Creating a simple Maintenance Document

Overview

This document describes the steps to create a simple maintenance document. It is assumed that the rice project skeleton was created and is running successful. This means that the development environment and database are setup as well. Instructions to set up the development, database and create a project skeleton are found in the Rice Installation Guide. In depth explanations of the various configurations are found in the KRAD Guide.

The code in https://github.com/cniesen/ClausWorkshop/tree/step0 represents the project skeleton. The following steps are used to create a simple maintenance document. The final code is located at https://github.com/cniesen/Clausworkshop/tree/step1.

1. Remove sample classes and tests that were build with archetype
2. Create database table
3. Create persistable business object
4. Create OJB mapping
5. Create data dictionary
6. Create view
7. Configure Spring
8. Add link to maintenance document from portal page
9. Create and ingest document type

Removing sample classes and test that were build with archetype

When the archetype is being build an addition and multiplication services and their test are created as well. These exist only as a sample and can safely be removed.

To do so delete the following files:

- AdditionProductServiceImpl.java
- MultiplicationProductServiceImpl.java
- ProductService.java
- AbstractProductServiceImplTest.java
- AdditionProductServiceImplTest.java
- MultiplicationProductServiceImplTest.java

And finally remove the *productService* bean from the *BootStrapSpringBeans.xml* file.

Create database table

Start out by creating the database table that will store the data object of the maintenance document. The table will be named *cw_request_t* and contain *id* (which is a unique numeric identifier for the requests), *date* (when the request was made), *beneficiary* (the name of the beneficiary for this request), and *description* (that describes the request).

```
CREATE TABLE cw_request_t
(
    id BIGINT NOT NULL,
    date DATE NOT NULL,
    beneficiary VARCHAR(50) NOT NULL,
    description VARCHAR(250) NOT NULL,
    obj_id VARCHAR(36) NOT NULL,
    ver_nbr DECIMAL(8) NOT NULL DEFAULT 1,
    PRIMARY KEY(id),
    UNIQUE cw_request_tc0(obj_id)
);
```

The *id* will be using a sequence number. Since MySQL doesn't support native sequence numbers the following table is required as well:

```
CREATE TABLE cw_request_s
(
    id BIGINT NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id)
);
```

The sql file containing this code can be found at https://github.com/cniesen/ClausWorkshop/blob/step1/db/create_tables.sql.

Create persistable business object

The persistable bo class extends *PersistableBusinessObjectBase* and has a field with getters and setters for each table column. The *versionNumber* and *objectId* fields are defined in the extended *PersistableBusinessObjectBase*.

```
package com.niesens.clausworkshop;

import java.util.Date;
import org.kuali.rice.krad.bo.PersistableBusinessObjectBase;

public class Request extends PersistableBusinessObjectBase {

    private static final long serialVersionUID = 1326930685254961234L;

    private Long id;
    private Date date;
    private String beneficiary;
    private String description;

    public Long getId() {
            return id;
    }

    public void setId(Long id) {
            this.id = id;
    }

    public Date getDate() {
            return date;
    }

    public void setDate(Date date) {
            this.date = date;
    }

    public String getBeneficiary() {
            return beneficiary;
    }
```

```
    public void setBeneficiary(String beneficiary) {
            this.beneficiary = beneficiary;
    }

    public String getDescription() {
            return description;
    }

    public void setDescription(String description) {
            this.description = description;
    }

}
```

The Java file for this class can be found at https://github.com/cniesen/ClausWorkshop/blob/step1/src/main/java/com/niesens/clausworkshop/Request.java.

Create OJB mapping

The following OJB mapping will map the database table to the business object. The additional properties on the id mapping are for auto generating a unique id number.

```
<descriptor-repository version="1.0">

 <jdbc-connection-descriptor
        jcd-alias="dataSource-samplepp"
        default-connection="false"
        jdbc-level="3.0"
        eager-release="false"
        batch-mode="false"
        useAutoCommit="0"
        ignoreAutoCommitExceptions="false">
   <object-cache class="org.apache.ojb.broker.cache.ObjectCachePerBrokerImpl"/>
   <sequence-manager className=
        "org.kuali.rice.core.framework.persistence.ojb.ConfigurableSequenceManager">
     <attribute attribute-name="property.prefix"
                attribute-value="datasource.ojb.sequenceManager"/>
   </sequence-manager>
 </jdbc-connection-descriptor>


 <class-descriptor class="com.niesens.clausworkshop.Request" table="CW_REQUEST_T">
   <field-descriptor name="id" column="ID" jdbc-type="BIGINT" primarykey="true"
                     autoincrement="true" sequence-name="CW_REQUEST_S"/>
   <field-descriptor name="date" column="DATE" jdbc-type="DATE"/>
   <field-descriptor name="beneficiary" column="BENEFICIARY" jdbc-type="VARCHAR"/>
   <field-descriptor name="description" column="DESCRIPTION" jdbc-type="VARCHAR"/>


   <field-descriptor name="versionNumber" column="VER_NBR" jdbc-type="BIGINT"
                     locking="true"/>
   <field-descriptor name="objectId" column="OBJ_ID" jdbc-type="VARCHAR" indexed="true"/>
 </class-descriptor>
</descriptor-repository>
```

