesheet>
</style>

```

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 (optional) 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 past the initiator's node.

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 be referenced and 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:

```
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <edoclite xmlns="ns:workflow/EDocLite" xsi:schemaLocation="ns:workflow/EDocLite resource:EDocLite">
    <style name="xyzAppStyle">
      <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">
        <!-- your custom stylesheet -->
      </xsl:stylesheet>
    </style>
  </edoclite>
</data>
```

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:

```
<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.

```
<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <documentTypes xmlns="ns:workflow/DocumentType" xsi:schemaLocation="ns:workflow/DocumentType
resource:DocumentType">
    <documentType>
      <name>eDoc.Example1Doctype</name>
      <parent>eDoc.Example1.ParentDoctype</parent>
      <description>eDoc.Example1 Request DocumentType</description>
      <label>eDoc.Example1 Request DocumentType</label>
      <postProcessorName>org.kuali.rice.kew.edl.EDocLitePostProcessor</postProcessorName>
      <superUserGroupName namespace="KUALI">eDoc.Example1.SuperUsers</superUserGroupName>
      <blanketApprovePolicy>NONE</blanketApprovePolicy>
      <defaultExceptionGroupName namespace="KUALI">eDoc.Example1.defaultExceptions</
defaultExceptionGroupName>
      <docHandler>${workflow.url}/EDocLite</docHandler>
      <active>true</active>
      <routingVersion>2</routingVersion>
      <routePaths>
        <routePath>
          <start name="Initiated" nextNode="eDoc.Example1.Node1" />
          <requests name="eDoc.Example1.Node1" />
        </routePath>
      </routePaths>
    </documentType>
  </documentTypes>
</data>
```

```

    <start name="Initiated">
      <activationType>P</activationType>
      <mandatoryRoute>>false</mandatoryRoute>
      <finalApproval>>false</finalApproval>
    </start>
    <requests name="eDoc.Example1.Node1">
      <activationType>P</activationType>
      <ruleTemplate>eDoc.Example1.Node1</ruleTemplate>
      <mandatoryRoute>>false</mandatoryRoute>
      <finalApproval>>false</finalApproval>
    </requests>
  </routeNodes>
</documentType>
</documentTypes>
</data>

```

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:

```

<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <documentTypes xmlns="ns:workflow/DocumentType" xsi:schemaLocation="ns:workflow/DocumentType
resource:DocumentType">
    <documentType>
      <name>eDoc.Example1.ParentDoctype</name>
      <description>eDoc.Example1 Parent Doctype</description>
      <label>eDoc.Example1 Parent Document</label>
      <postProcessorName>org.kuali.rice.kew.edl.EDocLitePostProcessor</postProcessorName>
      <superUserGroupName namespace="KUALI">eDoc.Example1.SuperUsers</superUserGroupName>
      <blanketApprovePolicy>NONE</blanketApprovePolicy>
      <docHandler>${workflow.url}/EDocLite</docHandler>
      <active>true</active>
      <routingVersion>2</routingVersion>
      <routePaths />
    </documentType>
  </documentTypes>
</data>

```

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 edl definition for campus looks like this:

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

Rule Routing

```
<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ns:workflow
resource:WorkflowData">
  <ruleAttributes xmlns="ns:workflow/RuleAttribute" xsi:schemaLocation="ns:workflow/RuleAttribute
resource:RuleAttribute">
    <ruleAttribute>
      <name>EDL.Campus.Example</name>
      <className>org.kuali.rice.kew.rule.xmlrouting.StandardGenericXMLRuleAttribute</className>
      <label>EDL Campus Routing</label>
      <description>EDL School Routing</description>
      <type>RuleXmlAttribute</type>
      <routingConfig>
        <fieldDef name="campus" title="Campus" workflowType="ALL">
          <display>
            <type>select</type>
            <values title="IUB">IUB</values>
            <values title="IUPUI">IUPUI</values>
          </display>
          <validation required="false" />
          <fieldEvaluation>
            <xpathexpression>//campus = wf:ruledata('campus')</xpathexpression>
          </fieldEvaluation>
        </fieldDef>
        <xmlDocumentContent>
          <campus>%campus%</campus>
        </xmlDocumentContent>
      </routingConfig>
    </ruleAttribute>
  </ruleAttributes>
</data>
```

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 15. Customizing Document Search

Each document carries an XML payload that describes metadata. You can specify pieces of that metadata to be indexed and searched on. This area focuses on the interface for searching through that data. For each **Document Search** page, you must setup the XML configuration files to define the search criteria and result fields.

Custom Search Screen

As an example of customizing a Document Search screen, we'll use a customized **Offer Request** screen:

Figure 15.1. Custom Search Screen: Offer Request Example

The screenshot shows a web interface for a custom document search. At the top, there is a navigation bar with the text "Document Search" and a help icon. To the right of the navigation bar are links for "detailed search", "superuser search", and "clear saved searches", followed by a dropdown menu labeled "Searches". Below the navigation bar is a table with search criteria. The first row is "Document Type:" with a dropdown menu set to "OfferRequest". The second row is "Initiator:" with an empty text input field. The third row is "Document Id:" with an empty text input field. The fourth row is "Group Viewer:" with a search icon. The fifth row is "Date Created From:" with an empty date input field. The sixth row is "Date Created To:" with an empty date input field. The seventh row is "OAA#:" with an empty text input field. The eighth row is "Department:" with an empty text input field. The ninth row is "Campus:" with a dropdown menu. The tenth row is "School:" with a dropdown menu. The eleventh row is "Name this search (optional):" with an empty text input field. At the bottom of the table are three buttons: "search", "clear", and "cancel". A small asterisk and the text "* required field" are located at the bottom right of the page.

What are custom document search attributes?

Custom document search attributes are associated with a document type. They specify which pieces of document data will be made searchable for documents of that type. When you take action on a document in the workflow engine, a background process extracts the custom search attributes from the document and adds them to a database table where they can be queried as part of a custom document search. These custom search attributes are defined and associated along with document types in WorkflowData XML files, and are added to Rice via the XML ingestor. They are defined within (using XPath notation) `/data/ruleAttributes/ruleAttribute` tags, and are associated with specific document types within `/data/documentTypes/documentType/attributes/attribute` tags.

A custom search attribute's logic is defined in a Java class that implements the `SearchAttribute` interface. A `SearchableAttribute` implementation defines:

- What parts of the document content will be made searchable
- Which fields will be present in the document search interface
- Which columns will be shown in the search results

- What is considered valid user input for the custom search fields

There is a built in SearchAttribute implementation, SearchableXMLAttribute, that is highly configurable via XML and will meet most requirements. If there is need for more complex or specific behavior, a custom SearchAttribute implementation can be written and utilized as well.

DocumentSearchAttributes is much like **XMLRuleAttributes**, except that DocumentSearchAttributes is responsible for drawing input fields on the Document Search form and collecting data for the query, as opposed to analyzing data for routing evaluation (done by XMLRuleAttributes).

Hide Search Fields and Result Columns

In a search configuration, the **<visibility>** tag lets you configure search criteria to be included or excluded from the entry of search criteria or from the search results. You can use the **<visibility>** tag on all field(s) and column(s) in the Document Search results except for **Document Id** and **Route Log**, which must always be visible.

