# Kuali Rice 2.5.8-SNAPSHOT KRMS Guide

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1. Kuali Rule Management System: Overview

## What is a Rule Management System, in general?

Wikipedia defines a business rule management system, in general, as follows: "a software system used to define, deploy, execute, monitor and maintain the variety and complexity of decision logic that is used by operational systems within an organization or enterprise. This logic, also referred to as business rules, includes policies, requirements, and conditional statements that are used to determine the tactical actions that take place in applications and systems."

A key aspect of a rules management system is that it enables rules to be defined and maintained separately from application code. This modularity has the potential to reduce application maintenance costs, enable increased automation and application flexibility, and to enable business analysts and business process experts who are not developers and who reside outside of the IT organizations in the business departments themselves, to be more directly involved in creating and managing their rules.

A rules management system in general includes a repository of decision logic and a rules engine that can be executed by applications in a run-time environment. Again from wikipedia: "... provides the ability to: register, define, classify, and manage all the rules, verify consistency of rules definitions ("Gold-level customers are eligible for free shipping when order quantity > 10" and "maximum order quantity for Silver-level customers = 15" ), define the relationships between different rules, and relate some of these rules to IT applications that are affected or need to enforce one or more of the rules."

## What is KRMS?

Kuali's Rule Management System (KRMS) supports the creation, maintenance, storage and retrieval of business rules and agendas (ordered sets of business rules) within business contexts (e.g., for a particular department or for a particular university-wide process). It allows applications to externalize business logic that commonly needs customization, and empowers business analysts to modify them to reflect changes in policy.

KRMS enables you to define a set of rules within a particular business unit or for a particular set of applications. These business rules define logical conditions and the set of actions that result when those conditions are met. KRMS enables you to call and use this logic from any application, without having to re-write and manage all the rules' maintenance logic within the application.

Integration with organizational hierarchies and structures can be accomplished today using KEW for routing and approval, and KEW also has a legacy rule system of its own that can be used to make routing decisions. But before KRMS, managing general customizable business logic such as "if the transaction date is in the future OR the transaction date is less than the account activation date then flag the transaction for review" was the responsibility of the applications themselves. KRMS now offers a way to manage this type of logic externally in a repository that allows for business analysts to change it without having to modify application code.

Because KRMS is a general-purpose business rules management system, you can use it for many things, for example, you can define a rule to specify that when an account is closed, a continuation account must

be specified. You can also define rules to manage your organizational hierarchies and internal structures. You can define compound propositional logic, for example, "Must meet":

- P1 - 12 credits of FirstYearScience (CLU set)

  AND

- P2 - Completed CALC101 with grade >= B+

  AND

- p3 - Average of B+ on last 12 credits

# What problems or functions does KRMS solve?

KRMS gives business applications a powerful tool to externalize logic in places where customization will often be needed. This lowers the costs of adopting and administering the application by reducing the need for changes to the software itself, and allows the application to more fluidly reflect the institution's desired business processes.

There are a wide variety of actions that KRMS rules can be used to govern:

- Workflow Action rules - e.g. route an approval request
- Notification rules - e.g. send a notification to these people
- Validation rules - e.g. display this validation error message
- Questionnaire rules - e.g. administer this questionaire
- Custom-developed actions

For example, calling a KRMS set of rules (an agenda) from your application can result in routing a document to a PeopleFlow*, which is a new feature in KEW in Rice 2.0, or to any other action you define in KRMS.

* 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 role, group or individual who might need to approve or be notified.

# What problems does KRMS not address?

Some rule engines are built upon special algorithms that allow for forward or backward chaining (one example is Rete) that make them suitable for efficiently evaluating highly complex systems of what are known as production rules. The default engine implementation for KRMS is not designed upon such an algorithm, and it does not support either forward or backward chaining.

# With which types of applications can KRMS integrate?

Any Rice-based application can use KRMS.

# Can I use KRMS without building a Rice application?

The project has aspirations to increase Rice's modularity, and some strides have been made, but at the time of this writing the answer is no.

# When to use KRMS?

A big question is, when should you use KRMS rules verses logic directly coded in an application or document? This question can be answered by careful consideration of questions like these:

- How often the rule is likely to change
- How tightly coupled the rule is to the organization's unchanging policies
- The complexity of the rule
- The flexibility of the rule
- How the rule relates to other rules

# Chapter 2. KRMS Concepts

## Namespaces, Contexts, Agendas, Rules and Propositions

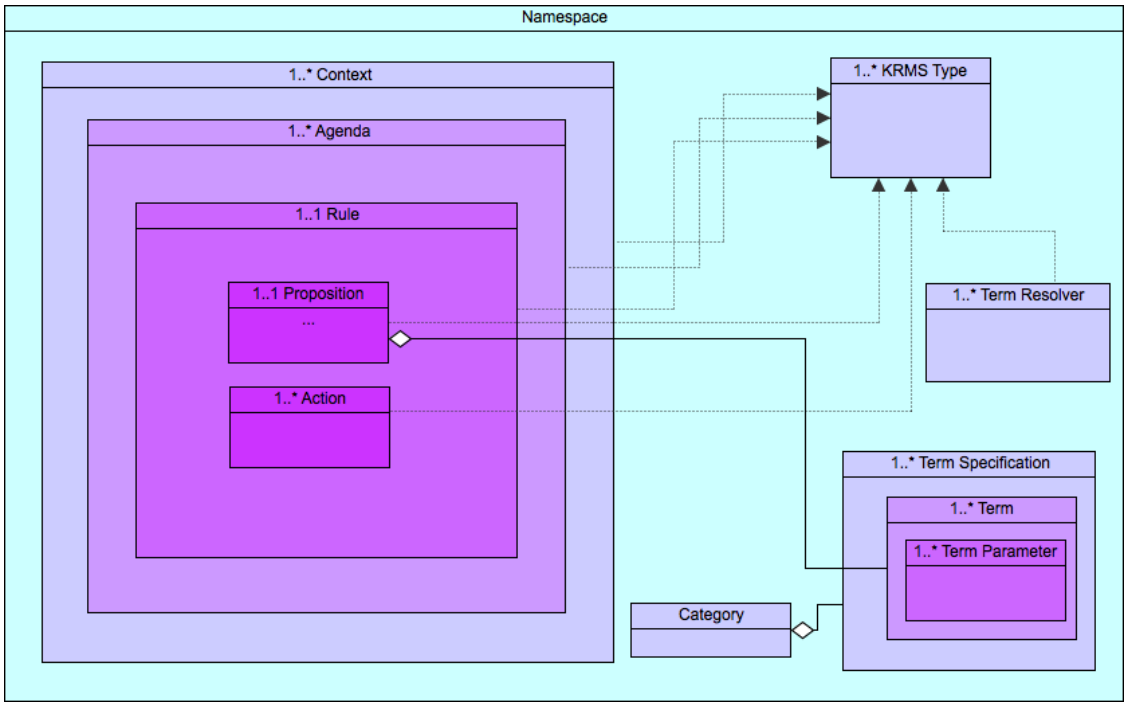
Namespaces are the top level container in KRMS. They contain Contexts, KRMS Types, and all things related to Terms. There isn't a namespace entity in the KRMS schema, they are specified via namespace code fields on the applicable child entities.

Rules in KRMS are placed into ordered sets called Agendas. The order of the Rules in an Agenda determines the sequencing: which rule gets evaluated first, second and so on. The Agenda also enables you to include conditional branching logic between Rules.

In turn, Agendas are are created in Contexts, which may represent any categories of rules that are relevant within your institution. For example, they will frequently correspond to document types, but they could be more finely grained to encompass only a certain kind of rule that you might run, e.g. you might have a context called "Proposal Validations". In some university environments, the following might be relevant contexts: Awards, Proposals, IRB reviews, Course co-requisites, Course pre-requisites, Student plan evaluations, and so on.

Each Context contains its own agendas, and each Agenda contains its own Rules. Rules aren't shared across Agendas (though you can copy/paste, they become unique Rule instances), and Agendas aren't shared across Contexts. There is no Context hierarchy, that is, Agendas and Rules can't be inherited across contexts within any sort of hierarchy.

The following diagram outlines the hierarchy of entities in KRMS (note that some entities are omitted)




You'll also note that many of the entities in the above diagram are KRMS Types. In most cases (the notable exception is Context) what that means is that you can develop and integrate custom implementations of the engine objects associated with those entities. These include:

- Agendas with custom selection and execution code
- Actions with custom execution code
- Rules with custom evaluation and Action triggering code
- Propositions with custom evaluation code
- Term Resolvers with custom value resolution code

# Propositions

Rules consist of propositions, and KRMS supports the following three main types of propositions:

1. Simple Propositions - a proposition of the form lhs op rhs where lhs=left-hand side, rhs=right-hand side, and op=operator
2. Compound Propositions - a proposition consisting of more than one simple proposition and a boolean algebra operator (AND, OR) between each of the simple propositions
3. Custom Propositions - a proposition which can optionally be parameterized by some set of values. Evaluation logic is implemented "by hand" and returns true or false.

The data model is designed in such a way to support each of these.

Next we'll look at each of the proposition tables in detail.

## Proposition - krms_prop_t

Every proposition in the repository will have an entry in this table. Propositions are referenced by a rule or another proposition (in the case of compound propositions). Propositions are never re-used across multiple rules.

Here is a summary of the non-common data elements in this proposition table:

### Table 2.1. Non-common data elements in the proposition table

| Column | Description |
|---|---|
| prop_id | A generated primary key identifier for the proposition |
| desc_txt | A plain-text description of the proposition |
| typ_id | Defines the PropositionType for the proposition. Defined in the krms_typ_t table. |
| dscrm_typ_cd | Discriminator type code which defines if the proposition is compound or simple. Valid values are C and S. |

## Proposition Parameters - krms_prop_parm_t

Each proposition can have zero or more parameters. The proposition parameter is the primary data element used to define the proposition. These parameters will be one of the following three types:

1. Constant Values
   - numbers
   - strings
   - dates

- etc.

2. Terms
   - data available in the execution environment and/or resolved by a term resolver
3. Functions
   - resolve to a value
   - Take a fixed number of parameters
4. Operators
   - one of a set of built-in "functions"
   - The full set of (currently) supported operators are as follows:
     - =
     - !=
     - >
     - <
     - >=
     - <=
   - custom operators can be can be configured for use in specific Contexts.

To that end, the proposition parameter list should be modeled as a list in Reverse Polish Notation (RPN). This allows for arbitrary nesting of parameters, which may have parameters of their own. However, this requires that each function explicitly define the number of arguments that it expects. This will be specified when the function is defined, so the proposition system can assume this is available. This requirement does prohibit the use of functions that have a variable arity since the model currently does not have anyway to group parameters. So this will currently be unsupported.

Examples of proposition parameter lists defined using RPN are as follows:

- [campusCode, "BL", =] *equivalent to* campusCode="BL"
- [totalDollarAmount, availableAmount, >] *equivalent to* totalDollarAmount > availableAmount
- [award, getTotalDollarAmountForAward, award, getAvailableAmountForAward, >] *equivalent to* getTotalDollarAmount(award) > getAvailableAmountForAward(award)

In the cases above the following are constants:

- "BL"

The following are terms:

- campusCode
- totalDollarAmount

- availableAmount
- award

The following are functions:

- getTotalDollarAmountForAward
- getAvailableAmountForAward

And the following are operators:

- =
- >

Here is a summary of the non-common data elements in this proposition parameter table:

### Table 2.2. Non-common data elements in the proposition parameter table

| Column | Description |
|---|---|
| prop_parm_id | A generated primary key identifier for the proposition parameter |
| prop_id | The proposition which this parameter applies to |
| parm_val | the value of the parameter |
| parm_typ_cd | Indicates whether the parameter value represents a constant, term, or function. Valid values are C, T, F, O |
| seq_no | Defines the order of the parameter within the larger parameter list. |

# KRMS Types

Many of the key concepts in KRMS are implemented by what are known as **type services**, and are customizable or pluggable using the service bus and the KRMS type table. You can add custom attributes and behavior that affects the options and execution behavior associated with a number of system components:

- ActionTypeService

  Enables custom actions that can be configured on rules.

