Multi Select Implementation Guide

Kevin McMillen

Orchestra Networks Professional Services

Contents

[1 Overview 1](#_Toc386628726)

[2 Selection Service 1](#_Toc386628727)

[3 Creation Service 2](#_Toc386628728)

[4 Creation Service Permission 2](#_Toc386628729)

[5 UIBeanEditor 3](#_Toc386628730)

[6 Access Rule 3](#_Toc386628731)

[7 Schema Extensions 4](#_Toc386628732)

[8 Table Filter 5](#_Toc386628733)

# Overview

Multi Select is a pattern that allows selection of multiple records for use in creating records in a table that joins that table to another. The “join table” must be linked to from a selection node on a “parent table”, and itself contains a foreign key to a “selection table”. For example, a Menu Item (parent table) record has a selection node to Menu Item Location (join table) records, which each have a foreign key to a Country (selection table) record.

A button on the selection node displays the selection table in a popup. The user can select multiple records and click on a Creation service from the Services menu. This will create the records in the join table and remove those records from the selection table view. When done, the user can click on the X in the top right corner to close the popup.

The join table must have a primary key that consists of a foreign key to the parent table and a foreign key to the selection table. It can have more fields in its primary key, but those would need to have default values specified either in the model or in its trigger.

# Selection Service

The UI that displays the table for the user to select from is handled by an HttpServlet class named **com.orchestranetworks.ps.multiselect. MultiSelectRecordSelectionService**. It gets invoked via a jsp that is registered as a Service on the table where the records will be created (the join table). (**WorkflowMultiSelectRecordSelection.jsp** invokes both this service and handles the workflow permissions pattern via the tracking info. However, if the workflow permissions pattern isn’t needed, one could write another jsp that ignores that part.)

The Selection service should be configured as follows:

* Service name: Something unique. (e.g. “MultiSelectCountrySelection”)
* Label: Something user-friendly, like “Select X”. (e.g. “Select Countries”)
* Web resource: The jsp. (e.g. “/services/WorkflowMultiSelectRecordSelection.jsp”)
* Activation paths: Associate it with the join table. (e.g. “/root/menuItemDetails/MenuItemLocation”)
* Permissions: Not defined.
* Display?: No. *(This service will be invoked by a UIBeanEditor and therefore can be hidden from all users.)*
* Order: Leave this blank.
* Confirm before launch: No.

# Creation Service

The records in the join table will be created via a HttpServlet class named **com.orchestranetworks.ps.multiselect.MultiSelectRecordCreationService**. It gets invoked via a jsp that is registered as a Service on the selection table. (**WorkflowMultiSelectRecordCreation.jsp** invokes both this service and handles the workflow permissions pattern via the tracking info. However, if the workflow permissions pattern isn’t needed, one could write another jsp that ignores that part.)

The Creation service should be configured as follows:

* Service name: Something unique. (e.g. “MultiSelectMenuItemLocationCreation”)
* Label: Something user-friendly, like “Create X”. (e.g. “Create Menu Item Locations”)
* Web resource: The jsp. (e.g. “/services/WorkflowMultiSelectRecordCreation.jsp”)
* Activation paths: Associate it with the table being selected from, requiring at least one selected record. (e.g. “/root/Country{+}”)
* Permissions: Configure the Creation Service Permission, as described [below](#_Creation_Service_Permission).
* Display?: Yes.
* Order: Leave this blank.
* Confirm before launch: Configure how you wish.

# Creation Service Permission

The permission on the Creation Service is handled by a class named **com.orchestranetworks.ps.multiselect.MultiSelectRecordCreationServicePermission**. These parameters should be filled in when adding it to the service:

* **separator** *(optional)*: The separator used by the session’s tracking info. Only needs to be specified if you’re not using the default separator. (e.g. “::”)
* **serviceTrackingInfoPosition**: Some information about the service needs to be inserted into the session’s tracking info. Since other patterns can utilize tracking info as well, this tells it what position that the service begins with. For example, if using the workflow permissions pattern, the first position is the username for the workflow permissions so this service begins at position 1. If no other pattern that utilizes the tracking info is being used, you can not specify this and it will use the default, which is 0. (e.g. “1”)
* **selectionServiceName**: The name of the Selection Service that displayed the selection table. Technically, this does not need to match the Selection Service name but it needs to be unique and using the Selection Service name is a good practice. It is inserted into the session’s tracking info to identify when the user is in the context of the Selection Service. (e.g. “MultiSelectCountrySelection”)

# UIBeanEditor

There must be a selection node to the join table. A custom UIBeanEditor named **com.orchestranetworks.ps.multiselect.MultiSelectUIBeanEditor** is placed on this field that will launch the multi select service. These parameters should be filled in when adding it to the field:

* **selectionServiceName**: The name of the Multi Select service to invoke. (e.g. “MultiSelectCountrySelection”)
* **joinTableParentPKPosition** *(optional)*: The join table must have a foreign key back to the parent table as part of its primary key. By default, it will be assumed to be the first part of the primary key (0) but if it is not, it can be specified here what position in the primary key it is. (e.g. “1”)
* **joinTableFKPath**: This is the path in the join table of the foreign key to the selection table. (e.g. “/country”)
* **selectLabel**: This is the label to show on the screen. (e.g. “Select Countries”)

The UIBeanEditor will launch the selection service in a popup and pass the joinTableParentPKPosition and joinTableFKPath to it as parameters, in addition to the current record’s data space, data set, table path, and primary key.

