Global Maintenance Documents 2

The maintenance framework lets us create, update, or inactivate a single business object at a time. Sometimes, however, there's a need for a mass functional update, where tons of business objects are created, updated, or deactivated in a single go. Let's say, for instance, that we wanted to change the name of the object code 5000 to "Expensive Stuff" for the fiscal years 2008 - 2010 and all chart codes. That could mean a lot of maintenance documents if we changed the object codes one by one. However, there is a an easier way to accomplish this; for this specific purpose, there's the "Global Maintenance Document" framework.

It currently exists for a whole five business object classes: Accounts, Object Codes, Account Delegates, Sub-Object Codes, and Organization Reversions. While largely similar to most other maintenance documents, there are a couple of differences to look at: global business objects must implement the GlobalBusinessObject method, they all have one or more collections of "details" (this is what makes them batch updates), and they must define a custom Maintainable.

Implementing org.kuali.core.bo.GlobalBusinessObject

A global business object is, much like a transactional document class, a business object - it has references that can be refreshed, it can be persisted to a datastore - but it also shares qualities with documents. Unlike a regular maintenance document, which wraps a business object in the MaintenanceDocument class, the global business object is itself a document. It must declare a documentNumber, and it tables for it and any associated "detail" objects must be created in the database along with OJB mappings. OJB-repository-chart.xml has the OJB mapping for org.kuali.module.chart.bo.AccountGlobal, and that provides an excellent example of what a table mapping for a global business object will look like.

A global business object can have properties with getters and setters. These properties typically represent the details that will be mass applied to every business object updated or created by the global operation. org.kuali.module.chart.bo.AccountGlobal, for instance, has several properties: organization code, sub-fund group code, and account city name to list just three. org.kuali.module.chart.bo.DelegateGlobal doesn't really have any properties like that (it has tricky properties like "model name" that...we...we just won't cover those here).

By convention, all global business objects are named {*regular business object name*}Global, ie: AccountGlobal, OrganizationReversionGlobal, ObjectCodeGlobal, and so on. All tables that hold global business objects end in "_CHG_T" and all detail tables have names that end in "_CHG_DTL_T".

Global business objects must implement the org.kuali.core.bo.GlobalBusinessObject interface. Let's take a tour of the methods we must implement:

- getDocumentNumber/setDocumentNumber: again, remember that the global business object is really a document. It needs a documentNumber so that workflow can track it through routing. The documentNumber will typically be the primary key of the global business object. Because of all of this, a getter and a setter for documentNumber are required for all global business objects.
- isPersistable: this tells the framework that the global maintenance document is in a state where it could persisted to the database or not. For instance, if some details within the document cannot be saved to the document, then isPersistable() should return false.
- getAllDetailObjects: This returns all of the detail objects associated with the global document. Even if there's multiple collections of detail objects, like there is for DelegateGlobal, all of the details from all collections need to be returned by this method.
- generateGlobalChangesToPersist and generateGlobalDeactivationsToPersist: Finally, we come to the methods that do the real work for Global Maintenance Documents. The purpose of global maintenance documents is to affect changes - insertions, updates, and deactivations - for many business objects. And these methods return Lists of PersistableBusinessObjects to perform these actions on. generateGlobalChangesToPersist() returns business objects that will be saved to the persistence store - so, any business object that should either be inserted or updated based on the global maintenance document should be returned here. generateGlobalDeactivationsToPersist(), on the other hand, is the reaper; any business objects returned by this method will be deactivated. We take a closer look at how to implement this in the section on generateGlobalChangesToPersist().

Detail collections

Every global maintenance document has at least one "detail collection" and sometimes more than one. What is a detail collection?

Again, since the changes caused by a global maintenance document are supposed to be applied to many business objects, the detail collections often represent part of the primary keys of the business objects to update. For instance, in ObjectCodeGlobal, the single detail collection represents financial years and charts. The object code business object has three fields that make up its primary key - fiscal year, chart, and object code. Therefore, by having a collection of various fiscal years and charts, we can vary those to find multiple object codes, for instance by adding details for fiscal year 2008 for every chart - in the demo data, that would be 14 object code details, which will update 14 object code business objects.

How do we create a business object detail? The requirements are pretty light.

First, we have to create a class that extends org.kuali.core.bo.GlobalBusinessObjectDetailBase. This class doesn't have much - just a document number, which acts as a foreign key to the global business object. The detail class can then have whatever methods it needs, typically getter/setter properties for the attributes that the user needs to enter. We also must create a database table to hold records of each type of detail

business object and we have to set up the OJB mappings for the detail table.

Secondly, the global business object must have a getter and setter pair to operate on a property, which is the List of getter and setter objects. For instance, GlobalObjectCode has two methods, getObjectCodeGlobalDetails() and setObjectCodeGlobalDetails() which return or set a List of ObjectCodeGlobalDetail objects.