- AgendaTypeService

  Enables agendas that support custom execution behavior and have custom attributes associated with them.

- FunctionTypeService

  Defines reusable execution logic (functions) that can be utilized at the proposition level during rule execution. Note that at present, functions are not directly supported within the rule authoring user interface.

- RuleTypeService

  Enables custom rule-level execution and action triggering behavior, as well as custom attributes at the Rule level.

- TermResolverTypeService

Enables custom resolution/reification of runtime values for Terms.

- CustomOperator

  Enables richer expressions within simple propositions.

# Example Rule

Previously if a newly hired employee met a set of criteria, they would need to have an eVerify check ran in addition to the standard I-9 process. The initial requirement stated that if the original hire date of an employee is greater that August 1, 2006, the grant is of a Government type, the grant amount is greater than $5,000, the grant is longer than 2 months, and the employee has not previously completed an eVerify check then their hire document needs to be routed to the special HR eVerify workgroup.

...or put another way

```
IF hr.hire_date > "08/01/2006) and cg.grant_type = "GOV" and
cg.grant_amount > "$5,000" and cg.grant_duration > 60 and
hr.eVerify_comp = "N" THEN ...
```

# KRMS Terms and Concepts

- **Agenda** - a collection of rules in a defined plan.

  KRMS agendas support conditional logic, e.g.

  - rule 1: is initiator admin?
    - When FALSE:
      - rule 2: initiator has special permission?
      - rule 3: exceeds dollar amount threshold?

- **Rule** - the logical expression in an Agenda. It consists of two parts, a proposition (condition or statement) that returns a true of false value, and an action or set of actions. The entire example presented above is a rule.

- **Proposition** - the logic that makes up a rule. A proposition is a single expression of logic that returns true or false. Propositions can be compounded, created using AND, OR, or both to create more complex logic. The entire IF statement in the example above comprises a compound propositio

- **Action** - the steps to perform in the event that the rule, after being evaluated against the propositions, returns TRUE. In the example stated above this would involve routing the document to the HR eVerify workgroup and presenting a warning on the document. Other examples of actions include:

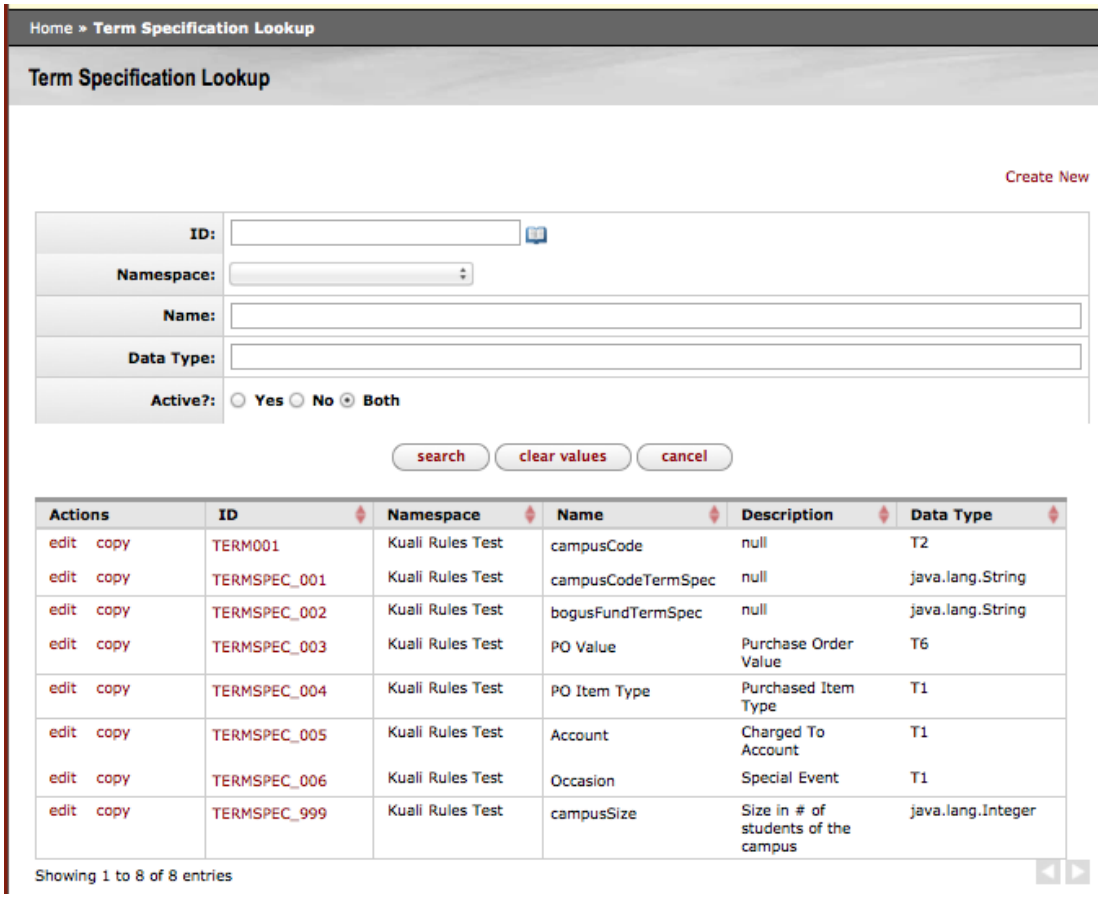
  - Route to a PeopleFlow
  - Present a questionnaire
  - Display a validation error

- **Term** - the definition of data that is evaluated in a proposition. hr.hire_date, cg.grant_type, etc. are examples of Terms in the working example.

- **Term Resolver** - the term specification that is evaluated in a proposition. Each term (for example, hr.hire_date, cg.grant_type, etc.) is given a specification which includes:

  Namespace
  Name
  Data Type
  Context Id
  Context Namespace
  Category Id
  Category Namespace

  This information is maintained and viewed via the Term Specification Lookup option from the Main Menu.




- **Fact** - the actual data for the term being evaluated in a proposition, the data being passed in for evaluation. In the example above, if the grant in question had an amount of $10,000, then $10,000 would be the fact.

- **Context** - a collection of agendas, rules, terms, term specifications. In our example a context of "HReVerify" would be established for easy identification of the items related to these business rules.

# Chapter 3. KRMS Administration Guide

(work in progress - content tdb. The below preface is patterned after the KEW TRG - what will admins need to administer for KRMS? I've put in some placeholder content-topics for a TOC skeleton.)

This guide provides information on administering a Kuali Rules Management System (KRMS) installation. Out of the box, KRMS 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 KRMS' administration tools.

# Initial Set up tasks

In this section we cover the types of tasks you'll need to do as a one-time setup at your institute in order for you and others to be able to define KRMS agendas for use by applications.

## What do I have to install so that people can use KRMS?

## What do I have to create or customize so that people can work with business contexts, agendas, and rules?

Below are the constructs you will need to point to or create for your institute:

- Use existing Namespaces or set up Namespaces for KRMS
- Use an existing Agenda Type service or set up an Agenda Type service for KRMS
- Use existing Types or set up Types for KRMS
- Use existing Contexts or configure new Contexts for KRMS
- Specify Terms
- Create Term Resolvers
- Create Parameterized Terms
- Create Custom Operators

Below are the instructions for doing these tasks.

### Point to or Set up Namespaces

You can use existing Namespaces or set up additional Namespaces specifically for KRMS use. Namespaces are used throughout Rice, the From the Administration tab of the Rice portal in the Configuration section you'll see the link to the Namespace lookup, from which you can view existing or create new namespaces.

### Point to or Set up an Agenda Type service for KRMS

You can use an existing Agenda Type service or set up an Agenda Type service specifically for KRMS (include information on how to do both of these).

For example, below is a snippet of Spring configuration for defining an Agenda Type service. The first bean definition configures the service implementation instance. The class being referenced implements the **AgendaTypeService** interface. The second bean definition exports the service to the service bus, which is required when the client application is running remotely to a Rice standalone server. It is being exported here to the **KR-SAP** namespace with the service name **campusAgendaTypeService**:

```
<bean id="campusAgendaTypeService"
      class="edu.sampleu.krms.impl.CampusAgendaTypeService">
    <property name="configurationService" ref="configurationService"/>
</bean>

<bean id="campusAgendaTypeService.exporter"
      class="org.kuali.rice.ksb.api.bus.support.CallbackServiceExporter"
      p:serviceBus-ref="rice.ksb.serviceBus"
      p:callbackService-ref="campusAgendaTypeService"
      p:serviceNameSpaceURI="KR-SAP"
      p:localServiceName="campusAgendaTypeService"
      p:serviceInterface="org.kuali.rice.krms.framework.type.AgendaTypeService"/>

```

## Point to or Set up the Types for KRMS

Below is example SQL code to insert the Type into the Agenda Type service -- be sure to replace the content of the 2nd parenthetical expressions in each of the following examples with your defined values:

- First, add the Type(s) itself:

  insert into krms_typ_t (typ_id, nm, nmspc_cd, srvc_nm, actv, ver_nbr) values (*'T6', 'Campus Agenda', 'KRMS_TEST', 'campusAgendaTypeService', 'Y', 1*);

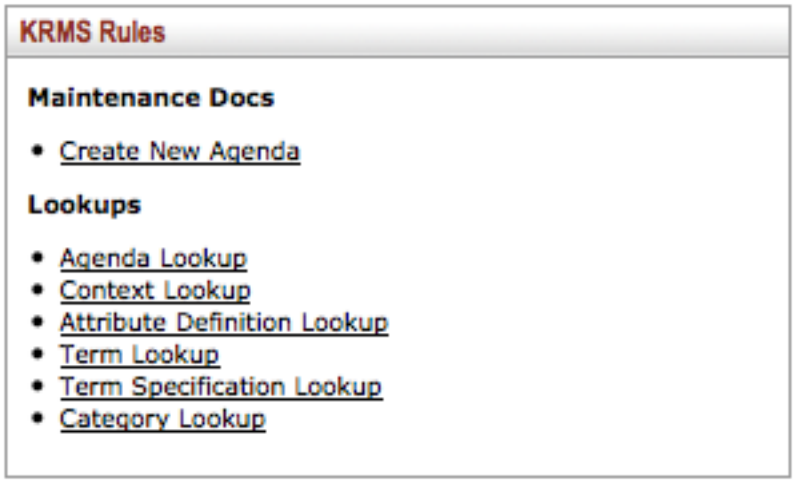
- Next, add the campus attribute(s) to the Campus Agenda Type:

  insert into krms_attr_defn_t (ATTR_DEFN_ID, NM, NMSPC_CD, LBL, CMPNT_NM, DESC_TXT) values (*'Q9901', 'Campus', 'KRMS_TEST', 'campus label', null, 'the campus which this agenda is valid for'*);