Hide a result column

```
<visibility>
  <column visible="false"/>
</visibility>
```

Hide a search field

```
<visibility>
  <field visible="false"/>
</visibility>
```

Field and column visibility based on workgroup membership

Use code like this in the XML file to display column(s) and field(s) based on the user's workgroup:

```
<visibility>
  <field>
    <isMemberOfWorkgroup>WorkflowAdmin</isMemberOfWorkgroup>
  </field>
  <column>
    <isMemberOfWorkgroup>WorkflowAdmin</isMemberOfWorkgroup>
  </column>
</visibility>
```

The example above indicates that the field and column only display for users who are a member of the workgroup, *WorkflowAdmin*.

Configure visibility for both field and column

A shortcut to configure the visibility for both fields and columns is the **<fieldAndColumn>** tag. A **<fieldAndColumn>** example:

```
<visibility>
  <fieldAndColumn>
    <isMemberOfWorkgroup>WorkflowAdmin</isMemberOfWorkgroup>
  </fieldAndColumn>
</visibility>
```

No field visibility

Declaring `<type>` as hidden is equivalent to setting visibility to false. An example of `<type>` and `<visibility>`, equivalent to a hidden field:

```
<searchingConfig>
  <fieldDef name="department" title="Department">
    <display>
      <type>text</type>
    </display>
    <visibility>
      <field visible="false"/>
    </visibility>
    <fieldEvaluation>
      <xpathexpression>normalize-space(substring-before(//department, ' '))</xpathexpression>
    </fieldEvaluation>
  </fieldDef>
</searchingConfig>

<!-- The above is equivalent to the following searching configuration -->

<searchingConfig>
  <fieldDef name="department" title="Department">
    <display>
      <type>hidden</type>
    </display>
    <fieldEvaluation>
      <xpathexpression>normalize-space(substring-before(//department, ' '))</xpathexpression>
    >
  </fieldEvaluation>
  </fieldDef>
</searchingConfig>
```

Configure Lookup Function

To make a lookupable available on the Document Search screen, you can use the `<quickfinder>` tag in the attribute definition. You can use the terms *quickfinder*, *lookup*, and *lookupable* interchangeably.

For example, you could set up an organizational hierarchic concept such as **Charts and Orgs** to implement a search. You could set up the code to perform this search using the **ChartOrgLookupableImpl** institutional plugin. This is an example of a standard lookupable component.

In the institutional plug-in, **ChartOrgLookupableImpl** is identified in the `LookupableServiceExtension` by the name of **ChartOrgLookupableImplservice**. **ChartOrgLookupableImpl** exposes two return parameters, which are:

- **Fin_coa_cd**: Represents the chart code
- **Org_cd**: Represents the organization code

An XML example of setting up a lookupable on the Document Search screen: **ChartOrgSearchAttribute.xml**


```

<ruleAttribute>
  <name>ChartOrgSearchAttribute</name>
  <className>org.kuali.rice.kew.docsearch.xml.StandardGenericXMLSearchableAttribute</className>
  <label>TestQuickfinderSearchAttribute</label>
  <description>TestQuickfinderSearchAttribute</description>
  <type>SearchableXmlAttribute</type>
  <searchingConfig>
    <fieldDef name="chart" title="Chart">
      <display>
        <type>text</type>
      </display>
      <quickfinder service="ChartOrgLookupableImplService" appliesTo="fin_coa_cd" draw="false"/>
      <fieldEvaluation>
        <xpathexpression>//chart</xpathexpression>
      </fieldEvaluation>
    </fieldDef>
    <fieldDef name="org" title="Organization">
      <display>
        <type>text</type>
      </display>
      <quickfinder service="ChartOrgLookupableImplService" appliesTo="org_cd" draw="true"/>
      <fieldEvaluation>
        <xpathexpression>//org</xpathexpression>
      </fieldEvaluation>
    </fieldDef>
    <xmlSearchContent>
      <chartOrg>
        <chart>%chart%</chart>
        <org>%org%</org>
      </chartOrg>
    </xmlSearchContent>
  </searchingConfig>
</ruleAttribute>

```

In the XML example above, there are two **<quickfinder>** tags representing the **Chart (fin_coa_cd)** and **Org (org_cd)** search. Notice the **draw** attribute for the **Org (org_cd)** search is set **true**. This means that a search icon will be displayed on the Document Search screen. Based on the XML code above, the final Document Search screen looks like this:

Figure 15.2. Custom Document Search: Department Example

Application Document Status

If the **<validApplicationStatuses>** configuration is specified in the document type definition, then setting the Document Type on the Document Search page will display a multi-select input titled "Application Document Status" that allows you to search by application statuses or status categories.

Figure 15.3. Document Search Screen: Application Document Status Example

The screenshot shows a web interface for document search. At the top, there is a header bar with the text "Document Search" and a help icon. To the right of the header are buttons for "detailed search", "superuser search", and "clear saved searches", along with a "Searches" dropdown menu. Below the header, there is a table of search filters. The "Application Document Status" filter is expanded, showing a list of status options: "Pre-Submit", "- Initiated", "- Validated", "In Process", and "- Awaiting Content Approval". Other filters include "Document Type" (set to "ApplicationStatusExampleDo"), "Initiator", "Document Id", "Date Created From", "Date Created To", "Travel Account Number", and "Name this search (optional)". At the bottom of the form are "search", "clear", and "cancel" buttons.

In the figure above, **Pre-Submit** is a category of statuses containing **Initiated** and **Validated** which are individual statuses. Selecting **Pre-Submit** and searching will return identical results to selecting both **Initiated** and **Validated** and then searching.

Please see [Document Type Policies: DOCUMENT_STATUS_POLICY](#) for configuration details.

Define Keyword Search

XMLSearchableAttributeStdFloatRang is an XML searchable attribute that enhances the keyword search function. It provides multiple searchable elements for a user to select under the <searchingConfig> section. This example is the XMLSearchableAttributeStdFloatRang attribute in the default setting:

```
<ruleAttribute>
  <name>XMLSearchableAttributeStdFloatRange</name>
  <className>org.kuali.rice.kew.docsearch.xml.StandardGenericXMLSearchableAttribute</className>
  <label>XML Searchable attribute</label>
  <description>XML Searchable attribute</description>
  <type>SearchableXmlAttribute</type>
  <searchingConfig>
    <fieldDef name="testFloatKey" title="Float in the Water">
      <display>
        <type>text</type>
      </display>
      <searchDefinition dataType="float">
        <rangeDefinition inclusive="false">
          <lower label="starting"/>
          <upper label="ending"/>
        </rangeDefinition>
      </searchDefinition>
      <fieldEvaluation>
        <xpathexpression>//putWhateverWordsIwantInsideThisTag/testFloatKey/value</xpathexpression>
      </fieldEvaluation>
    </fieldDef>
    <xmlSearchContent>
      <putWhateverWordsIwantInsideThisTag>
        <testFloatKey>
          <value>%testFloatKey%</value>
        </testFloatKey>
      </putWhateverWordsIwantInsideThisTag>
    </xmlSearchContent>
  </searchingConfig>
</ruleAttribute>
```

```
</ruleAttribute>
```

Caution

Cautions about the <searchingConfig> section:

1. <searchDefinition> identifies the search data type and search ranges.
2. <rangeDefinition> contains both the <lower> and <upper> elements that set up the parameters for the range search.
3. If you set the <display><type> tag to be date, then KEW automatically sets: <searchDefinition dataType="datetime">.
4. If the data type that you enter is not a *datetime*, then KEW sets all *datePicker* attributes to *false*.
5. Based on the **dataType** you enter, *datePicker* changes the default setting to either *true* or *false*.
6. To use a range search, you can either set <searchDefinition rangeSearch="true"> or put the tag <rangeDefinition> under the <searchDefinition> tag. Either way, KEW will force a range search.