Finally, in the global's data dictionary, the collection section of the details needs to be named after the property created in the last step. So, for instance, the property name of ObjectCodeGlobal's detail collection is "objectCodeGlobalDetails", and therefore, in the ObjectCodeGlobalMaintenanceDocument.xml data dictionary configuration file, we see this code:

```
<maintainableCollection name="objectCodeGlobalDetails"
 businessObjectClass="org.kuali.module.chart.bo.ObjectCodeGlobalDetail"
 sourceClassName="org.kuali.module.chart.bo.Chart"
sourceAttributeName="objectCodeGlobalDetails"
 summaryTitle="Year and Chart">
```

The collection is named after the property, and it uses the detail class as its business object class.

Extending org.kuali.core.maintenance.KualiGlobalMaintainableImpl

Global maintenance documents have a special maintainable implementation to extend, called KualiGlobalMaintainableImpl. It has a lot of utility methods for the maintenance of the global object, and we'll take a look at those in a second. However, it has one abstract method, which requires that we create an implementation for it: generateMaintenanceLocks(). Just as for non-global maintenance documents, this method creates a String representation of a business object to lock, so that no other user can edit that business object while the maintenance document is in workflow. While KualiMaintainableImpl can use an algorithm to generate those locks correctly most of the time, there's no way to do that for the global maintenance document, because the global maintenance document needs to generate a lock for every single business object that could be updated, inserted, or deactivated by the document. There, this method must be created. For examples of use, check out the existing global maintenance documents; for instance, org.kuali.module.chart.maintenance.ObjectCodeGlobalMaintainableImpl has a very straightforward lock generation algorithm that can be adapted to the needs of different global maintenance documents.

KualiGlobalMaintainableImpl extends KualiMaintainableImpl, so all of the useful methods from Maintainable are there. Furthermore, two extra methods have been added which can be overridden if necessary:

- processGlobalsAfterRetrieve - called basically in place of KualiMaintainableImpl's processAfterRetrieve(), this is called after a global business object is retrieved from the persistence store. For an example, this is used by org.kuali.module.chart.maintenance.OrganizationReversionGlobalMaintainableImpl to prevent users from being able to add new organization reversion category records to the organization reversion.
- processGlobalsForSave - called before a global business object is saved. An example of this can be seen in org.kuali.module.chart.maintenance.ObjectCodeGlobalMaintainableImpl, where the object code details are all populated with the object code field from the ObjectCodeGlobal.

A second look at generateGlobalChangesToPersist()

Because it's so important, let's take a look at a generateGlobalChangesToPersist() implementation, specifically the one from org.kuali.module.chart.bo.ObjectCodeGlobal. ObjectCodeGlobal has a collection of fiscal year and charts which is the "detail collection" of the global maintenance document, and then has a list of attributes to change. So, for instance, let's say that we carry out the operation we described in the introduction, that we change the name of object code 5000 for all charts in 2008 to "Expensive Stuff". To do this, we'd fill out the object code ("5000") and fill in the object code name ("Expensive Stuff"), and then we'd add a "detail" record for each of the charts in 2008.

Then, what does the generateGlobalChangesToPersist() do to generate the changes? Here's the code:

```
1.  public List<PersistableBusinessObject> generateGlobalChangesToPersist() {
2.    List result = new ArrayList();

3.    // Iterate through Object Codes; create new or update as necessary
      // Set reports-to Chart to appropriate value

4.    for (ObjectCodeGlobalDetail detail : objectCodeGlobalDetails) {
5.      Map pk = new HashMap();

6.      Integer fiscalYear = detail.getUniversityFiscalYear();
7.      String chart = detail.getChartOfAccountsCode();

8.      if (fiscalYear != null && chart != null && chart.length() > 0) {
9.        pk.put("UNIV_FISCAL_YR", fiscalYear);
10.       pk.put("FIN_COA_CD", chart);
11.       pk.put("FIN_OBJECT_CD", financialObjectCode);

12.       ObjectCode objectCode = (ObjectCode)
SpringContext.getBean(BusinessObjectService.class).findByPrimaryKey(ObjectCode.class,
pk);
13.       if (objectCode == null) {
14.         objectCode = new ObjectCode(fiscalYear, chart, financialObjectCode);
15.         objectCode.setFinancialObjectActiveCode(true);
16.       }
17.       populate(objectCode, detail);
18.       Map<String, String> hierarchy =
SpringContext.getBean(ChartService.class).getReportsToHierarchy();
19.       objectCode.setReportsToChartOfAccountsCode(hierarchy.get(chart));

20.       result.add(objectCode);
21.     }
22.   }

23.   return result;
24. }
```

Let's take a look at what this code does. On line 2, we create the result List to hold all of the updated object codes in. Then in line 4, we start looping through each of the object code details - again, financial years and chart codes. For each financial year and chart code, an object code is looked up (lines 6 - 12). If the object code is null, a new object code is created (lines 13 - 16).

On line 17, the private method "populate" is applied to the object code; this merely puts the relevant details into the new or updated object code. Then on lines 18-19, it finds the correct reports to hierarchy chart for the object code and then puts the new or updated object code into the list.

The maintenance framework does the rest - once the document gets to a final state in workflow, all of the object codes are saved. As we can see, the global maintenance framework gives us a lot of power and it's fairly simple.

The trail ends here. Go back to the beginning, or leave your browser parked here and bask in the wisdom.

Unable to render {include} The included page could not be found.