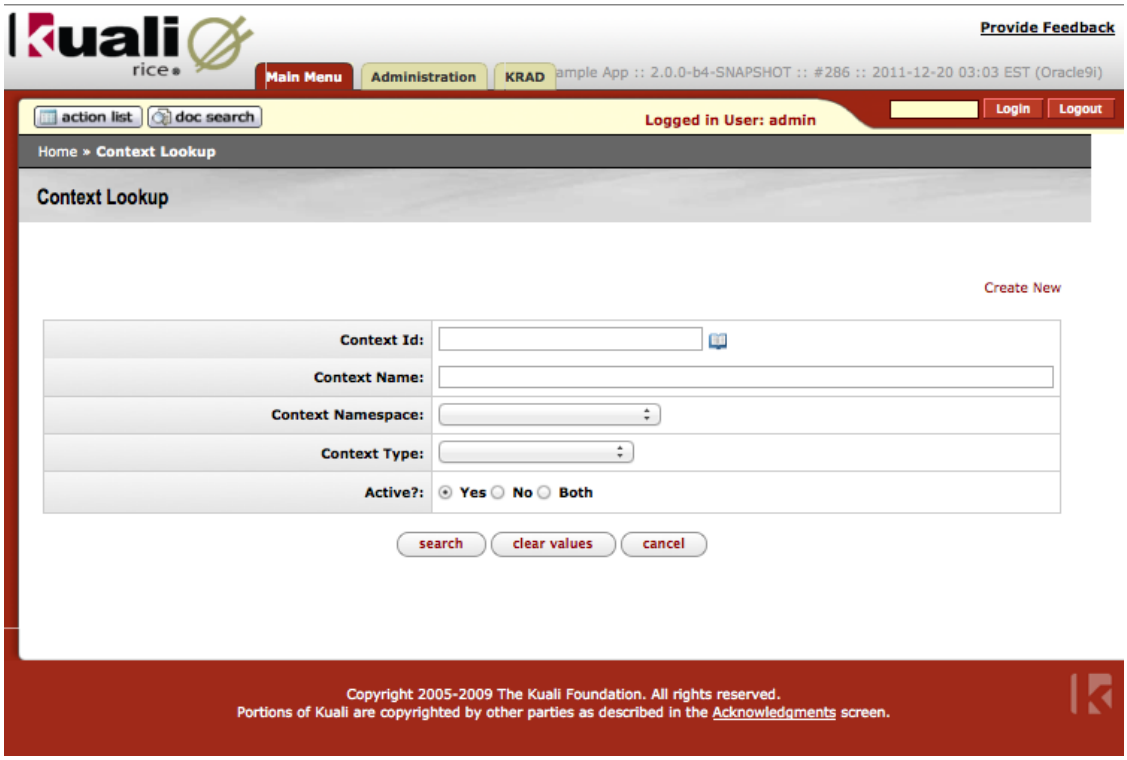
  insert into krms_typ_attr_t (TYP_ATTR_ID, SEQ_NO, TYP_ID, ATTR_DEFN_ID) values (*'T6A', 1, 'T6', 'Q9901'*);

## Point to or Set up Contexts for KRMS

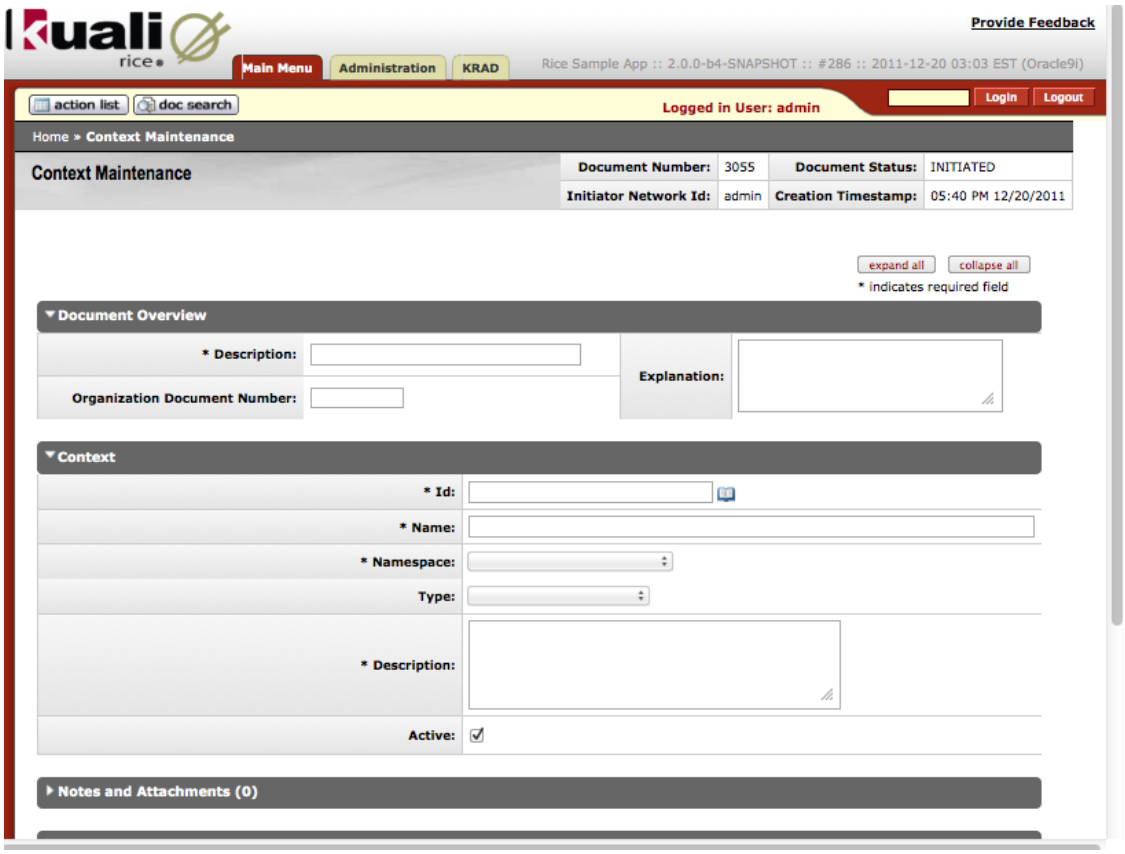
You can use existing Contexts or configure new Contexts for KRMS. There is graphical user interface support for configuring a new Context, through a maintenance page. For example, in the Rice demo / sample application, on the Main menu page, under KRMS Rules, select the Context Lookup.




You can search for existing Contexts or create a new one. To create a new one, select "Create New" at the top right on the context lookup page:




The resulting Context Maintenance screen enables you to define a new Context. The Context ID must be unique:




After creating your Context(s), you must 1) set "CampusAgendaType" as valid*, 2) set "Route to PeopleFlow" action as valid* for them, and 3) make the Type(s) you created valid for your Context(s). See the following examples, and replace the content of each of the 2nd parenthetical expressions with your defined values:

- insert into krms_cntxt_vld_agenda_t (cntxt_vld_agenda_id, cntxt_id, agenda_typ_id, ver_nbr) values (*'agendaid', 'contextid', 'agendatypeid', version#*);
- insert into krms_cntxt_vld_actn_t (cntxt_vld_actn_id, cntxt_id, actn_typ_id, ver_nbr) values (*'agendaid', 'contextid', 'agendatypeid', version#*);
- insert into krms_cntxt_vld_agenda_t (cntxt_vld_agenda_id, cntxt_id, agenda_typ_id, ver_nbr) values (*''agendaid', 'contextid', 'agendatypeid', version#*);

## Specify the Terms for KRMS

You can point to existing terms or specify new terms for KRMS (include information on how to do both of these).

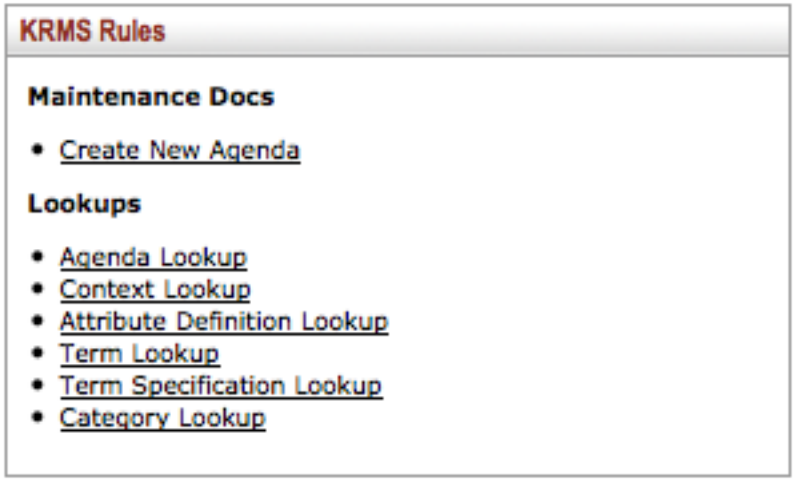
To specify newTerms, you will probably want to first create term categories. See the following examples, and replace the content of each of the 2nd parenthetical expressions with your defined values:

- Example - Generic Workflow Properties
- • insert into krms_ctgry_t (ctgry_id, nm, nmspc_cd, ver_nbr) values (*'CAT02', 'Workflow Document Properties', 'KR-SAP', '1'*);

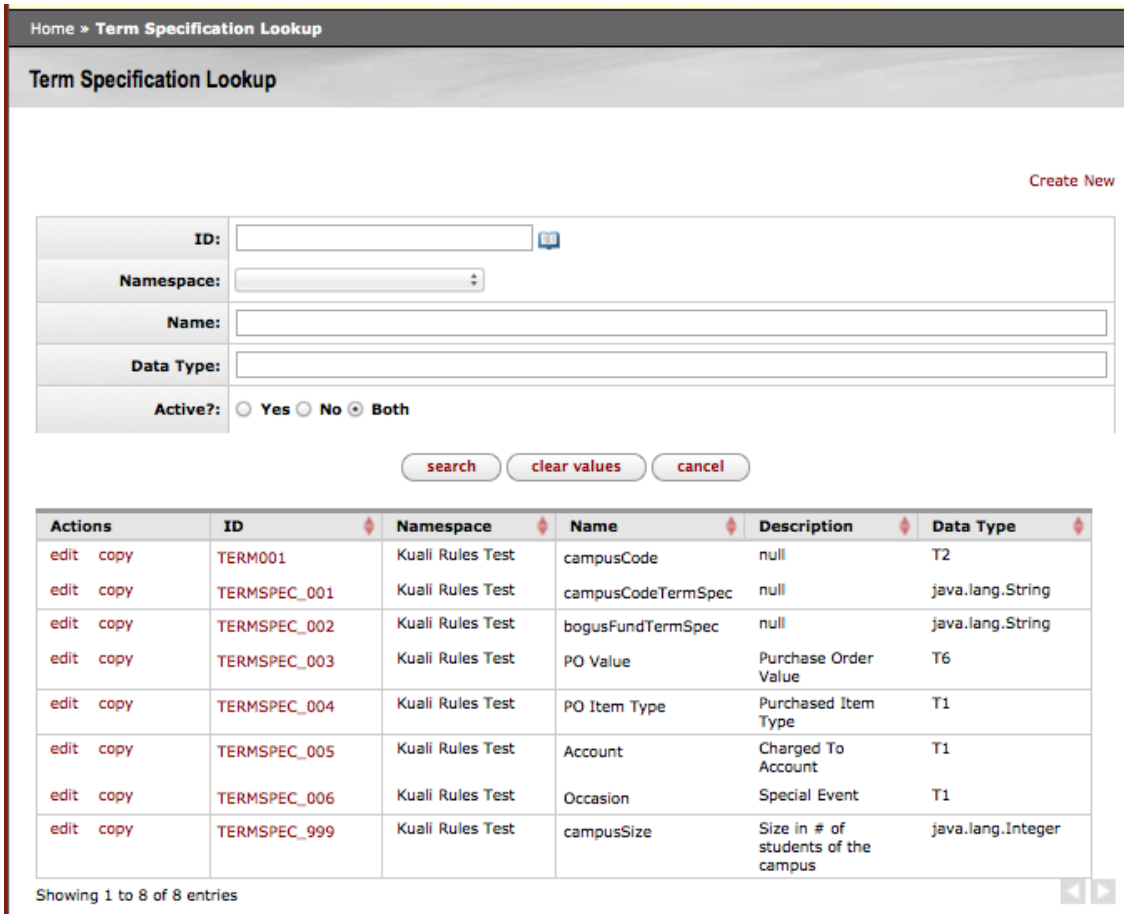
- Example - Travel Account Properties

- • insert into krms_ctgry_t (ctgry_id, nm, nmspc_cd, ver_nbr) values (*'CAT03', 'Travel Account Properties', 'KR-SAP', '1'*);

