Multiple Value Lookups 2

What is a multiple value lookup?

A multiple value lookup is a special form of the normal lookups. In normal lookups, depending on how the lookup was invoked (i.e. from the portal or a document), the user typically has the option of returning a record to the calling page or to copy/edit that record. In multiple value lookups, each result row is displayed with a check box next to it. When the user clicks on the "return selected" button, all of the checked records will be returned to the calling page.

Multiple value lookups for maintenance documents

The maintenance framework provides a significant amount of support when returning from multiple value lookups. Normally, no java code needs to be written to return values.

`maintainableCollection` definitions need to have several attributes defined that will provide the MV lookups with the information necessary to perform all processing.

```
<!-- note: the field names do not correspond to real fields in BO and have been
modified to make the example clearer -->
<maintainableCollection name="accountChangeDetails"
businessObjectClass="org.kuali.module.chart.bo.AccountChangeDetail"
sourceClassName="org.kuali.module.chart.bo.Account"
sourceAttributeName="accountChangeDetails">
    <maintainableField name="changeChartOfAccountsCode" template="chartOfAccountsCode"
required="true" readOnlyAfterAdd="true" />
    <maintainableField name="changeAccountNumber" template="accountNumber"
webUILeaveFieldFunction="loadAccountName" required="true" readOnlyAfterAdd="true" />
    <maintainableField name="changeAccountName" template="accountName" readOnly="true"
/>
    <maintainableField name="versionNumber" />
</maintainableCollection>
```

`<maintainableCollection>`'s need to have several attributes defined:

- `sourceClassName` : the type of BOs that will be queried. It is not always the same type of BO which are elements in the maintainable collection. In the example, we are maintaining a collection of `AccountChangeDetail` BOs, and when we perform a MV lookup, we are using the values in the `Account` BO (the source BO) to fill in values in the `AccountChangeDetail` BO.
- `sourceAttributeName` : for MV lookups to work, this value must not be missing/empty.

`<maintainableField>`'s may have one attribute defined:

- `template` : designates the name of the attribute from which the value of the source BO will be copied into the maintained collection element. In the example, `AccountChangeDelegate`'s `changeChartOfAccountsCode` will be copied from `Account`'s `chartOfAccountsCode`, `changeAccountNumber` is copied from `accountNumber`, and `changeAccountName` is copied from `accountName`. Note that not all field definitions need to have a template; these fields are simply left blank upon returning from a multiple value lookup.

Multiple value lookups for transactional/other documents

This type of documents is less structured. Therefore, more steps are required to return values from a multiple value lookup.

Rendering a multiple value lookup icon

Rendering a lookup tag is simple: simply invoke the multipleValueLookup.tag from the JSP page.

The following are the MV lookup specific attributes for this tag:

- boClassName (required): the class name of the business object being looked up
- lookedUpBODisplayName (not required): this value is the human readable name of the BO being looked up (e.g. Object Code)
- lookedUpCollectionName (required): the name of the collection being looked up (perhaps on a document collection), this value will be

returned to the calling document.

Struts form support

To return from a MV lookup, forms for the return struts action must/should support the retrieval of the following parameters:

- refreshCaller (should support): a value of "multipleValues" indicates that the MV lookups were used
- lookedUpCollectionName (should support): the name of the collection being looked up. It should be the property name of the collection, relative to the document object.
- lookupResultsSequenceNumber (must support): if not blank, this represents the serial number of the lookup. If blank, may represent that the user cancelled lookup, or returned with no value
- lookupResultsBOClassName (must support): represents the class that was being looked up. This is not necessarily the class of the collection elements. This corresponds to the "boClassName" value passed into the multipleValueLookup.tag tag.

If the latter two variables are non-blank, then it signifies that the user attempted to return results

Nervous system support

1. After retrieving the lookupResultsSequenceNumber and the lookupResultsBOClassName and they are non-blank, find the class corresponding to lookupResultsBOClassName (using `Class.forName`)
2. Get the lookupResultsService Spring bean, and invoke something similar to the following

```
KNSServiceLocator.getLookupResultsService().retrieveSelectedResultBOs(lookupResultsSeq
uenceNumber, lookupResultsBOClass,

GlobalVariables.getUserSession().getUniversalUser().getPersonUniversalIdentifier());
```

This method returns a java.util.Collection of the BOs selected.

Passing the universal user id ensures that only the user that initiated the lookup may retrieve those results

Add returned result BOs into a document collection

The section above described how to retrieve a collection of the BOs being looked up. Often, in many documents that invoke multiple value lookups, the selected rows are returned as some additional elements in some (sub)collection defined on the document. The class type of the looked up BO does not match with the collection element type. For example, for the account global documents, a multiple value lookup is available to lookup multiple `Account` objects, but the collection to which these results are added has elements of type `AccountChangeDetail`.

The KNS provides support to facilitate the addition of rows into the collection.

Example code

```
// retrieve the list of BOs
Collection<PersistableBusinessObject> rawValues =
KNSServiceLocator.getLookupResultsService()
        .retrieveSelectedResultBOs(lookupResultsSequenceNumber, lookupResultsBOClass,
GlobalVariables.getUserSession().getUniversalUser().getPersonUniversalIdentifier());

// request is the http servlet request
String lookedUpCollectionName =
request.getParameter(RiceConstants.LOOKED_UP_COLLECTION_NAME);
// the collection from the document, and we're assuming that it's of type
java.util.Collection
Collection documentCollection = (Collection) ObjectUtils.getPropertyValue(document,
lookedUpCollectionName);

// the type of elements from the documentCollection variable above
Class collectionElementClass = ...;

if (rawValues != null) {
    // see below for an explanation of this Map
    Map<String, String> template = getTemplate();
    for (PersistableBusinessObject nextBo : rawValues) {
        PersistableBusinessObject templatedBo = (PersistableBusinessObject)
ObjectUtils.createHybridBusinessObject(collectionClass, nextBo, template);
        // Do other necessary processing before adding to the collection, including
setting fields untouched by ObjectUtils.createHybridBusinessObject
        documentCollection.add(templatedBo);
    }
}
```

In the above example, the `template` map is used to indicate how fields from the looked up BO (e.g. `Account`) are mapped into the collection element BO (e.g. `AccountChangeDetail`). For each key-value mapping in the map, the key is a field name in the collection BO, and the corresponding value is a field name in the looked up BO.

In the method call `ObjectUtils.createHybridBusinessObject(collectionClass, nextBo, template)` above:

1. A new BO of type `collectionClass` is constructed.
2. For each key-value mapping of `template`, the field named with the key of the new instance created in (1) is assigned with the value of the field in `nextBo` named with the mapping's value.
3. The instance created in (1) is returned.

To see how the maint doc framework implements adding elements to a collection, look at `KualiMaintenanceDocumentAction.refresh(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)`