The OJB mapping can be found at https://github.com/cniesen/ClausWorkshop/blob/step1/src/main/resources/com/niesens/clausworkshop/OJB-repository.xml.

Create data dictionary

The data dictionary defines the content of the maintenance document with the *DataObjectEntry* bean. The *dataObjectClass* property specifies that this definition is for the Request object. The *attributes* property list tells Rice what fields the Request object has. For clarity the detailed field definitions are broken out into their own beans. The field specified in the *titleAttribute* property will be rendered with an inquiry link in the lookup results.

Each field is defined through an *AttributeDefinition* bean. The *name* property specifies the field in the BO. The *label* and *shortLabel* properties contains the text that is displayed on the user interface. The *maxLength* property specifies the maximum length of the field for data entry and display. The *controlField* property specifies the input control type. This can be a simple *Uif-TextControl* for text input or fancy controls like the *Uif-DateControl* that includes a date picker widget. With the *validCharactersConstraint* property input constraints can be specified.

The maintenance document itself is defined via the *uifMaintenanceDocumentEntry* bean in the data dictionary. The *dataObjectClass* property again specifies that is definition is for the Request object. The *lockingKeys* property list contains usually the primary keys to prevent that duplicates of the maintenance document are added to the workflows. The *documentTypeName* property ties the maintenance document to the document type that is created and ingested into the workflow later.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                          http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">


   <bean id="Request" parent="Request-parentBean"/>
   <bean id="Request-parentBean" abstract="true" parent="DataObjectEntry">
       <property name="dataObjectClass" value="com.niesens.clausworkshop.Request"/>
       <property name="objectLabel" value="Request"/>
       <property name="attributes">
           <list>
               <ref bean="Request-id"/>
               <ref bean="Request-date"/>
               <ref bean="Request-beneficiary"/>
               <ref bean="Request-description"/>
           </list>
       </property>
       <property name="titleAttribute" value="date"/>
   </bean>


   <!-- Attribute Definitions -->

   <bean id="Request-id" parent="Request-id-parentBean"/>
   <bean id="Request-id-parentBean" abstract="true" parent="AttributeDefinition">
       <property name="name" value="id"/>
       <property name="label" value="Request ID"/>
       <property name="shortLabel" value="ID"/>
       <property name="maxLength" value="14"/>
       <property name="validCharactersConstraint">
           <bean parent="NumericPatternConstraint"/>
       </property>
```

```
        <property name="controlField">
            <bean parent="Uif-SmallTextControl"/>
        </property>
    </bean>


    <bean id="Request-date" parent="Request-date-parentBean"/>
    <bean id="Request-date-parentBean" abstract="true" parent="AttributeDefinition">
        <property name="name" value="date"/>
        <property name="label" value="Request Date"/>
        <property name="shortLabel" value="Date"/>
        <property name="controlField">
            <bean parent="Uif-DateControl"/>
        </property>
        <property name="required" value="true"/>
    </bean>


    <bean id="Request-beneficiary" parent="Request-beneficiary-parentBean"/>
    <bean id="Request-beneficiary-parentBean" abstract="true"
          parent="AttributeDefinition">
        <property name="name" value="beneficiary"/>
        <property name="label" value="Beneficiary"/>
        <property name="shortLabel" value="Beneficiary"/>
        <property name="maxLength" value="100"/>
        <property name="validCharactersConstraint">
            <bean parent="AnyCharacterPatternConstraint" p:allowWhitespace="true"/>
        </property>
        <property name="controlField">
            <bean parent="Uif-TextControl" p:size="50"/>
        </property>
        <property name="required" value="true"/>
    </bean>