Custom Search Criteria Processing

URL Parameter Options

You can modify the search criteria and the display of the search screen by passing in URL parameters. Only use this method when the configuration desired is *preferable* and not *required*. If a particular piece of the search criteria is *required*, please see the section below titled, Using a Custom Search Criteria Processor.

Force the link to display the Detailed Search screen

Use the parameter **isAdvancedSearch** and set the value to **YES**.

Show or Hide All Criteria and/or the Workflow Header Bar

The default value of each of these parameters must be set to true to show both the criteria and the header bar.

- To hide the header bar, use the URL parameter **headerBarEnabled** and set the value to *false*.
- To hide the search criteria (including the buttons), use the URL parameter **searchCriteriaEnabled** and set the value to *false*.

Passing in Common Search Criteria Values

Common search criteria fields can be populated by supplying their values in the URL query parameters. For example, the following URL specifies a search on **KualiNotification** documents with initiator **user1**:

```
http://yourlocalip:8080/DocumentSearch.do?documentTypeName=KualiNotification&initiatorPrincipalName=user1
```

Common search criteria fields include:

- **documentTypeName** - the document type name
- **documentId** - the document id
- **initiatorPrincipalName** - the initiator principal name
- **dateCreated** - the document creation date
- **approverPrincipalName** - the approver principal name (use with advanced search)
- **viewerPrincipalName** - the viewer principal name (use with advanced search)
- **applicationDocumentId** - the application-supplied document id (use with advanced search)
- **dateApproved** - the approval date (use with advanced search)
- **dateLastModified** - the last modified date (use with advanced search)
- **dateFinalized** - the finalization date (use with advanced search)
- **title** - the document title(use with advanced search)

For a comprehensive list of search criteria fields, consult the

```
org.kuali.rice.kew.impl.document.search.DocumentSearchCriteriaBo
```

class.

The CURRENT_USER variable

In addition to literal field values, the 'CURRENT_USER' special token is dynamically replaced with an identifier for the currently authenticated user when the search is executed. This value can be supplied in any field (typically a field that takes a principal name or id). Several variants allow embedding different types of user ids:

- **CURRENT_USER**, **CURRENT_USER.principalName**, **CURRENT_USER.authenticationId**, **CURRENT_USER.a** - the current user principal name
- **CURRENT_USER.principalId**, **CURRENT_USER.workflowId**, **CURRENT_USER.w** - the current user principal id
- **CURRENT_USER.empId**, **CURRENT_USER.e** - the current user employee id

Example:

```
http://yourlocalip:8080/DocumentSearch.do?  
documentTypeName=KualiNotification&initiatorPrincipalName=CURRENT_USER
```

Passing in Searchable Attribute Values

Searchable attributes can be specified via URL parameters by prefixing the searchable attribute field name with **documentAttribute..**

Here is an example using two `<fieldDef>` objects with names *firstname* and *lastname*:

```
http://yourlocalip:8080/DocumentSearch.do?documentAttribute.firstname=John&documentAttribute.lastname=Smith
```

Using a Custom Search Criteria Processor

The best way to do custom criteria processing is to implement a custom class that extends the class `org.kuali.rice.kew.docsearch.DocumentSearchCriteriaProcessor`. This file is ingested as a Workflow Attribute in KEW, using the `<type>` of `DocumentSearchCriteriaProcessorAttribute`. Once the Workflow Attribute is ingested, you can set the name value of the ingested attribute on one or more document type xml definitions in the Attributes section. A document type can only have one Criteria Processor Attribute.

Creating a child class of the `DocumentSearchCriteriaProcessor` class, a client can override various methods to modify the behavior of the search. The `DocumentSearchCriteriaProcessor` class can access the `WorkflowUser` object of the user performing the search. By having access to these objects, a custom processor class could implement dynamic hiding and showing of specific criteria fields based on ordinary user's data or search field data.

Show or Hide All Criteria and/or the Workflow Header Bar

Here are some helpful methods that you may override from the `DocumentSearchCriteriaProcessor` class file to hide or display full criteria (including buttons) and/or the header bar:

- `isHeaderBarDisplayed()` – If this function returns *false*, KEW hides the header bar on both the advanced and basic search screens (default return value is *true*).
- `isBasicSearchCriteriaDisplayed()` – If this function returns *false*, KEW hides criteria on the basic search screen (default return value is *true*).
- `isAdvancedSearchCriteriaDisplayed()` – If this function returns *false*, KEW hides the criteria on the advanced search screen (default return value is *true*).

Hiding Specific Fields or Criteria Using Field Key Values

The `DocumentSearchCriteriaProcessor` class has methods that allow classes to extend from it for basic field display. This is based on static string key values and makes it easier for clients to allow basic field display or to hide particular fields, whether they are searchable attributes or standard Document Search fields.

You may override these methods from the `DocumentSearchCriteriaProcessor` class to do specific field hiding by returning a list of string keys:

- `getGlobalHiddenFieldKeys()` – This function returns a list of keys (strings) for fields to be hidden on both the basic and advanced search screen.
- `getBasicSearchHiddenFieldKeys()` – This function returns a list of keys (strings) for fields to be hidden on the basic search screen.
- `getAdvancedSearchHiddenFieldKeys()` – This function returns a list of keys (strings) for fields to be hidden on the advanced search screen.

You can find the standard Document Search field key names in the class file `org.kuali.rice.kew.docsearch.DocumentSearchCriteriaProcessor`. They are constants prefixed by

the text **CRITERIA_KEY_**. For example, the static criteria key for the **Document Id** field is **DocumentSearchCriteriaProcessor.CRITERIA_KEY_DOCUMENT_ID**.

A client can also use searchable attribute **<fieldDef>** name values to hide fields in the same way that you use constants. If a particular searchable attribute **<fieldDef>** name exists in a list returned by one of the above **hidden field key** methods, the criteria processor class overrides the default behavior of that **<fieldDef>** searchable attribute for visibility.

Here is a general example of a custom criteria processor class that extends **StandardDocumentSearchCriteriaProcessor**:

```
public class CustomDocumentSearchCriteriaProcessor extends DocumentSearchCriteriaProcessor {

/**
 * Always hide the header bar on all search screens
 */
@Override
public boolean isHeaderBarDisplayed() {
    return Boolean.FALSE;
}

/**
 * Always hide all criteria and buttons on the advanced search screen
 */
@Override
public Boolean isAdvancedSearchCriteriaDisplayed() {
    return Boolean.FALSE;
}

/**
 * Hide the Initiator Criteria field on both Basic and Advanced Search screens
 */
@Override
public List<String> getGlobalHiddenFieldKeys() {
    List<String> hiddenKeys = super.getGlobalHiddenFieldKeys();
    hiddenKeys.add(DocumentSearchCriteriaProcessor.CRITERIA_KEY_INITIATOR);
    return hiddenKeys;
}

/**
 * Hide the Document Title criteria field on the basic search screen
 * Hide the searchable attribute field with name 'givenname' on the basic search screen
 */
@Override
public List<String> getBasicSearchHiddenFieldKeys() {
    List<String> hiddenKeys = super.getAdvancedSearchHiddenFieldKeys();
    hiddenKeys.add(DocumentSearchCriteriaProcessor.CRITERIA_KEY_DOCUMENT_TITLE);
    hiddenKeys.add("givenname");
    return hiddenKeys;
}

/**
 * Hide the Document Title criteria field on the advanced search screen
 * Hide the searchable attribute field with name 'givenname' on the basic search screen
 */
@Override
public List<String> getAdvancedSearchHiddenFieldKeys() {
    List<String> hiddenKeys = super.getAdvancedSearchHiddenFieldKeys();
    hiddenKeys.add(DocumentSearchCriteriaProcessor.CRITERIA_KEY_DOCUMENT_TITLE);
    hiddenKeys.add("givenname");
}
}
```

```
return hiddenKeys;
}
}
```