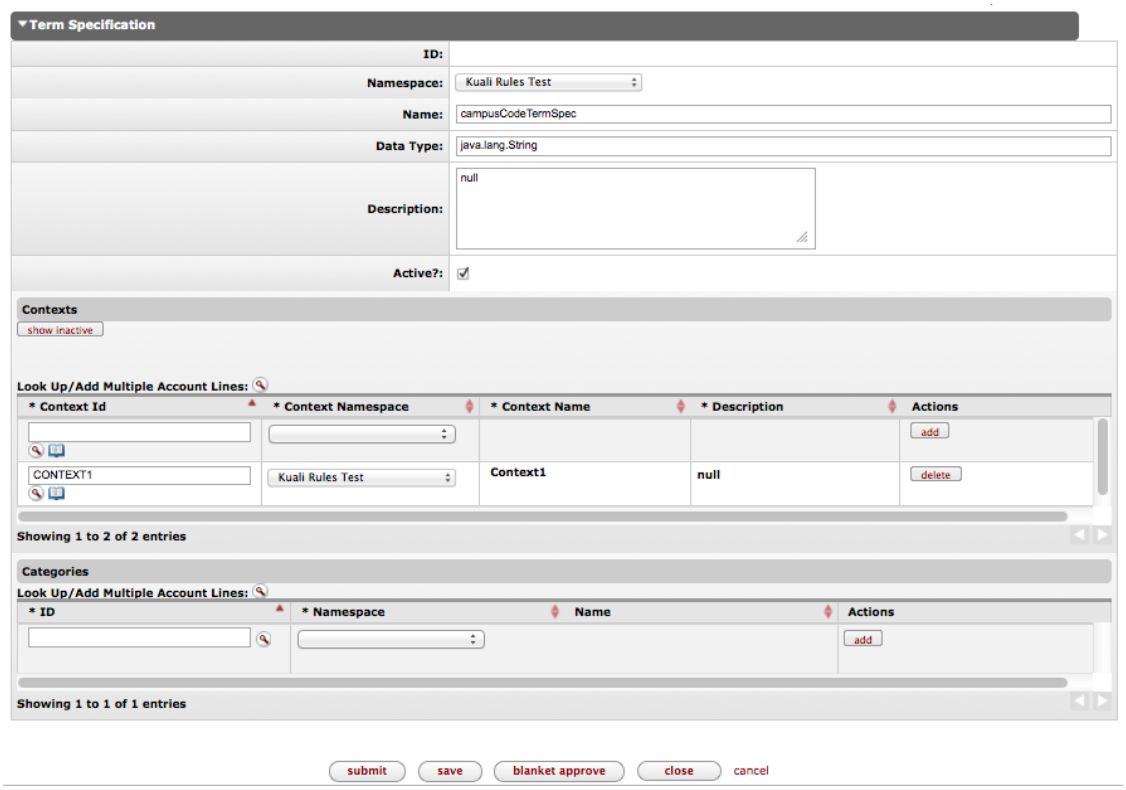
And next, you can use existing Terms or configure new Terms for KRMS. There is graphical user interface support for configuring a new Term, through a maintenance page. For example, in the Rice demo / sample application, on the Main menu page, under KRMS Rules, select the Term Specification Lookup and, after completing that, select the Term Lookup.




You can search for existing Term Specifications and Terms or create new ones. To create a new one, select "Create New" at the top right on the term specification lookup page or copy and then modify an existing one. See example Term Specification Lookup screen below:

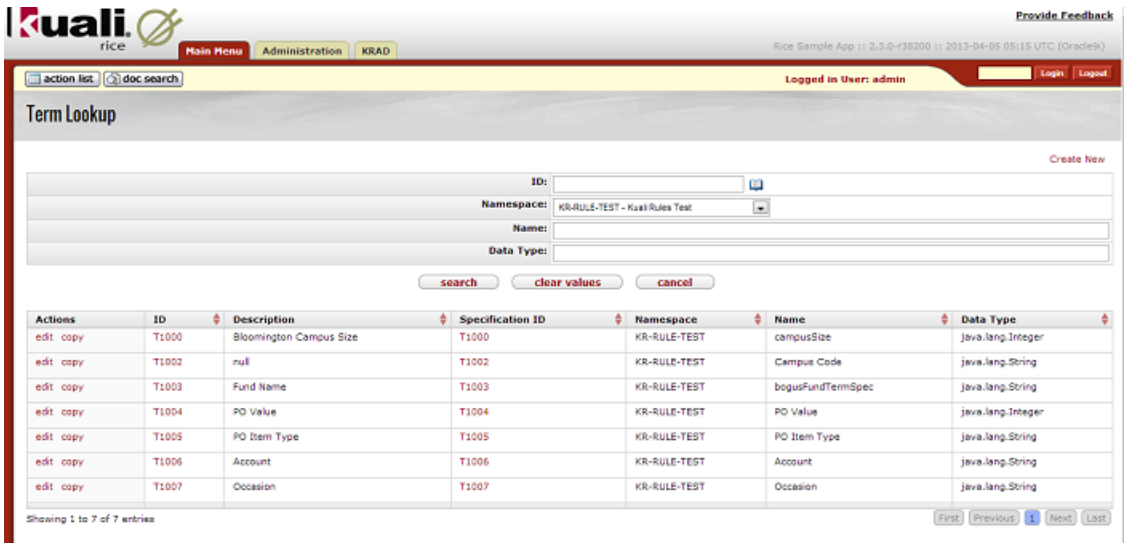



If you copy an existing term specification, be sure to give it a new and unique name before you change and save or submit it. Below is a view of the term specification screen showing the types of attributes you can associate with it.



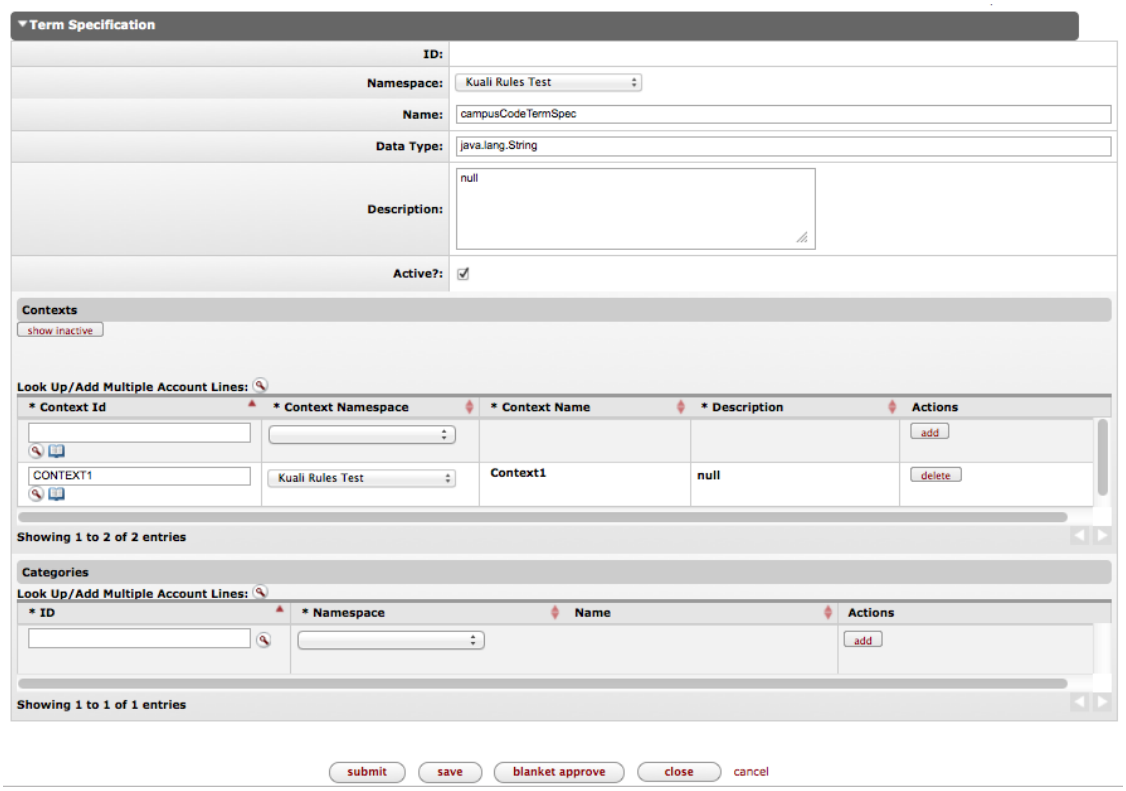

After creating your term specifications (your categories of terms), you can use the Term Lookup screen to add or create new terms. See the example Term Lookup screen below:

### Figure 3.1. Term Lookup screen example




If you copy an existing term, be sure to change the name to a new and unique term before you save or submit it. Below is a view of the term specification screen showing the types of attributes you can associate with it.

**Figure 3.2. Term specification screen example**




# Create Custom Operators

Custom Operators give additional flexibility and power by allowing you to expand the kinds of logical propositions that can be expressed. For example, you could add a "matches" operator that would allow wildcarding instead of requiring strict string equality.

Custom Operators leverage a executable component that is loaded through a **FunctionTypeService**. The Custom Operator implementation references that service indirectly by providing a **FunctionDefinition** to the system that identifies the name of the function and the service that provides it in executable form.

Adding a Custom Operator to KRMS requires:

- development of a Java class implementing several methods
- Spring configuration to wire it up and export it to the service bus
- the addition of some database rows in the KRMS schema to configure the KRMS type and make it usable from some Contexts.

This process is outlined below.

The Java class must implement the **org.kuali.rice.krms.api.repository.operator.CustomOperator** interface, which has just two methods. The first of those methods, *getOperatorFunctionDefinition*, provides the **FunctionDefinition** that the engine will use to find the **FunctionTypeService**. The second method, *validateOperandClasses*, performs simple validation of the arguments based on their declared Java types.

It will be most convenient to implement **org.kuali.rice.krms.framework.type.FunctionTypeService** as well with your custom operator service class. In that case, you will need to implement a third

method, *loadFunction*, that provides back the executable form for the function. An example of a Custom Operator that implements both can be found in the Rice sample application module, the class name being **edu.sampleu.krms.impl.ContainsOperator**.

Here is the Spring configuration for this Custom Operator. The first bean definition wires up the service implementation instance. The second bean definition exports the service to the service bus. It is being exported here to the **KR-SAP** namespace with the service name **sampleAppContainsOperatorService**:

```
<bean id="sampleAppContainsOperatorService"
  class="edu.sampleu.krms.impl.ContainsOperator"/>

<bean id="sampleAppContainsOperatorService.exporter"
        class="org.kuali.rice.ksb.api.bus.support.CallbackServiceExporter"
        p:serviceBus-ref="rice.ksb.serviceBus"
        p:callbackService-ref="sampleAppContainsOperatorService"
        p:serviceNameSpaceURI="KR-SAP"
        p:localServiceName="sampleAppContainsOperatorService"
        p:serviceInterface="org.kuali.rice.krms.api.repository.operator.CustomOperator"/>
```

The remainder of the work is done in the database in three discrete parts:

- Identify the Custom Operator on the service bus as a KRMS type
- Flag our custom operator as usable within Context of a certain type.
- Specify some metadata about our executable function.