    <bean id="Request-description" parent="Request-description-parentBean"/>
    <bean id="Request-description-parentBean" abstract="true"
          parent="AttributeDefinition">
        <property name="name" value="description"/>
        <property name="label" value="Request Description"/>
        <property name="shortLabel" value="Description"/>
        <property name="maxLength" value="250"/>
        <property name="controlField">
            <bean parent="Uif-MediumTextAreaControl"/>
        </property>
        <property name="required" value="true"/>
    </bean>

    <!-- Maintenance Document Entry -->
    <bean id="RequestMaintenanceDocument" parent="RequestMaintenanceDocument-parentBean"/>
    <bean id="RequestMaintenanceDocument-parentBean" abstract="true"
          parent="uifMaintenanceDocumentEntry">
        <property name="dataObjectClass" value="com.niesens.clausworkshop.Request"/>
        <property name="lockingKeys">
          <list>
            <value>id</value>
          </list>
        </property>
        <property name="documentTypeName" value="PeopleFlowMaintenanceDocument"/>
    </bean>

</beans>
```

The data dictionary can be found at https://github.com/cniesen/ClausWorkshop/blob/step1/src/main/resources/com/niesens/clausworkshop/datadictionary/Request.xml.

Create view

While the data dictionary defines defaults for the fields of the business object, the user interface view defines the final layout of lookup, inquiry and maintenance.

With all three views, the *dataObjectClassName* ties the view to the business object and the *headerText* specifies what is being displayed as the title of the view on the user interface.

With the inquiry view, the *items* property list specifies the components that are displayed on the page. Usually this one of the various section components that displays the field values of the business object using the *Uif-DataField* component. But any components can be used on the inquiry view and may be built into a complex view if desired.

Lookups are usually kept pretty standard. The lookup criteria are specified in the *criteriaFields* property list using the *Uif-LookupCriteriaInputField* component. The lookup result is specified in the *resultFields* property list using the *Uif-DataField* component.

The maintenance view is often a bit more complex, with content split over multiple beans for readability. Like the inquiry view, the *items* property list generally has one or more sections defined. This example is quite simple and uses only the *Uif-DataField* component for read only fields and the *Uif-InputField* component for editable fields.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                          http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">


   <!-- Inquiry View -->

   <bean id="Request-InquiryView" parent="Uif-InquiryView">
       <property name="headerText" value="Request Inquiry"/>
       <property name="dataObjectClassName" value="com.niesens.clausworkshop.Request"/>
       <property name="items">
           <list>
               <bean parent="Uif-Disclosure-GridSection">
                   <property name="headerText" value="Request Details"/>
                   <property name="layoutManager.numberOfColumns" value="2"/>
                   <property name="items">
                       <list>
                           <bean parent="Uif-DataField" p:propertyName="id"/>
                           <bean parent="Uif-DataField" p:propertyName="date"/>
                           <bean parent="Uif-DataField" p:propertyName="beneficiary"/>
                           <bean parent="Uif-DataField" p:propertyName="description"/>
                       </list>
                   </property>
               </bean>
           </list>
       </property>
   </bean>
```

```
    <!--  Lookup View -->

    <bean id="Request-LookupView" parent="Uif-LookupView">
        <property name="headerText" value="Request Lookup"/>
        <property name="dataObjectClassName" value="com.niesens.clausworkshop.Request"/>
        <property name="criteriaFields">
            <list>
                <bean parent="Uif-LookupCriteriaInputField" p:propertyName="beneficiary"/>
                <bean parent="Uif-LookupCriteriaInputField" p:propertyName="date"/>
            </list>
        </property>
        <property name="resultFields">
            <list>
                <bean parent="Uif-DataField" p:propertyName="id"/>
                <bean parent="Uif-DataField" p:propertyName="date"/>
                <bean parent="Uif-DataField" p:propertyName="beneficiary"/>
            </list>
        </property>
    </bean>

    <!-- Maintenance View -->

    <bean id="Request-MaintenanceView" parent="Uif-MaintenanceView">
        <property name="headerText" value="Request Maintenance"/>
        <property name="dataObjectClassName" value="com.niesens.clausworkshop.Request"/>
        <property name="items">
          <list merge="true">
            <bean parent="Request-Details"/>
          </list>
        </property>
    </bean>

    <bean id="Request-Details" parent="Uif-Disclosure-GridSection">
        <property name="headerText" value="PeopleFlow Summary"/>
        <property name="items">
            <list>
                <bean parent="Uif-DataField" p:propertyName="id"
                      p:render="@{#dp.id != null}"/>
                <bean parent="Uif-InputField" p:propertyName="date" />
                <bean parent="Uif-InputField" p:propertyName="beneficiary"/>
                <bean parent="Uif-InputField" p:propertyName="description"/>
            </list>
        </property>
    </bean>
</beans>
```

The views can be found at https://github.com/cniesen/ClausWorkshop/blob/step1/src/main/resources/com/niesens/clausworkshop/datadictionary/RequestView.xml.