Custom Search Generation

The best way to do custom search generation or processing is to implement a custom class that extends the class **org.kuali.rice.kew.impl.document.lookup.DocumentSearchGenerator**. This file is ingested as a Workflow Attribute in KEW using the **<type>** value of DocumentSearchGeneratorAttribute. Once the Workflow Attribute is ingested, the name value of the ingested attribute can be set on one or more document type xml definitions in the Attributes section. A Document Type can only have one Search Generator Attribute.

Using an extension of the **DocumentSearchGenerator** class, a client has access to override various methods to modify the behavior of the search. Also, the **DocumentSearchGenerator** class has helper methods that may be used to get the WorkflowUser object of the user performing the search.

Implementing a Custom Result Set Limit

To implement a custom result set limit, simply override the method **getDocumentSearchResultSetLimit()** from the **StandardDocumentSearchGenerator** class.





Custom Search Results

You can create a Custom Search Result table using an XML rule attribute of the type **DocumentSearchResultXMLResultProcessorAttribute**.

The *standard* Search Result table:

Figure 15.4. Standard Doc Search Results Set

20 items retrieved, displaying all items.

Document Id	Document Type	Title	Status	Initiator	Date Created	Route Log
3091	Waiver Request		ENROUTE	admin, admin	12/05/2011 03:23 PM	
3085	Permission	New GenericPermissionBo - KRMS Testing Perm MS	FINAL	admin, admin	12/05/2011 02:49 PM	
3084	KRMS Term Maintenance Document	New TermBo - New Term Document	FINAL	admin, admin	12/05/2011 02:48 PM	
3083	KRMS Term Specification Maintenance Document	New TermSpecificationBo - New Term Specification Document	FINAL	admin, admin	12/05/2011 02:45 PM	

The Standard Search Result fields:

- Document Id
- Document Type
- Title
- Status
- Initiator
- Date Created

- Route log

The fields of **Document Id** and **Route Log** are always shown in the farthest left and right columns of the Search Result table. These fields cannot be hidden. You can add both columns a second time in the XML search result attributes if needed.

Custom XML Document Search Result Processor Attribute

An example of a custom XML result processor:

```
<ruleAttribute>
  <name>KualiContractsAndGrantsDocSearchResultProcessor</name>

  <className>org.kuali.rice.kew.docsearch.xml.DocumentSearchXMLResultProcessorImpl</className>
  <label>Contracts & Grants Document Search Result Processor</label>
  <description>Attribute to allow for custom search results for Contracts & Grants documents</description>
  <type>DocumentSearchXMLResultProcessorAttribute</type>
  <searchResultConfig overrideSearchableAttributes="false" showStandardSearchFields="false">
    <column name="docTypeLabel" />
    <column name="docRouteStatusCodeDesc" />
    <column name="initiator" />
    <column name="dateCreated" />
  </searchResultConfig>
</ruleAttribute>
```

The result of the code displayed above is a Search Result table with these columns:

- Document Id
- Document Type
- Status
- Initiator
- Date Created
- Route Log

The key for the search result customization is focused on the elements and column tag(s) under the `<searchResultConfig>`.

Attributes that are included in the `<searchResultConfig>` tag:

- **overrideSearchableAttributes:** The indicator of whether to display the column *name* attributes defined by the searchAttribute fieldDef 'name's configured by setting the *true* or *false*
 - *true*: Display the `<column>` *name* attributes based on searchAttribute fieldDef names.
 - *false*: Display the *name* based on the `<column>` attribute.
- **showStandardSearchFields:** The indicator of whether to display the standard search fields by setting the value *true* or *false*.
 - *true*: Display the search result with the standard result fields; the *name* attribute of the `<column>` tag should match the values in the java file *DocumentSearchResult.java*.

- *false*: Display the search result based on the custom result fields.

Attributes that can be added in a <column> tag:

- **Name**: The key for connecting the value of a particular attribute. For example, *routeHeaderId* equals *Document Id*. For more information about the attribute key, please refer to the Key reference table below.
- **Title**: The title of the field
- **Sortable**: The indicator of whether to sort the search result by setting the value *true* or *false*
 - *true*: Sort option for this column is enabled to sort either alphabetically or numerically depending on attribute type.
 - *false*: Sort option for this column is disabled.

For <column> with *sortable = true*, the field title becomes a link and when a user clicks the link, KEW sorts the results by that column.

An example of a custom ruleAttribute:

```
<ruleAttribute>
  <name>KualiContractsAndGrantsDocSearchResultProcessor</name>

  <className>org.kuali.rice.kew.docsearch.xml.DocumentSearchXMLResultProcessorImpl</className>
  <label>Contracts & Grants Document Search Result Processor</label>
  <description>Attribute to allow for custom search results for Contracts & Grants documents</description>
  <type>DocumentSearchXMLResultProcessorAttribute</type>
  <searchResultConfig overrideSearchableAttributes="true" showStandardSearchFields="false">
    <column name="docTypeLabel" />
    <column name="docRouteStatusCodeDesc" />
    <column name="initiator" />
    <column name="dateCreated" />
    <column name="proposal_number" />
    <column name="chart" />
    <column name="organization" />
    <column name="proposal_award_status" />
    <column name="agency_report_name" />
  </searchResultConfig>
</ruleAttribute>
```

Table 15.1. Key Reference Table: Default field names and reference keys

Field	Key
Document Id	routeHeaderId
Document Type	docTypeLabel
Title	documentTitle
Status	docRouteStatusCodeDesc
Initiator	initiator
Date Created	dateCreated
Route Log	routeLog

Custom Document Search Result Processor Class File

You may also use a custom Document Search Result Processor by extending the class `org.kuali.rice.kew.docsearch.StandardDocumentSearchResultProcessor` and overriding individual methods.

Differences between SearchableAttribute and RuleAttribute

- SearchableAttribute does NOT have a **workflowType** attribute in the field tag.
- For SearchableAttribute, **xpathexpression** indicates the value's location in the document; it does not use **wf:ruledata("")**. For RuleAttribute, xpathexpression is a Boolean expression.
- SearchableAttribute uses xmlSearchContent instead of xmlDocumentContent; xmlDocumentContent is for RuleAttribute.

Document Security

Kuali Enterprise Workflow provides a declarative mechanism to facilitate Document-level security for these three screens:

- Document Search
- Route Log
- Doc Handler Redirection

Overview

1. You can create a security definition on a **Document Type**, which allows you to apply varying levels and types of security.
2. This definition is inheritable through the Document Type hierarchy.
3. If security is defined on a Document Type, rows for that Document Type that are returned from a search apply the security constraints and filter the row if the constraints fail.
4. Security constraints are evaluated against a document when its **Route Log** is accessed. If the security constraints fail, the user receives a *Not Authorized* message.
5. Security constraints are evaluated against a document when a **Doc Handler** link is clicked from either the **Action List** or **Document Search**. If the security constraints fail, the user receives a *Not Authorized* message.