First, we identify our custom operator as a KRMS type by creating an entry in the KRMS_TYP_T table. You will not want to use this SQL directly, rather you will want to change the values as appropriate. The significance of the columns and how you will want them set will be discussed below.

```
 insert into KRMS_TYP_T (TYP_ID, NM, NMSPC_CD, SRVC_NM, ACTV, VER_NBR)
    values ('OPERATOR-KRMS-TYPE-ID', 'contains operator', 'KR-SAP', 'sampleAppContainsOperatorService', 'Y',
 1);
```

### Table 3.1. Columns being inserted into KRMS_TYP_T

| Column | Description |
|---|---|
| TYP_ID | the primary key column for the KRMS_TYP_T table. You will want to set it to a unique value, one that will not collide with a value from the KRMS_TYP_S sequence in the future. |
| NM | the name of the type service, for descriptive purposes. |
| NMSPC_CD | the namespace code that the service is exported to the bus within. |
| SRVC_NM | the name that the service is exported to the bus with. |
| ACTV | the active status of your type, you'll want to set this to 'Y'. |
| VER_NBR | used for optimistic locking, you'll want to set this to '1'. |

Note that the **FunctionDefinition** returned by your service will need to agree with the values configured here. You can achieve that dynamically by loading it with the FunctionBoService using the service namespace and name.

Next we will create the type-type relation that will make your Custom Operator usable within a certain Context. Again, you will not want to use this SQL directly, rather you will want to change the values as appropriate. The significance of the columns and how you will want them set will be discussed below.

```
insert into KRMS_TYP_RELN_T (TYP_RELN_ID, FROM_TYP_ID, TO_TYP_ID, RELN_TYP, SEQ_NO)
          values ('A-UNIQUE-ID', 'CONTEXT-TYPE-ID', 'OPERATOR-KRMS-TYPE-ID', 'A', 1);
```

### Table 3.2. Columns being inserted into KRMS_TYP_RELN_T

| Column | Description |
|---|---|
| TYP_RELN_ID | the primary key column for the KRMS_TYP_RELN_T table. You will want to set it to a unique value, one that will not collide with a value from the KRMS_TYP_RELN_S sequence in the future. |
| FROM_TYP_ID | The TYP_ID value from your Context. The custom operator will be available in any Contexts having this TYP_ID. |
| TO_TYP_ID | the TYP_ID value you used when creating your KRMS type in the previous step. |
| RELN_TYP | the type of relationship, which you should set to 'A' indicating that usage is allowed. |
| SEQ_NO | Used for order of presentation in some other cases. Set this to 1. |

Finally we will configure our function definition:

```
insert into KRMS_FUNC_T
           (FUNC_ID, NMSPC_CD, NM, DESC_TXT, RTRN_TYP, TYP_ID, ACTV, VER_NBR)
         values
           ('A-UNIQUE-ID', 'KR-SAP', 'contains', 'descr...', 'java.lang.Boolean', 'OPERATOR-KRMS-TYPE-ID',
 'A', 1);
```

### Table 3.3. Columns being inserted into KRMS_FUNC_T

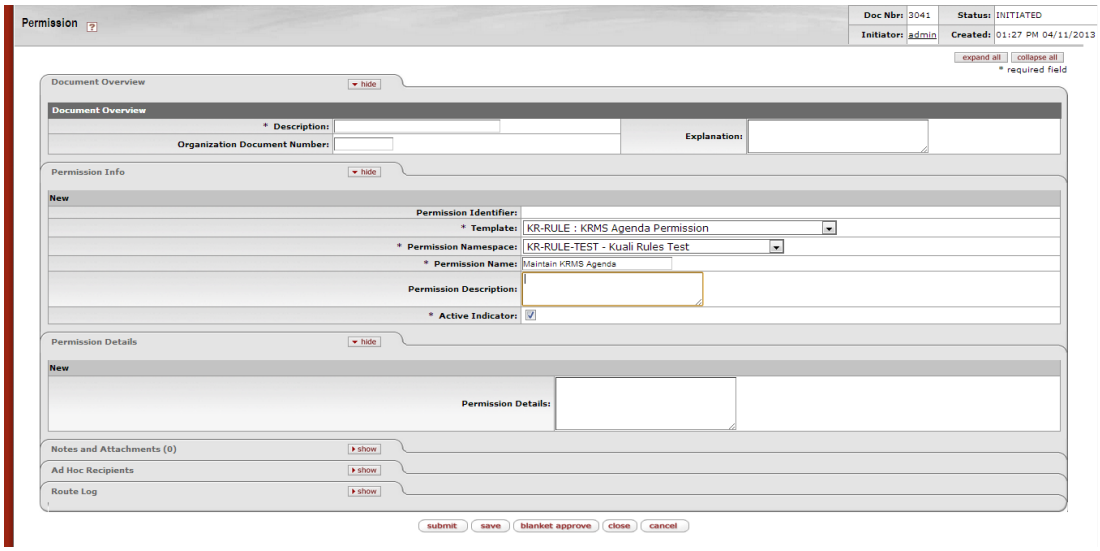
| Column | Description |
|---|---|
| FUNC_ID | the primary key column for the KRMS_FUNC_T table. You will want to set it to a unique value, one that will not collide with a value from the KRMS_FUNC_S sequence in the future. |
| NMSPC_CD | the code for the namespace that your function belongs in. Technically, this doesn't have to match that of the corresponding service, but in most cases it makes sense for it to do so. |
| NM | the name of the function, which is used to identify the custom operator in the user interface. This is what will show up in the operator selector widget. |
| DESC_TXT | some descriptive text about the function. |
| RTRN_TYP | the Java type of your function return value, set this to 'java.lang.Boolean'. |
| TYP_ID | the TYP_ID value for your FunctionTypeService (which will, if you implement both interfaces in one class, be the same as for your CustomOperator service). |
| ACTV | the active status of your function, you'll want to set this to 'Y'. |
| VER_NBR | used for optimistic locking, you'll want to set this to '1'. |

With that configuration complete, you should be able to see the new operator available when editing propositions for any Agenda whose Context has the type that we configured our type-type relation with in the second SQL step.

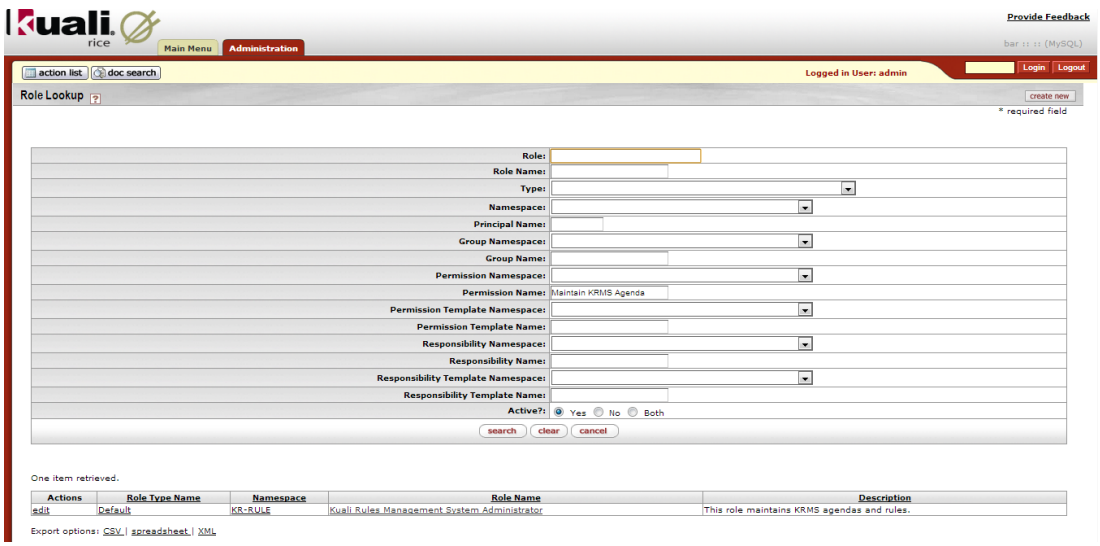
# Do I have to define permissions or approval processes so that people can create and maintain KRMS agendas and rules?

The answer is yes. To set up Permissions for creating and maintaining agendas, you can use the maintenance screens for identities and for role/group/permission/responsibility-type that are available on the Administration tab in the Kuali Rice portal:

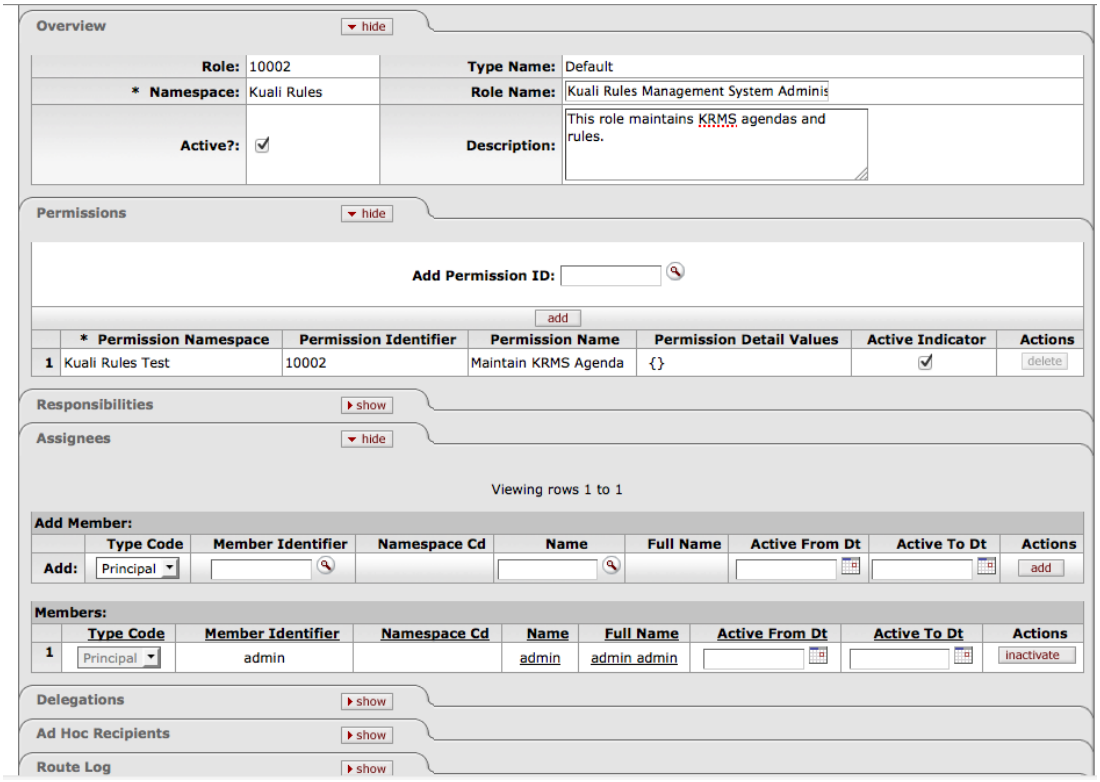
- First, in the Kuali Rice portal, go to the Administration tab, and in the Identity category, select "Permission". Create a new permission: give it a unique Permission name and select the namespace you'd like to associate it with. For example, give it the name 'Maintain KRMS Agenda' and select the appropriate namespace.