Configure Spring

Spring is used to tie the OJB mapping, the data dictionary and user interface view to the application. For clarity the application specific spring configurations are moved into their own file. To do so the following is added to the *BootStrapSpringBeans.xml* file:

```
<import resource="classpath:com/niesens/clausworkshop/ClausWorkshopModuleBeans.xml"/>
```

And the associated *ClausWorkshopModuleBeans.xml* file with the following content is created:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                      http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

    <bean id="ClausWorkshopModuleConfiguration"
          class="org.kuali.rice.krad.bo.ModuleConfiguration">
        <property name="namespaceCode" value="CW"/>
        <property name="initializeDataDictionary" value="true"/>
        <property name="dataDictionaryPackages">
            <list>
                <value>com/niesens/clausworkshop/datadictionary</value>
            </list>
        </property>
        <property name="databaseRepositoryFilePaths">
            <list>
                <value>com/niesens/clausworkshop/OJB-repository.xml</value>
            </list>
        </property>
        <property name="packagePrefixes">
            <list>
                <value>com.niesens.clausworkshop</value>
            </list>
        </property>
    </bean>

    <bean id="ClausWorkshopModuleService"
          class="org.kuali.rice.krad.service.impl.ModuleServiceBase">
        <property name="moduleConfiguration" ref="ClausWorkshopModuleConfiguration"/>
    </bean>

</beans>
```

The module is given its own namespace via the *namespaceCode* property. This allows module specific permission, workflow and other configuration.

The *dataDictionaryPackages* property list specifies the location of the data dictionary files. This can be either directories or individual files.

Finally the module service must be created for the module configuration.

Add link to maintenance document from portal page

The *customApplication.tag* is renamed to *clausWorkshop.tag* to make it more appropriate within the context of the application. The reference to it in *mainTab.tag* needs to be updated as well.

Now in the *clausWorkshop.tag* file the

```
<li>insert custom content here</li>
```

is being replaces with the following link that includes the view ID of the lookup:

```
<li><portal:portalLink displayTitle="true" title="Request"
url="${ConfigProperties.krad.url}/lookup?methodToCall=start&viewId=Request-
LookupView&showMaintenanceLinks=true&returnLocation=${ConfigProperties.application.url}/p
ortal.do&hideReturnLink=true" /></li>
```

Just to make things prettier, change the channel heading in the *web.xml* file from "clausworkshop" to "Claus' Workshop".

Create and ingest document type

The document type needs to be defined in KEW in order to create and edit the maintenance document.

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

<data xmlns="ns:workflow" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="ns:workflow resource:WorkflowData">

    <documentTypes xmlns="ns:workflow/DocumentType"
                   xsi:schemaLocation="ns:workflow/DocumentType resource:DocumentType">

        <documentType>
            <name>RequestMaintenanceDocument</name>
            <parent>RiceDocument</parent>
            <description>Create/Edit a Request</description>
            <label>Request</label>
            <docHandler>${kr.krad.url}/maintenance?methodToCall=docHandler</docHandler>
            <active>true</active>
            <routingVersion>2</routingVersion>
        </documentType>
    </documentTypes>

</data>
```

This document type contains only the basic information without any routing. Following is a short description of each attribute:

<*name*> unique name that identifies the document type
<*parent*> document type from which to inherit attributes
<*description*> describes the primary responsibilities of the document type
<*label*> UI name of the document type
<*postProcessorName*> postprocessor completes the business transaction; is notified on node transition, status change or action taken; specific to a set of document types
<*docHandler*> how the document is displayed
<*routingVersion*> for backwards compatibility

The document type is loaded into the database using the XML Ingester. To do so, start up the application, navigate to the *Administration* panel (tab), follow the *XML Ingester* link, choose the xml file, and click the *upload xml data* button.

The document type xml can be found at https://github.com/cniesen/ClausWorkshop/blob/step1/db/RequestMaintenanceDocument.xml.