Security Definition

You can define the security constraints in the Document Type XML. Here's a sample of the XML format:

```
<documentType>
  ....
  <security>
    <securityAttribute class="org.kuali.security.SecurityFilterAttribute" />
    <securityAttribute name="TestSecurityAttribute" />
    <initiator>true</initiator>
    <routeLogAuthenticated>true</routeLogAuthenticated>
    <searchableAttribute idType="emplid" name="emplid" />
    <group>MyWorkgroup</group>
    <role allowed="true">FACULTY</role>
    <role allowed="true">STAFF</role>
  </security>
</documentType>
```

```
</security>
....
</documentType>
```

There is an implicit **OR** in the evaluation of these constraints. Thus, the definition above states that the authenticated user has access to the document if:

- The attribute **org.kuali.security.SecurityFilterAttribute** defines the user as having access **OR**
- The attribute defined in the system by the name **TestSecurityAttribute** defines the user as having access **OR**
- The user is the initiator of the document **OR**
- The user is on the Route Log of the document **OR**
- The user's EMPL ID is equal to the searchable attribute on the document with the key of *emplid* **OR**
- The user is a member of the *MyWorkgroup* workgroup **OR**
- The user has the FACULTY role **OR**
- The user has the STAFF role

<initiator>

Validates that the authenticated user is or isn't the initiator of the document.

<routeLogAuthenticated>

Validates that the authenticated user is or isn't *Route Log Authenticated*.

Route Log Authenticated means that one of these is true:

1. The user is the initiator of the document.
2. The user has taken action on the document.
3. The user has received a request for the document (either directly or as the member of a workgroup).

Route Log Authenticated checks for security but **does not** simulate or check future requests.

<securityAttribute>

Validates based on a custom-defined class. Class must have implemented the *SecurityAttribute* interface class. There are two methods of defining a security attribute:

- KEW Attribute Name: Specify an already-defined attribute (via KEW XML ingestion) using the XML attribute *name* .

(Use of *applicationId* in a *ruleAttribute* specification sets the id of the application which contains the implementation of the security attribute.)

```
<documentType>
....
```

```
<security>
  <securityAttribute name="TestSecurityAttribute"/>
</security>
...
.</documentType>
```

- **Class Name:** Define the fully qualified class name using the XML attribute class.

(Use of Class Name is limited to classes which are locally defined.)

```
<documentType>
...
  <security>
    <securityAttribute class="org.kuali.security.SecurityFilterAttribute"/>
  </security>
...
</documentType>
```

<searchableAttribute>

Validate that the authenticated User ID of the given idType is equivalent to the searchable attribute field with the given name.

The following id types are valid:

- emplid
- authenticationid
- uuid
- workflowid

<group>

Validate that the authenticated user is a member of the workgroup with the given name.

<role>

Validate that the authenticated user has the given role. The existence and names of these roles are determined by your setup in KEW. (You can create these roles when you implement *WebAuthenticationService*.) Typically, the roles mirror your organization structure.

For example, you may choose to expose these roles:

- STAFF
- FACULTY
- ALUMNI
- STUDENT
- FORMER-STUDENT

- APPLICANT
- ENROLLED
- ADMITTED
- PROSPECT
- GRADUATE
- UNDERGRADUATE

If the role is marked as **allowed=true**, then anyone with that role passes the security constraint. If the role is marked as **allowed=false**, then if the individual has the given disallowed role but none of the allowed roles, he or she fails the security check.

Order of Evaluation

The security constraints are evaluated in the following order. If any single constraint passes, it bypasses evaluating the remaining constraints.

1. Security attribute
2. Initiator
3. Role
4. Workgroup
5. Searchable attribute
6. Route log authenticated

Security - Warning Messages

These security scenarios generate security warning messages:

Document Search

- If no rows are filtered because of security, the user sees the search result without any warning message on the **Document Search** page.
- If rows are filtered because of security, a red warning message on top of the **Document Search** page shows how many rows were filtered. For example, "19 rows were filtered for security purposes."
- If the initial result set returns more than the search result threshold (500 rows), and rows in the set subsequently get filtered because of security, then a red warning message shows how many rows were returned and filtered. For example, "Too many results returned, displaying only the first 450. 50 rows were filtered for security purpose. Please refine your search."

Route Log and Doc Handler

- If the defined security constraints stop a user from viewing a document, a red warning message shows at the top of the page if they attempt to access the Route Log. For example, "You are not authorized to access this portion of the application."

Service Layer

In an out-of-the-box installation of KEW, Document Security is handled by `org.kuali.rice.kew.doctype.DocumentSecurityServiceImpl`, which implements the `org.kuali.rice.kew.doctype.DocumentSecurityService` service interface.

Chapter 16. Document Link

Document Link Features

KEW provides an option for linking documents and BOs that are functionally related. The link between related documents is created and removed in a double link double delete fashion, which means: when a link is added/deleted from 1 document to another document, a link in the reverse direction is also added/deleted, this feature will guarantee that searching for linked documents can be done from either side of the link. Using this option, client applications can link documents by using document link API.

Document Link API

Document link API is exposed to the client through WorkflowDocument interface, below is the summary of the api:

1. get all links to orgn doc

```
public List<DocumentLinkDTO> getLinkedDocumentsByDocId(Long id) throws WorkflowException
```

2. get the link from orgn doc to a specific doc

```
public DocumentLinkDTO getLinkedDocument(DocumentLinkDTO docLinkVO) throws WorkflowException
```

3. add a link by id

```
public void addLinkedDocument(DocumentLinkDTO docLinkVO) throws WorkflowException
```

4. remove all links to this doc as orgn doc

```
public void removeLinkedDocuments(Long docId) throws WorkflowException
```

5. remove the link to the specific doc

```
public void removeLinkedDocument(DocumentLinkDTO docLinkVO) throws WorkflowException
```

Document Link API Example

It is pretty straightforward to use this api, below are some examples:

1. To add a link

```
WorkflowDocument doc = new WorkflowDocument(...);

DocumentLinkDTO testDocLinkVO = new DocumentLinkDTO()
testDocLinkVO.setOrgnDocId(Long.valueOf(5000));

testDocLinkVO.setDestDocId(Long.valueOf(6000));
doc.addLinkedDocument(testDocLinkVO);
```

2. To retrieve all links to a document

```
List<DocumentLinkDTO> links2 = doc.getLinkedDocumentsByDocId(Long.valueOf(5000));
```

3. To remove a link

```
doc.removeLinkedDocument(testDocLinkVO);
```

Chapter 17. Reporting Guide

Reporting Features

KEW provides various options for reporting on and simulation of routing scenarios. There is a GUI for performing these reporting functions as well as an API that you can use to run routing reports against the system.

The Routing Report Screen

From the Rice main menu there is a link to the **Routing Report** screen. From this set of screens you can enter various criteria for running reports against the routing engine. The output of this reporting is a simulated view of the **Route Log**, displaying the result of the report.

The Report APIs

The KEW client API also provides facilities for running reports against the routing engine. At the core of KEW is a **Simulation Engine** that is responsible for running these types of reports. The method for executing these reports is on the Workflow Info object that is part of the client API. The method is defined:

```
public DocumentDetailVO routingReport(ReportCriteriaVO reportCriteria) throws WorkflowException;
```

This method takes the report criteria and returns the results of the routing report.