- Next, add that permission to a role that your agenda-maintaining user has. You can do that through the Kuali Rice portal, by going to the Administration tab, and in the Identity category, select "Role". You can search for the role you just created (in the example above, 'Maintain KRMS Agenda', and select to edit it.




Add the individuals you would like to this role, so that they will have the permissions needed to create and maintain agendas and rules. Click on the **`edit`** action to bring up the following screen to add the individuals to this role.




Alternatively, you could set up these permissions via code - see SQL server code examples below:

```
insert into krim_role_perm_id_s values (null);

insert into krim_role_perm_t (role_perm_id, role_id, perm_id, actv_ind,
ver_nbr, obj_id) values ((select max(id) from krim_role_perm_id_s),
<YOUR_ROLE_ID>, <YOUR_PERMISSION_ID>, 'Y', 1, uuid());
```

# Chapter 4. The KRMS User Interface

## KRMS Agenda Editor

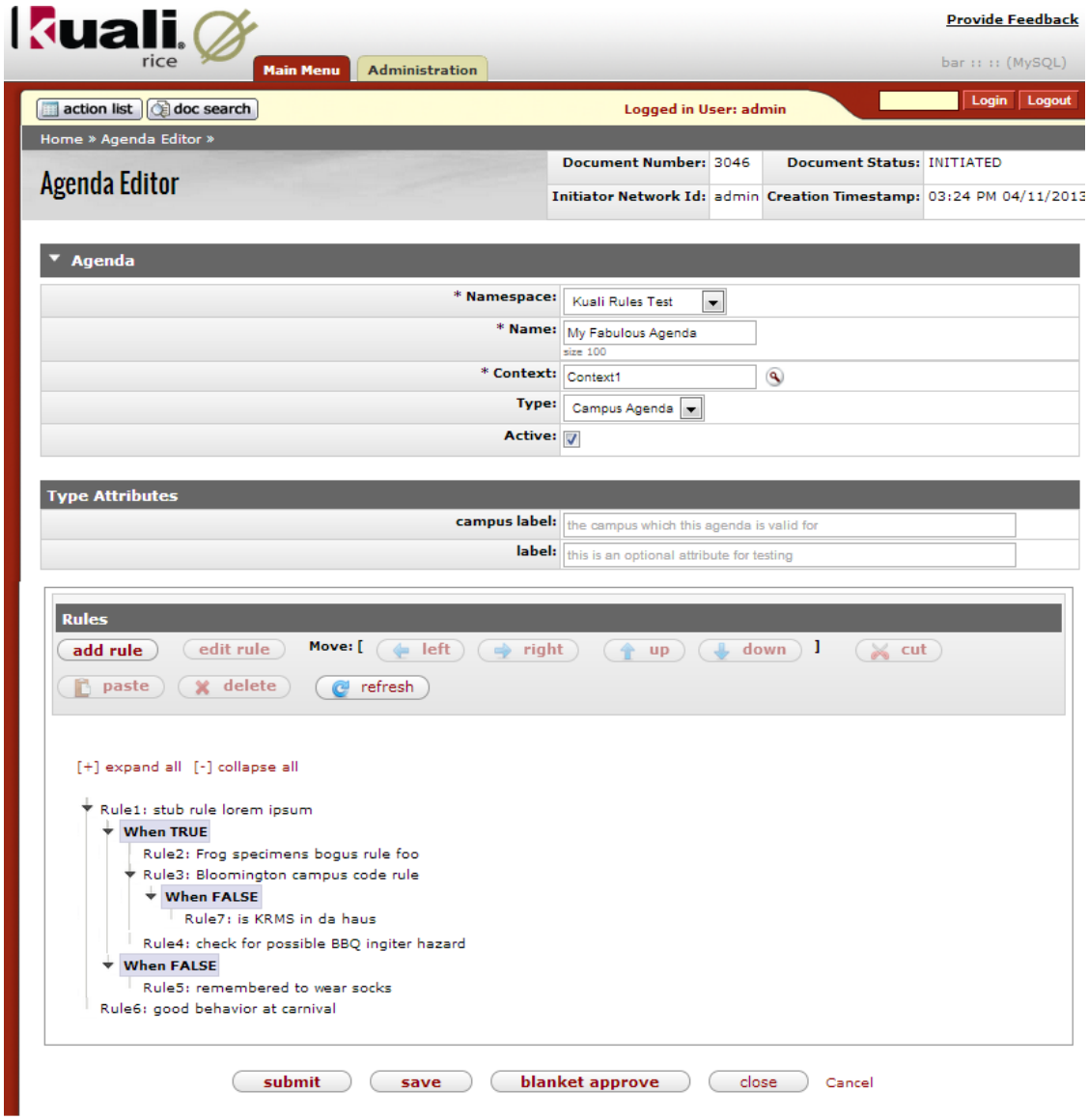
Rules in KRMS are placed into ordered sets called Agendas. The order of the Rules in an Agenda determines the sequencing: which rule gets evaluated first, second and so on. The Agenda also enables you to include conditional branching logic between Rules.

In turn, Agendas are created in Contexts, which may represent any categories that are relevant within your institution. For example, they could correspond to document types or business processes or any other categories. In some university environments, the following might be relevant contexts: Awards, Proposals, IRB reviews, Course co-requisites, Course pre-requisites, Student plan evaluations, and so on.

Each Context contains its own agendas, and each Agenda contains its own rules. Rules aren't shared across agendas (though you can copy/paste them, in which case they become unique Rule instances), and Agendas aren't shared across Contexts. There is no Context hierarchy; that is, Agendas and Rules can't be inherited across contexts within any sort of hierarchy.

See below for a view of the Agenda Editor in KRMS.

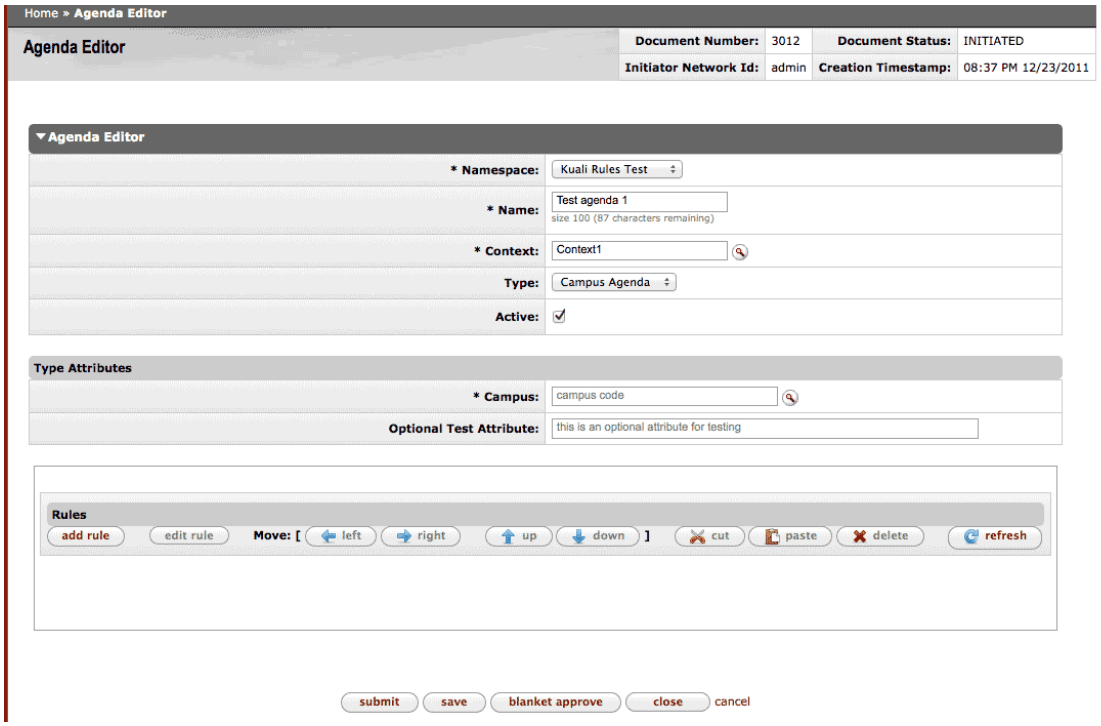
**Figure 4.1. KRMS Agenda Editor**




Rules in the Agenda can be selected by clicking on them. The order and conditional logic within the agenda can be manipulated by selecting rules and clicking the buttons on the toolbar above the Rules. Clicking on the **`Add Rule`** button on the Agenda Editor screen will take you to the Rule Editor with a new blank Rule, and selecting a Rule and clicking on the **`Edit Rule`** button will open that existing Rule in the Rule Editor.

And see below for an example of how attributes can be progressively rendered in KRMS. In this example, the selected context, "Context 1", requires the selection of a type, and the selected type, "CampusAgendaType", requires some additional attributes, that are not required by all types. These are shown to the end user only when required. This is an example of KRAD's progressive disclosure capability:

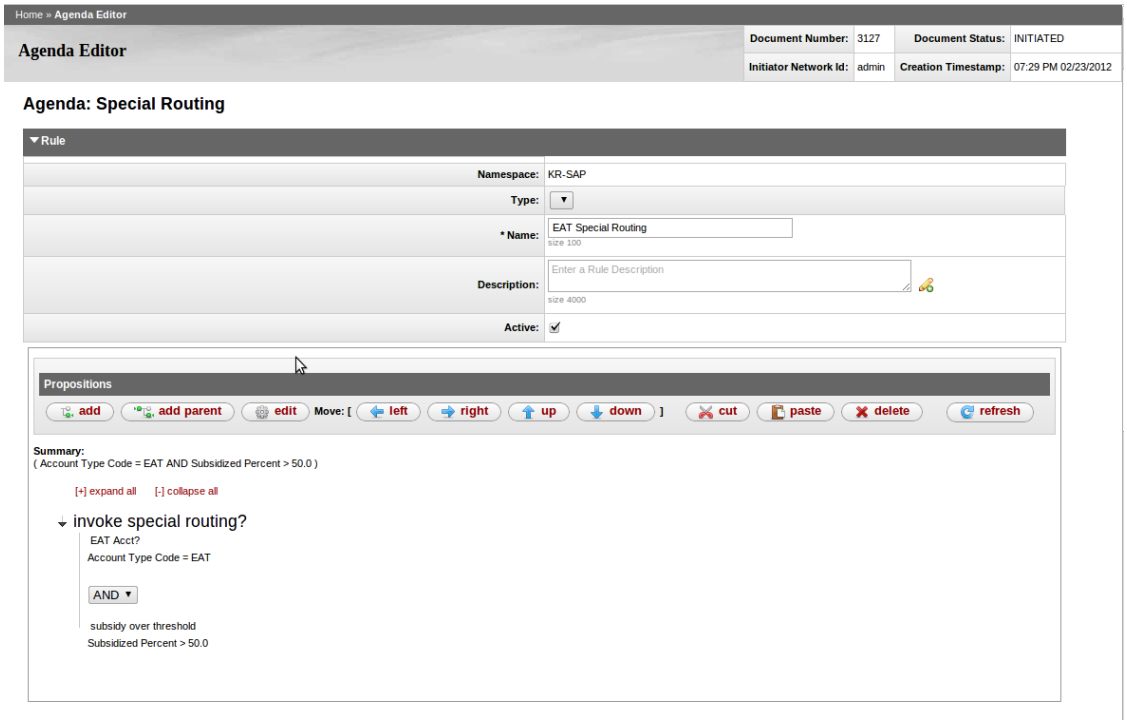
**Figure 4.2. KRMS Agenda Editor with additional attributes displayed**