# Access Rule

Any records that are already linked to from the join table should not be shown in the Selection Service. Also, once a record has been created via the Creation Service, the record that was selected should not be shown. This is accomplished by a class called **com.orchestranetworks.ps.multiselect.MultiSelectAccessRule**. It needs to be associated with the selection table via the SchemaExtensionsContext.setAccessRuleOnOccurrence method. Only one Access Rule can be put on a table occurrence, so if you already have an Access Rule, you will need to extend MultiSelectAccessRule, or integrate its logic into your class, or utilize the AccessRulesManager that was developed by the Orchestra Networks Professional Services team. These parameters need to be configured when creating the Access Rule:

* **serviceID**: The name of the Selection Service. Technically, this does not need to match the Selection Service name but it needs to match the selectionServiceName parameter value of the Creation Service and using the Selection Service name is a good practice. (e.g. “MultiSelectCountrySelection”)
* **baseTrackingInfoHelper** *(optional)*: The Multi Select pattern utilizes a class called **com.orchestranetworks.ps.util.TrackingInfoHelper** developed by Orchestra Networks Professional Services to manage the session’s tracking info. If you need the Multi Select to work with another pattern which also utilizes the tracking info, then you need to specify the “base” TrackingInfoHelper that the other pattern uses so that the Multi Select can start its tracking info segments after the last of the base tracking info segments. In that way, there won’t be a conflict between the two patterns. For example, the workflow permissions pattern uses the first segment of the tracking info. Others may utilize multiple segments. (e.g. new com.orchestranetworks.ps.util.FirstSegmentTrackingInfoHelper(WorkflowAccessRule.SEGMENT\_WORKFLOW\_PERMISSIONS\_USERS))
* **filter** *(optional)*: The records to be selected from may need to be filtered, perhaps based on properties of the parent table record with the selection node to the join table. To accomplish this, an instance of **com.orchestranetworks.schema.TableRefFilter** must be implemented. This is the same interface utilized by a foreign key when implementing a programmatic filter. You may already have a filter on the foreign key from the join table that you could modify to allow for this use, by passing in a flag for which context it’s being invoked from for example. If no additional filtering is required, this doesn’t need to be specified. (e.g. new MenuItemLocationCountryTableRefFilter(true))

# Schema Extensions

The Access Rule is associated with the selection table occurrence in the same manner as all Access Rules, via a Schema Extensions class. See the [above section](#_Access_Rule) for details on the Access Rule. It is not required, but a good practice would be to pass in the name of the Selection Service as a parameter to the Schema Extensions. That way, it could be configured from the model and not require a code change. This is an example of a Schema Extensions class that creates a Multi Select Access Rule:

public class ReferenceExtensions implements SchemaExtensions

{

private String countrySelectServiceID;

@Override

public void defineExtensions(SchemaExtensionsContext pContext)

{

AccessRulesManager manager = new AccessRulesManager(pContext);

manager.setAccessRuleOnOccurrence(

BkReferencePaths.\_Root\_Country.getPathInSchema(),

new CountryAccessRule());

TrackingInfoHelper baseTrackingInfoHelper =

new FirstSegmentTrackingInfoHelper(

WorkflowAccessRule.SEGMENT\_WORKFLOW\_PERMISSIONS\_USERS);

TableRefFilter filter =

new MenuItemLocationCountryTableRefFilter(true);

manager.setAccessRuleOnOccurrence(

BkReferencePaths.\_Root\_Country.getPathInSchema(),

new MultiSelectAccessRule(

countrySelectServiceID, baseTrackingInfoHelper, filter));

}

public String getCountrySelectServiceID()

{

return this.countrySelectServiceID;

}

public void setCountrySelectServiceID(String countrySelectServiceID)

{

this.countrySelectServiceID = countrySelectServiceID;

}

}

# Table Filter

This is not a required component of the Multi Select pattern, but often you will need to filter the records being shown in the selection table, which requires an instance of **com.orchestranetworks.schema.TableRefFilter**. It is passed into the Access Rule upon construction. See [that section](#_Access_Rule) for more details. Here is an example of a filter that can be written, but it is not specific to the Multi Select pattern. In this case, it’s being utilized from both the Menu Item Location table on its foreign key to Country, as well as by the Multi Select pattern. It uses a field passed into the constructor to determine if it’s being called from the context of a Menu Item, which it will be when using Multi Select. The no-argument constructor is always used by EBX when instantiating on a foreign key but we can allow any arguments we want when constructing it for our Access Rule.

public class MenuItemLocationCountryTableRefFilter

implements TableRefFilter

{

private static final String MESSAGE =

"Only countries that match region are allowed.";

private boolean menuItemContext;

private static final Path parentNodeRegionPath = Path.PARENT.add(

BkProductMasterPaths.\_Root\_MenuItemDetails\_MenuItemLocation.

\_\_region);

public MenuItemLocationCountryTableRefFilter()

{

this(false);

}

public MenuItemLocationCountryTableRefFilter(

boolean menuItemContext)

{

this.menuItemContext = menuItemContext;

}

@Override

public boolean accept(Adaptation record, ValueContext context)

{

String countryRegion =

record.getString(BkReferencePaths.\_Root\_Country.\_Region);

String menuItemRegion;

if (menuItemContext)

{

menuItemRegion = (String) context.getValue(

BkProductMasterPaths.\_Root\_MenuItem.\_Region);

}

else

{

menuItemRegion = (String) context.getValue(

parentNodeRegionPath);

}

return ObjectUtils.equals(countryRegion, menuItemRegion);

}

@Override

public void setup(TableRefFilterContext context)

{

context.addFilterErrorMessage(MESSAGE);

}

@Override

public String toUserDocumentation(

Locale locale, ValueContext context)

throws InvalidSchemaException

{

return MESSAGE;

}

}