Report Criteria

The routing report operates under two basic modes:

1. Reports that run against existing Documents
2. Reports that simulate a Document from a Document Type

In each these cases there are certain properties that you need to set on the ReportCriteriaVO to obtain the desired results.

In the first case, the report runs against a document that has already been created in the system. This document already has a Document Id and may be en route. Using this style of reporting, you can run simulations to determine where the document will go in future route nodes. For example, to run a simulation against an existing document to determine to whom it will route in the future, execute this code:

Routing Report against a Document

```
WorkflowInfo info = new WorkflowInfo();
RoutingReportCriteriaVO criteria = new ReportCriteriaVO(new Long(1234));
DocumentDetailVO results = info.routingReport(criteria);
// examine results...
```

This runs a report against the document with ID 1234, starting at the active nodes of the document and continuing to the terminal nodes of the document. The DocumentDetailVO will contain the Action Requests generated during the report simulation.

You can also stop the report at a particular node or once Rice generates a request for a particular user. For example, to stop the report simulation at a node or when Rice generates a certain user's request, configure the report criteria like this:

Terminate Report at Node or User

```
WorkflowInfo info = new WorkflowInfo();

RoutingReportCriteriaVO criteria = new ReportCriteriaVO(new Long(1234), "MyNodeName");
criteria.setTargetUsers(new UserIdVO[] { new NetworkIdVO("ewestfal") });

DocumentDetailVO results = info.routingReport(criteria);
```

This executes the report until it reaches a node named **MyNodeName** or a request is generated for user **ewestfal**.

In the second style of reporting, the report is run against an arbitrary Document Type and the simulation engine creates a temporary document against which to run the report. When setting up the report criteria for these scenarios, you usually populate the XML content of the document on the criteria (provided that the routing of that document evaluates the XML). Also, the criteria need to be configured with the valid node names (or rule templates) against which the report should be run. For example, to run a Document Type report, you can invoke the routing report this way:

Report against a Document Type

```
WorkflowInfo info = new WorkflowInfo();
RouteReportCriteriaVO criteria = new ReportCriteriaVO("MyDocumentType");

criteria.setXmlContent("<accountNumber>1234</accountNumber>");

criteria.setNodeNames(new String[] { "MyNodeName" });
DocumentDetailVO results = info.routingReport(criteria);
```

The code above simulates the generation of requests for **MyDocumentType** at the **MyNodeName** node with the XML given. This sort of reporting is especially useful if you simply need to determine what rules in the rule system will fire and generate action requests under a particular scenario.

As an alternative to specifying the node names, you can also specify rule template names. This is simply another way to target a specific node in the document. It searches the Document Type definition for nodes with the specified rule templates and then runs the report against those nodes. Currently, the rule template must exist on a node in the Document Type definition or an error will be thrown. In the case of our previous example, you could simply change the line that sets the node names on the criteria to:

```
criteria.setRuleTemplateName(new String[] { "MyRuleTemplate" });
```

As above, this is primarily useful for determining who will have requests generated to them from the KEW rule system.

Interpreting Report Results

As we've seen, the object returned by the Routing Report is an instance of DocumentDetailVO. This object extends RouteHeaderVO and provides three more pieces of data along with it:

1. An array of ActionRequestVO objects representing the action requests on the document

2. An array of ActionTakenVO objects representing the actions that have been performed against the document
3. An array of RouteNodeInstanceVO objects that represent nodes in the document path

For reporting, the most important piece of data here is typically the ActionRequestVO objects. After running a report, this array contains the Action Requests that were generated as the result of the simulation. So, for example, in the example above where we run a document type report against the **MyRuleTemplate** rule template, this array contains all of the Action Requests that were generated to users or workgroups during the report simulation.

Chapter 18. Workflow Plugin Guide

Overview

Kuali Enterprise Workflow (KEW) has a plugin framework that allows you to load code into the core system without requiring changes to the KEW source code or configuration. This framework provides:

- A custom class loading space
- Hot deploy and reload capabilities
- Participation in Workflow's JTA transactions
- An application plugin for installation of routing components

Application Plugin

Use an application plugin to deploy an application area's routing components into Workflow. These routing components might include:

- Rule attributes
- Searchable attributes
- Post processors
- Route modules

If these components require access to a data source, then the application plugin also configures the data source and allows it to participate in Workflow's JTA transactions.

In addition to routing components, the application plugin can also configure a plugin listener and a Resource Loader. The Resource Loader is responsible for loading resources (both Java classes and service implementations) from the plugin and providing them to the core system.

Application plugins are hot-deployable, so a restart of the server is not required when they are added or modified. The core system searches for plugins in a directory configured in the application configuration (see KEW Module Configuration).

Plugin Layout

You build the plugin as a set of files and directories. You then zip this structure and place it in the appropriate Workflow plugin space. For application plugins, this directory location is defined in the core system configuration.

The name of the zip file (minus the .zip extension) is used as the name of the plugin. The Plugin Loader only looks for files that end in .zip when determining whether to load and hot-deploy a plugin.

In general, application plugins can be named as desired. However, there is one reserved plugin name:

shared - A special plugin that provides a shared classloading space to all plugins (see Plugin Shared Space).

The directory structure of a plugin is similar to that of a web application. It should have this structure:

```
classes/
lib/
META-INF/
  workflow.xml
```

- **classes** - All java .class files that are used by the plugin should reside in this directory
- **lib** - All .jar library files that are used by the plugin should reside in this directory
- **META-INF** - The **workflow.xml** configuration file must reside in this directory

Plugin Configuration

Application plugins usually provide a subset of the functionality that an institutional plugin provides, since the institutional plugin can provide core service overrides.

The plugin framework provides two configuration points:

1. Plugin XML Configuration (described below)
2. Transaction and DataSource Configuration

Plugin XML Configuration

The XML configuration is defined in a file called workflow.xml. The format of this file is relatively simple. An example workflow.xml file:

```
<plug-in>
  <param name="my.param.1">abc</param>
  <param name="my.param.2">123</param>
  <listener>
    <listener-class>org.kuali.rice.core.ApplicationInitializeListener</listener-class>
  </listener>
  <resourceLoader class="my.ResourceLoader" />
</plug-in>
```

We'll explain each of these elements in more detail below:

Plugin Parameters

The parameter configuration uses XML as the syntax. These parameters are placed into a configuration context for the plugin. The configuration inherits (and can override) values from the parent configurations. The configuration hierarchy is core -> institutional plugin -> application plugins.

A plugin can access its configuration using this code:

```
org.kuali.rice.Config config = org.kuali.rice.Core.getCurrentContextConfig();
```

Plugin Listeners

You can define one or more listeners that implement the interface **org.kuali.rice.kew.plugin.PluginListener**. These can be used to receive plugin lifecycle notifications from KEW.

The interface defines two methods to implement:

- Invoked when a plugin starts up

```
public void pluginInitialized(Plugin plugin);
```

- Invoked when a plugin shuts down

```
public void pluginDestroyed(Plugin plugin);
```

It is legal to define more than one plugin listener. Plugin listeners are started in the order in which they appear in the configuration file (and stopped in reverse order).

Resource Loader

A plugin can define an instance of **org.kuali.rice.resourceloader.ResourceLoader** to handle the loading of classes and services. When KEW attempts to load classes or locate services, it searches the institutional plugin, then the core, then any application plugins. It does this by invoking the **getObject(..)** and **getService(...)** methods on the plugin's ResourceLoader.