# KRMS Rule Editor

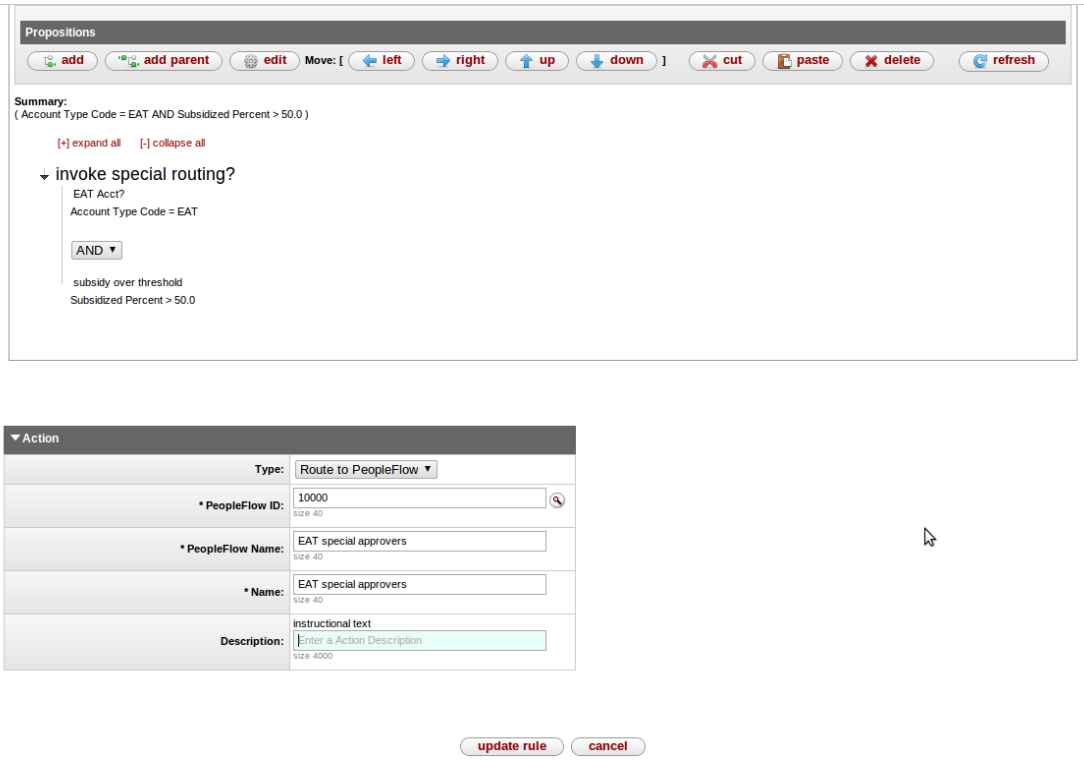
See below for views of editing a Rule from an Agenda in KRMS.

**Figure 4.3. KRMS Rules Editor**



And below is the same Rule but scrolled down further on the page to show the Action configuration as well. In this example, when the logical proposition for this rule is satisfied (when it is true), the rule will call a PeopleFlow to route a request to it.

### Figure 4.4. KRMS proposition and PeopleFlow Action




You can add Simple Propositions via **`add`**, or Compound Propositions via **`add parent`**. Simple Propositions are configured with a Term and a user-supplied value to compare it with.

# 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:<br>• Approve: requests an approve or disapprove action.<br>• 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.<br>• Acknowledge: requests an acknowledgment by the user that the document has been opened - the doc will not leave the Action List until acknowledgment has occurred; however, the document routing will not be held up and the document will be permitted to transaction into the processed state if neccessary.<br>• 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:<br>• Acknowledge: requests that the users states he or she has reviewed the document.<br>• Approve: requests that the user either Approve or Disapprove a document.<br>• Complete: requests the user to enter additional information in a document so that the content of the document is complete.<br>• 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:<br>• Acknowledged: Reviewer has viewed and acknowledged document.<br>• Approved: Reviewer has approved the action requested on document. |

- Blanket Approved: Reviewer has requested a blanket approval up to a specified point in the route path on the document.
- Canceled: Reviewer has canceled the document. The document will not be routed to any more reviewers.
- Cleared FYI: Reviewer has viewed the document and cleared all of his or her pending FYI(s) on this document.
- Completed: Reviewer has completed and supplied all data requested on document.
- Created Document: User has created a document
- Disapproved: Reviewer has disapproved the document. The document will not being routed to any subsequent reviewers for approval. Acknowledge Requests are sent to previous approvers to inform them of the disapproval.
- Logged Document: Reviewer has added a message to the Route Log of the document.
- Moved Document: Reviewer has moved the document either backward or forward in its routing path.
- Returned to Previous Node: Reviewer has returned the document to a previous routing node. When a Reviewer does this, all the actions taken between the current node and the return node are removed and all the pending requests on the document are deactivated.
- Routed Document: Reviewer has submitted the document to the workflow engine for routing.
- Saved: Reviewer has saved the document for later completion and routing.
- Superuser Approved Document: *Superuser* has approved the entire document, any remaining routing is cancelled.
- Superuser Approved Node: Superuser has approved the document through all nodes up to (but not including) a specific node. When the document gets to that node, the normal Action Requests will be created.
- Superuser Approved Request: Superuser has approved a single pending Approve or Complete Action Request. The document then goes to the next routing node.
- Superuser Cancelled: Superuser has canceled the document. A Superuser can cancel a document without a pending Action Request to him/her on the document.
- Superuser Disapproved: Superuser has disapproved the document. A Superuser can disapprove a document without a pending Action Request to him/her on the document.

| | |
|---|---|
| | • Superuser Returned to Previous Node: Superuser has returned the document to a previous routing node. A Superuser can do this without a pending Action Request to him/her on the document. |
| Activated | The state of an action request when it is has been sent to a user's Action List. |
| Activation | The process by which requests appear in a user's Action List |
| Activation Type | Defines how a route node handles activation of Action Requests. There are two standard activation types:<br>• 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.<br>• Parallel: All Action Requests at the route node are activated immediately, regardless of priority |
| Active Indicator | An indicator specifying whether an object in the system is active or not. Used as an alternative to complete removal of an object. |
| Ad Hoc Routing | A type of routing used to route a document to users or groups that are not in the Routing path for that Document Type. When the Ad Hoc Routing is complete, the routing returns to its normal path. |
| Annotation | Optional comments added by a *Reviewer* when taking action. Intended to explain or clarify the action taken or to advise subsequent Reviewers. |
| Approve | A type of workflow action button. Signifies that the document represents a valid business transaction in accordance with institutional needs and policies in the user's judgment. A single document may require approval from several users, at multiple route levels, before it moves to final status. |
| Approver | The user who approves the document. As a document moves through Workflow, it moves one route level at a time. An Approver operates at a particular route level of the document. |
| Attachment | The pathname of a related file to attach to a Note. Use the "Browse..." button to open the file dialog, select the file and automatically fill in the pathname. |
| Attribute Type | Used to strongly type or categorize the values that can be stored for the various attributes in the system (e.g., the value of the arbitrary key/value pairs that can be defined and associated with a given parent object in the system). |
| Authentication | The act of logging into the system. The Out of the box (OOTB) authenticaton implementation in Rice does not require a password as it is intended for testing puposes only. This is something that must be enabled as part of an implementation. Various authentication solutions exist, such as CAS or Shibboleth, that an implementer may want to use depending on their needs. |
| Authorization | Authorization is the permissions that an authenticated user has for performing actions in the system. |
| Author Universal ID | A free-form text field for the full name of the Author of the Note, expressed as "Lastname, Firstname Initial" |

# B

**Base Rule Attribute**

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

- Active: A true/false flag to indicate if the Routing Rule is active. If false, then the rule will not be evaluated during routing.
- Document Type: The *Document Type* to which the Routing Rule applies.
- From Date: The inclusive start date from which the Routing Rule will be considered for a match.
- Force Action: a true/false flag to indicate if the review should be forced to take action again for the requests generated by this rule, even if they had taken action on the document previously.
- Name: the name of the rule, this serves as a unique identifier for the rule. If one is not specified when the rule is created, then it will be generated.
- Rule Template: The Rule Template used to create the Routing Rule.
- To Date: The inclusive end date to which the Routing Rule will be considered for a match.

**Blanket Approval**

Authority that is given to designated *Reviewers* who can approve a document to a chosen route point. A Blanket Approval bypasses approvals that would otherwise be required in the *Routing* For an authorized Reviewer, the *Doc Handler* typically displays the Blanket Approval button along with the other options. When a Blanket Approval is used, the Reviewers who are skipped are sent Acknowledge requests to notify them that they were bypassed.

**Blanket Approve Workgroup**

A workgroup that has the authority to Blanket Approve a document.

**Branch**

A path containing one or more Route Nodes that a document traverses during routing. When a document enters a *Split Node* multiple branches can be created. A *Join Node* joins multiple branches together.

**Business Rule**

1. Describes the operations, definitions and constraints that apply to an organization in achieving its goals.
2. A restriction to a function for a business reason (such as making a specific object code unavailable for a particular type of disbursement). Customizable business rules are controlled by Parameters.

# C

**Campus**

Identifies the different fiscal and physical operating entities of an institution.

**Campus Type**

Designates a campus as physical only, fiscal only or both.

**Cancel**

A workflow action available to document initiators on documents that have not yet been routed for approval. Denotes that the document is void and should be disregarded. Canceled documents cannot be modified in any way and do not route for approval.

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

# D

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

| | |
|---|---|
| | For both *Primary* and *Secondary Delegation* the Delegate may act on any of the entries with the full authority of the Delegator. |
| Disapprove | A workflow action that allows a user to indicate that a document does not represent a valid business transaction in that user's judgment. The initiator and previous approvers will receive Acknowledgment requests indicating the document was disapproved. |
| Disapproved | A status that indicates the document has been disapproved by an approver as a valid transaction and it will not generate the originally intended transaction. |
| Doc Handler | The Doc Handler is a web interface that a Client uses for the appropriate display of a document. When a user opens a document from the Action List or Document Search, the Doc Handler manages access permissions, content format, and user options according to the requirements of the Client. |
| Doc Handler URL | The URL for the *Doc Handler*. |
| Doc Nbr | See *Document Number*. |
| Document | Also see *E-Doc*.<br><br>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.<br><br>Document Types have the following characteristics:<br><br>• They are specifications for a document that can be created in KEW |

| | |
|---|---|
| | • They contain identifying information as well as policies and other attributes<br>• They defines the Route Path executed for a document of that type (Process Definition)<br>• 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*.<br>• They are typically defined in XML, but certain properties can be maintained from a graphical interface |
| Document Type Hierarchy | A hierarchy of Document Type definitions. Document Types inherit certain attributes from their parent Document Types. This hierarchy is also leveraged by various pieces of the system, including the Rules engine when evaluating rule sets and KIM when evaluating certain Document Type-based permissions. |
| Document Type Label | The human-readable label assigned to a Document Type. |
| Document Type Name | The assigned name of the document type. It must be unique. |
| Document Type Policy | These advise various checks and authorizations for instances of a Document Type during the routing process. |
| Drilldown | A link that allows a user to access more detailed information about the current data. These links typically take the user through a series of inquiries on different business objects. |
| Dynamic Node | An advanced type of *Route Node* that can be used to generate complex routing paths on the fly. Typically used whenever the route path of a document cannot be statically defined and must be completely derived from document data. |

# E

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

| | |
|---|---|
| Entity | An Entity record houses identity information for a given Person, Process, System, etc. Each Entity is categorized by its association with an Entity Type. |
| Entity Attribute | Entities have directory-like information called Entity Attributes that are associated with them<br><br>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 interfact 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.<br><br>Groups can be given authorization to perform actions within applications by assigning them as members of *Roles*.<br><br>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.<br><br>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.<br><br>Group Attributes make up the identity information for a Group record.<br><br>Group Attributes can be maintained at runtime through a user interface that is capable of workflow. |