If no ResourceLoader is defined in the plugin configuration, then the default implementation **org.kuali.rice.resourceloader.BaseResourceLoader** is used. The BaseResourceLoader lets you examine the plugin's classloader for objects when requested (such as post processors, attributes, etc.). This is sufficient for most application plugins.

For more information on configuring service overrides in a plugin, see the [Overriding Services with a ResourceLoader](#) section below.

Configuring an Extra Classpath

Sometimes it is desirable to be able to point in a plugin to classes or library directories outside of the plugin space. This can be particularly useful in development environments, where the plugin uses many of the same classes as the main application that is integrating with Workflow. In these scenarios, configuring an extra Classpath may mean you don't need to jar or copy many common class files.

To do this, specify these properties in your plugin's **workflow.xml** file:

1. **extra.classes.dir** - Path to an additional directory of **.class** files or resources to include in the plugin's classloader
2. **extra.lib.dir** - Path to an additional directory of **.jar** files to include in the plugin's classloader

The classloader then includes these classes and/or lib directories into its classloading space, in the same manner that it includes the standard **classes** and **lib** directories. The classloader always looks in the default locations first, and then defers to the extra classpath if it cannot locate the class or resource.

Transaction and DataSource Configuration

The easiest method to configure Datasources and Transactions is through the Spring Framework. Here is a snippet of Spring XML that shows how to wire up a Spring Transaction Manager inside of a plugin:

```
<bean id="userTransaction" class="org.kuali.rice.jta.UserTransactionFactoryBean" />
```

```
<bean id="jtaTransactionManager" class="org.kuali.rice.jta.TransactionManagerFactoryBean" />
<bean id="transactionManager" class="org.springframework.transaction.jta.JtaTransactionManager">
  <property name="userTransaction" ref="userTransaction" />
  <property name="transactionManager" ref="jtaTransactionManager" />
  <property name="defaultTimeout" value="${transaction.timeout}"/>
</bean>
```

The factory beans in the above XML will locate the **javax.transaction.UserTransaction** and **java.transaction.TransactionManager**, which are configured in the core system. These can then be referenced and injected into other beans (such as the Spring JtaTransactionManager).

Once you configure the transaction manager, you also need to configure any DataSources you require. Here's an example of configuring a DataSource that participates in Atomikos JTA transactions (the default Transaction Manager distributed with Rice Standalone).

```
<bean id="myDataSource" class="com.atomikos.jdbc.nonxa.NonXADataSourceBean">
  <property name="uniqueResourceName" value="myDataSource"/>
  <property name="driverClassName" value="..."/>
  <property name="url" value="..."/>
  ...
</bean>
```

So, the application can access it's datasource by either injecting it into Spring services or by fetching it directly from the Spring context.

You can find more information on configuring Rice DataSources and TransactionManagers in Datasource and JTA Configuration.

OJB Configuration within a Plugin

If your plugin needs to use OJB, there are a few other configuration steps that you need to take. First, in your Spring file, add the following line to allow Spring to locate OJB and the JTA Transaction Manager:

```
<bean id="ojbConfigurer" class="org.kuali.rice.ojb.JtaOjbConfigurer">
  <property name="transactionManager" ref="jtaTransactionManager" />
</bean>
```

Next, for OJB to plug into Workflow's JTA transactions, you need to modify some settings in the plugin's **OJB.properties** file (or the equivalent):

```
PersistenceBrokerFactoryClass=org.apache.ojb.broker.core.PersistenceBrokerFactorySyncImpl
ImplementationClass=org.apache.ojb.odmg.ImplementationJTAImpl
OJBTxManagerClass=org.apache.ojb.odmg.JTATxManager
ConnectionFactoryClass=org.kuali.rice.ojb.RiceDataSourceConnectionFactory
JTATransactionManagerClass=org.kuali.rice.ojb.TransactionManagerFactory
```

The first three properties listed are part of the standard setup for using JTA with OJB. However, there are custom Rice implementations:

- **org.kuali.rice.ojb.RiceDataSourceConnectionFactory**
- **org.kuali.rice.ojb.TransactionManagerFactory**
- **org.kuali.rice.ojb.RiceDataSourceConnectionFactory**

This OJB ConnectionFactory searches your Spring Context for a bean with the same name as your **jcd-alias**. Here is what an OJB connection descriptor might look like inside of a Workflow plugin:

```
<jdbc-connection-descriptor
  jcd-alias="myDataSource"
  default-connection="true"
  platform="Oracle9i"
  jdbc-level="3.0"
  eager-release="false"
  batch-mode="false"
  useAutoCommit="0"
  ignoreAutoCommitExceptions="false">

  <sequence-manager className="org.apache.ojb.broker.util.sequence.SequenceManagerNextValImpl" />
  <object-cache class="org.apache.ojb.broker.cache.ObjectCachePerBrokerImpl" />
</jdbc-connection-descriptor>
```

Notice that the **jcd-alias** attribute matches the name of the DataSource Spring bean defined in the example above.

Another important thing to notice in this configuration is that **useAutoCommit** is set to 0. This tells OJB not to change the auto commit status of the connection because it is being managed by JTA.

Finally, when your plugin needs to use OJB, you need to use this:

```
org.kuali.rice.ojb.TransactionManagerFactory
```

This provides OJB with the **javax.transaction.TransactionManager** that was injected into your JtaOjbConfigurer, as in the example above.

Overriding Services with a ResourceLoader

For a service override, you need to define a custom ResourceLoader implementation and configure it in your workflow.xml plugin configuration file. The org.kuali.rice.resourceloader.ResourceLoader interface defines this relative method:

```
public Object getService(javax.xml.namespace.QName qname);
```

When KEW is searching for services, it invokes this method on its plugins' ResourceLoader implementations. The service name is a qualified name (as indicated by the use of **javax.xml.namespace.QName**), but for services being located from the core, service names typically contain only a local part and no namespace.

The easiest way to implement a custom ResourceLoader is to create a class that extends from **org.kuali.rice.resourceloader.BaseResourceLoader** and just override the **getService(QName)** method. The BaseResourceLoader provides standard functionality for loading objects from ClassLoaders, among other things.

For example, if you want to override the User Service, you might implement this ResourceLoader:

```
public class MyResourceLoader extends BaseResourceLoader {
  public MyResourceLoader() {
    super(new QName("MyResourceLoader"));
  }
}
```



```

@Override
public Object getService(QName serviceName) {
    if ("enUserOptionsService".equals(serviceName.getLocalPart())) {
        // return your custom implementation of org.kuali.rice.kew.useroptions.UserOptionsService
    } else if (...) {
        ...
    } else if (...) {
        ...
    }
    return super.getService(serviceName);
}
}

```

In the next section, we'll look at some of the services commonly overridden in an institutional plugin

Commonly Overridden Services

In theory, you can override any service defined in the **org/kuali/workflow/resources/KewSpringBeans.xml** file in the Institutional Plugin. What follows is a list of the most commonly overridden services:

Table 18.1. Commonly Overridden Services

Service Name	Interface	Description
enUserOptionsService	org.kuali.rice.kew.useroptions.UserOptionsService	Provides User lookup and searching services
IdentityHelperService	org.kuali.rice.kew.identity.service.IdentityHelperService	Interfaces with KIM identity management services
enEmailService	org.kuali.rice.kew.mail.service.impl.DefaultEmailService	Provides email sending capabilities
enNotificationService	org.kuali.rice.kew.service.NotificationService	Provides callbacks for notifications within the system

User Service

The Workflow core uses the `UserService` to resolve and search for users. The `UserService` could be as simple as a static set of users or as complex and integrated as a university-wide user system. Your institution may choose how to implement this, as long as you provide capabilities for the ID types that you intend to use. At the very least, implementations are required for the **WorkflowUserId** and **AuthenticationUserId** types (and their corresponding VO beans). All of the `UserId` types must be unique across the entire set of users.

The **WorkflowUserId** is typically associated with a unique numerical sequence value and the **AuthenticationUserId** is typically the username or network ID of the user.

The default `UserService` implementation provides a persistent user store that allows you to create and edit users through the GUI. It also caches users for improved performance and implements an XML import for mass user import. Institutions usually override the default user service with an implementation that integrates with their own user repository.

IdentityHelper Service

The `IdentityHelper` service helps to interact with the KIM identity management services in the system. `IdentityHelpers` are identified in one of two ways:

1. **PrincipalId** - A numerical identifier for a KIM principal
2. **Group** - An object associated with a group of principal users numerical identifier assigned to a Workgroup

Both of these object variables are implemented in KEW in the IdentityHelperServiceImpl file.

Email Service

The Email service is used to send emails from KEW. You can configure the default implementation when you configure KEW (see KEW Configuration). However, if more custom configuration is needed, then you can override the service in the plugin.

For example, you could override this service if you need to make a secure and authorized SSL connection with an SMTP server because of security policies.

Notification Service

The Notification service is responsible for notifying users when they receive Action Items in their Action List.

The default implementation simply sends an email (using the EmailService) to the user according to the individual user's preferences. A custom implementation might also notify other (external) systems in addition to sending the email.

Encryption Service

The Encryption service is responsible for encrypting document content.

The default implementation uses DES to encrypt the document content. If the **encryption.key** configuration parameter is set as a *Base64* encoded value, then the document content is encrypted using that key. If it is not set, then document content will not be encrypted and will be stored in the database in plain text.

Plugin Shared Space

All plugins also load certain classes from a shared space. The shared space contains certain classes that link with certain libraries that might exist in each application or institutional plugin's classloader (such as OJB and Spring). Current classes that Workflow publishes in the **shared** space are those in the shared module of the Rice project (**rice-shared-version.jar**). This is important because some of these classes link with libraries like Spring or OJB and since the plugin needs its own copy of these libraries, it needs to ensure that it doesn't retrieve these classes from any classloader but it's own.

Chapter 19. KEW Usage of the Kuali Service Bus

General Usage

The Kuali Enterprise Workflow engine makes use of both synchronous service endpoints and asynchronous messaging features from the Kuali Service Bus.

Most asynchronous processing that KEW does is implemented using asynchronous messaging on the service bus. This includes:

1. Workflow engine processing
2. Blanket approval orchestration
3. Action processing for actions taken directly from the Action List
4. Re-resolving actions requests resulting from a responsibility change
5. Sending email reminders
6. Distributed cache flush notifications

In each of these cases, there exists a service that processes asynchronous messages and performs the appropriate actions for each of these functions.

In terms of synchronous services, Kuali Enterprise Workflow publishes two different types of services. One is used for performing workflow document actions (such as creating, approving, disapproving, etc.). The other is used to perform various query or read-only operations against the workflow system.

Implications of using "Synchronous" KSB messaging with KEW

For general information on synchronous messaging and its implications in the KSB, please read "Implications of synchronous vs. asynchronous Message Deliver" in the KSB technical reference guide.

In terms of Kuali Enterprise Workflow, the usage of synchronous messaging means that operations like workflow engine processing will happen immediately and synchronously at the time it's invoked.

The main implication here besides what is listed in the KSB documentation is that, since message exception handling isn't implemented, exception routing does not work when using synchronous KSB messaging.

This means that if this messaging model is being used in a batch job, or similar type of program, routing exceptions will need to be manually caught. If it's desired to place a document into exception status from here, there are methods on the KEW APIs to do this manually.

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: <ul style="list-style-type: none">• 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 transition into the processed state if necessary.• 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: <ul style="list-style-type: none">• 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: <ul style="list-style-type: none">• 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 be 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.

	<ul style="list-style-type: none">• 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 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: <ul style="list-style-type: none">• 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) authentication implementation in Rice does not require a password as it is intended for testing purposes 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	<p>The standard fields that are defined and collected for every Routing Rule. These include:</p> <ul style="list-style-type: none"> • 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	<p>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.</p>
Blanket Approve Workgroup	<p>A workgroup that has the authority to Blanket Approve a document.</p>
Branch	<p>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.</p>
Business Rule	<ol style="list-style-type: none"> 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	<p>Identifies the different fiscal and physical operating entities of an institution.</p>
Campus Type	<p>Designates a campus as physical only, fiscal only or both.</p>
Cancel	<p>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.</p>

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. Quali 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 Quali Identity Management.
Client	A Java Application Program Interface (API) for interfacing with the Quali 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: <ul style="list-style-type: none">• 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	<ol style="list-style-type: none"> 1. An acronym for Educational Community License. 2. All Quali 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 - Kualii Coeus TODO

KCA - Kualii Commercial Affiliates A designation provided to commercial affiliates who become part of the Kualii Partners Program to provide for-fee guidance, support, implementation, and integration services related to the Kualii software. Affiliates hold no ownership of Kualii intellectual property, but are full KPP participants. Affiliates may provide packaged versions of Kualii that provide value for installation or integration beyond the basic Kualii software. Affiliates may also offer other types of training, documentation, or hosting services.

KCB – Kualii Communications Broker KCB is logically related to KEN. It handles dispatching messages based on user preferences (email, SMS, etc.).

KEN - Kualii 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 – Kualii Enterprise Workflow Kualii 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 Kualii Enterprise Workflow to automate and regulate the approval process for the transactions or documents they create.

KFS – Kualii 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 – Kualii Identity Management A Kualii Rice module, Kualii Identity Management provides a standard API for persons, groups, roles and permissions that can be implemented by an institution. It also provides an out of the box reference implementation that allows for a university to use Kualii 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: <ul style="list-style-type: none"> • 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	<ol style="list-style-type: none"> 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 Kualu Partners program.

Kualu Rice

Provides an enterprise-class middleware suite of integrated products that allow both Kualu and non-Kualu 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 "KUALU".

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	<p>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.</p> <p>In other words, an Entity is for identity while a Principal is for access management.</p> <p>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.</p>
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 Kualu 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	<p>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.</p> <ul style="list-style-type: none">• 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	<p>The status of a document in the course of its routing:</p> <ul style="list-style-type: none">• 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. <u>No changes are allowed to a document that is in Final status.</u>

- **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 its 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	<p>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.</p> <p>Technical considerations for a Searchable Attributes are:</p> <ul style="list-style-type: none">• 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	<p>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.</p> <p>Secondary Delegation is often configured for a range of dates and it must be registered in KEW or KIM to be in effect.</p>
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	<p>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.</p> <p>This field is equivalent to the "Subject" field in an email.</p>

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 Kualu Enterprise Workflow (KEW). The Kualu 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 Ingestor](#).

1. An acronym for Extensible Markup Language.
2. Used for data import/export.

XML Ingestor

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