# H

| | |
|---|---|
| Hierarchical Tree Structure | A hierarchical representation of data in a graphical form. |

# I

| | |
|---|---|
| Initialized | The state of an Action Request when it is first created but has not yet been Activated (sent to a user's Action List). |
| Initiated | A workflow routing status indicating a document has been created but has not yet been saved or routed. A Document Number is automatically assigned by the system. |

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

# J

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

# K

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

| | |
|---|---|
| KNS – Kuali Nervous System | A core technical module composed of reusable code components that provide the common, underlying infrastructure code and functionality that any module may employ to perform its functions (for example, creating custom attributes, attaching electronic images, uploading data from desktop applications, lookup/ search routines, and database interaction). |
| KPP - Kuali Partners Program | The Kuali Partners Program (KPP) is the means for organizations to get involved in the Kuali software community and influence its future through voting rights to determine software development priorities. Membership dues pay staff to perform Quality Assurance (QA) work, release engineering, packaging, documentation, and other work to coordinate the timely enhancement and release of quality software and other services valuable to the members. Partners are also encouraged to tender functional, technical, support or administrative staff members to the Kuali Foundation for specific periods of time. |
| KRAD - Kuali Rapid Application Development | TODO |
| KRMS - Kuali Rules Management System | TODO |
| KS - Kuali Student | Delivers a means to support students and other users with a student-centric system that provides real-time, cost-effective, scalable support to help them identify and achieve their goals while simplifying or eliminating administrative tasks. The high-level entities of person (evolving roles-student, instructor, etc.), time (nested units of time - semesters, terms, classes), learning unit (assigned to any learning activity), learning result (grades, assessments, evaluations), learning plan (intentions, activities, major, degree), and learning resources (instructors, classrooms, equipment). The concierge function is a self-service information sharing system that aligns information with needs and tasks to accomplish goals. The support for integration of locally-developed processes provides flexibility for any institution's needs. |
| KSB – Kuali Service Bus | Provides an out-of-the-box service architecture and runtime environment for Kuali Applications. It is the cornerstone of the Service Oriented Architecture layer of the architectural reference framework. The Kuali Service Bus consists of:<br>• A services registry and repository for identifying and instantiating services<br>• Run time monitoring of messages<br>• Support for synchronous and asynchronous service and message paradigms |
| Kuali | 1. Pronounced "ku-wah-lee". A partnership organization that produces a suite of community-source, modular administrative software for Carnegie-class higher education institutions. See also *Kuali Foundation*<br>2. (n.) A humble kitchen wok that plays an important role in a successful kitchen. |
| Kuali Foundation | Employs staff to coordinate partner efforts and to manage and protect the Foundation's intellectual property. The Kuali Foundation manages a growing portfolio of enterprise software applications for colleges and universities. A lightweight Foundation staff coordinates the activities of Foundation members for critical software development and coordination activities such as source code control, release engineering, packaging, documentation, project management, |

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

# L

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

# M

| | |
|---|---|
| Maintenance Document | An e-doc used to establish and maintain a table record. |
| Message | The full description of a *notification message*. This is a specific field that can be filled out as part of the Simple Message or Event Message form. This can also be set by the programmatic interfaces when sending notifications from a client system. |
| Message Queue | Allows administrators to monitor messages that are flowing through the Service Bus. Messages can be edited, deleted or forwarded to other machines for processing from this screen. |

# N

| | |
|---|---|
| Namespace | A Namespace is a way to scope both *Permissions* and *Entity Attributes* Each Namespace instance is one level of scoping and is one record in the system. For example, "KRA" or "KC" or "KFS" could be a Namespace. Or you could further break those up into finer-grained Namespaces such that they would roughly correlate to functional modules within each application. Examples could be "KRA Rolodex", "KC Grants", "KFS Chart of Accounts".<br><br>Out of the box, the system is bootstrapped with numerous Rice namespaces which correspond to the different modules. There is also a default namespace of "KUALI".<br><br>Namespaces can be maintained at runtime through a maintenance document. |
| Note Text | A free-form text field for the text of a Note |
| Notification Content | This section of a *notification message* which displays the actual full message for the notification along with any other content-type-specific fields. |

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

# O

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

# P

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

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

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

Permissions are aggregated by *Roles*.

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

Attributes

1. Id - a system generated unique identifier that is the primary key for any Permission record in the system
2. Name - the name of the permission; also a human understandable unique identifier
3. Description - a full description of the purpose of the Permission record
4. Namespace - the reference to the associated *Namespace*

Relationships

1. Permission to *Role* - many to many; this relationship ties a Permission record to a Role that is authorized for the Permission
2. Permission to *Namespace* - many to one; this relationship allows for scoping of a Permission to a Namespace that contains functionality which keys its authorization checking off of said

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

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

# R

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

| | |
|---|---|
| Route Header Id | Another name for the *Document Id*. |
| Route Log | Displays information about the routing of a document. The Route Log is usually accessed from either the Action List or a Document Search. It displays general document information about the document and a detailed list of Actions Taken and pending *Action Requests* for the document. The Route Log can be considered an audit trail for a document. |
| Route Module | A routing component that the engine uses to generate action requests at a particular *Route Node*. *FlexRM* (Flexible Route Module) is a general Route Module that is rule-based. Clients can define their own Route Modules that can conduct specialized Routing based on routing tables or any other desired implementation. |
| Route Node | Represents a step in the routing process of a document type. Route node "instances" are created dynamically as a document goes through its routing process and can be defined to perform any function. The most common functions are to generate Action Requests or to split or join the route path.<br>• Simple: do some arbitrary work<br>• Requests: generate action requests using a Route Module or the Rules engine<br>• Split: split the route path into one or more parallel branches<br>• Join: join one or more branches back together<br>• Sub Process: execute another route path inline<br>• Dynamic: generate a dynamic route path |
| Route Path | The path a document follows during the routing process. Consists of a set of route nodes and branches. The route path is defined as part of the *document type* definition. |
| Route Status | The status of a document in the course of its routing:<br>• Approved: These documents have been approved by all required reviewers and are waiting additional postprocessing.<br>• 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.<br>• Disapproved: These documents have been disapproved by at least one reviewer. Routing has stopped for these documents.<br>• Enroute: Routing is in progress on these documents and an action request is waiting for someone to take action.<br>• 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.<br>• Final: All required approvals and all acknowledgements have been received on these documents. No changes are allowed to a document that is in Final status. |

- Initiated: A user or a process has created this document, but it has not yet been routed to anyone's Action List.
- Processed: These documents have been approved by all required users, and is completed on them. They may be waiting for Acknowledgements. No further action is needed on these documents.
- Saved: These documents have been saved for later work. An author (initiator) can save a document before routing begins or a reviewer can save a document before he or she takes action on it. When someone saves a document, the document goes on that person's Action List.

Routed By User

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

Routing

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

Routing Priority

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

Routing Rule

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

Technical considerations for a Routing Rules are:

- Configured via a GUI (or imported from XML)
- Created against a Rule Template and a Document Type
- The Rule Template and it's list of Rule Attributes define what fields will be collected in the Rule GUI
- Rules define the users, groups and/or roles who should receive action requests
- Available Action Request Types that Rules can route
  - Complete
  - Approve
  - Acknowledge
  - FYI
- During routing, Rule Evaluation Sets are "selected" at each node. Default is to select by Document Type and Rule Template defined on the Route Node

- Rules match (or 'fire') based on the evaluation of data on the document and data contained on the individual rule
- Examples
  - If dollar amount is greater than $10,000 then send an Approval request to Joe.
  - If department is "HR" request an Acknowledgment from the HR.Acknowledgers workgroup.

Rule Attribute

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

Technical considerations for a Rule Attribute are:

- They might be backed by a Java class to provide lookups and validations of appropriate values.
- Define how a Routing Rule evaluates document data to determine whether or not the rule data matches the document data.
- Define what data is collected on a rule.
- An attribute typically corresponds to one piece of data on a document (i.e dollar amount, department, organization, account, etc.).
- Can be written in Java or defined using XML (with matching done by XPath).
- Can have multiple GUI fields defined in a single attribute.

Rule QuickLinks

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

Rule Template

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

Technical considerations for a Rule Templates are:

- They are a composition of Rule Attributes
- Adding a 'Role' attribute to a template allows for the use of the Role on any rules created against the template
- When rule attributes are used for matching on rules, each attribute is associated with the other attributes on the template using an implicit 'and' logic attributes
- Can be used to define various other aspects to be used by the rule creation GUI such as rule data defaults (effective dates, ignore previous, available request types, etc)

# S

| | |
|---|---|
| Save | A workflow action button that allows the Initiator of a document to save their work and close the document. The document may be retrieved from the initiator's Action List for completion and routing at a later time. |
| Saved | A routing status indicating the document has been started but not yet completed or routed. The Save action allows the initiator of a document to save their work and close the document. The document may be retrieved from the initiator's action list for completion and routing at a later time. |
| Searchable Attributes | Attributes that can be defined to index certain pieces of data on a document so that it can be searched from the *Document Search screen*.<br><br>Technical considerations for a Searchable Attributes are:<br><br>• They are responsible for extracting and indexing document data for searching<br><br>• They allow for custom fields to be added to Document Search for documents of a particular type<br><br>• They are configured as an attribute of a Document Type<br><br>• They can be written in Java or defined in XML by using Xpath to facilitate matching |
| Secondary Delegation | The Secondary Delegate acts as a temporary backup Delegator who acts with the same authority as the primary Approver/the Delegator when the Delegator is not available. Documents appear in the Action Lists of both the Delegator and the Delegate. When either acts on the document, it disappears from both Action Lists.<br><br>Secondary Delegation is often configured for a range of dates and it must be registered in KEW or KIM to be in effect. |
| Service Registry | Displays a read-only view of all of the services that are exposed on the Service Bus and includes information about them (for example, IP Address, or Endpoint URL). |
| Simple Node | A type of node that can perform any function desired by the implementer. An example implementation of a simple node is the node that generates Action Requests from route modules. |
| SOA | An acronym for Service Oriented Architecture. |
| Special Condition Routing | This is a generic term for additional route levels that might be triggered by various attributes of a transaction. They can be based on the type of document, attributes of the accounts being used, or other attributes of the transaction. They often represent special administrative approvals that may be required. |
| Split Node | A node in the routing path that can split the route path into multiple branches. |
| Spring | The Spring Framework is an open source application framework for the Java platform. |
| State | Defines U.S. Postal Service codes used to identify states. |
| Status | On an Action List; also known as Route Status. The current location of the document in its routing path. |

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

# T

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

# U

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

# V

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

# W

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

| | |
|---|---|
| | that electronically routes an e-doc to its approvers in a prescribed sequence, according to established business rules based on the e-doc content. See also *Kuali Enterprise Workflow*. |
| Workflow Engine | The component of KEW that handles initiating and executing the route path of a document. |
| Workflow QuickLinks | A web interface that provides quick navigation to various functions in KEW. These include:<br>• Quick EDoc Watch: The last five Actions taken by this user. The user can select and repeat these actions.<br>• Quick EDoc Search: The last five EDocs searched for by this user. The user can select one and repeat that search.<br>• Quick Action List: The last five document types the user took action with. The user can select one and repeat that action. |

# X

| | |
|---|---|
| XML | See also *XML Ingester*.<br>1. An acronym for Extensible Markup Language.<br>2. Used for data import/export. |
| XML Ingester | A workflow function that allows you to browse for and upload XML data. |
| XML RuleAttribute | Similar in functionality to a RuleAttribute but built using XML only |