How to Enhance the Shipping Cockpit

Applicable Releases:
Extended Warehouse Management 9.1 and higher
SAP NetWeaver 7.40

Topic Area:
Extended Warehouse Management – Shipping & Receiving

Version 1.2
July 2014
## Document History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>First official release of this guide</td>
</tr>
<tr>
<td>1.10</td>
<td>Instructions on how to add confirmation dialog boxes to actions added</td>
</tr>
<tr>
<td></td>
<td>(chapter 4.7)</td>
</tr>
<tr>
<td>1.20</td>
<td>Note on BAdI filters (chapter 2.3), minor corrections in code examples</td>
</tr>
<tr>
<td></td>
<td>(section 4.9.3)</td>
</tr>
</tbody>
</table>
## Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Text</td>
<td>Words or characters quoted from the screen. These include field names, screen titles, push buttons labels, menu names, menu paths, and menu options. Cross-references to other documentation</td>
</tr>
<tr>
<td>Example text</td>
<td>Emphasized words or phrases in body text, graphic titles, and table titles</td>
</tr>
<tr>
<td>Example text</td>
<td>File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.</td>
</tr>
<tr>
<td>Example text</td>
<td>User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td>&lt;Example text&gt;</td>
<td>Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.</td>
</tr>
<tr>
<td>EXAMPLE TEXT</td>
<td>Keys on the keyboard, for example, F2 or ENTER.</td>
</tr>
</tbody>
</table>

## Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>⚠️</td>
<td>Caution</td>
</tr>
<tr>
<td>⚡️</td>
<td>Important</td>
</tr>
<tr>
<td>🔧</td>
<td>Note</td>
</tr>
<tr>
<td>🔸</td>
<td>Recommendation or Tip</td>
</tr>
<tr>
<td>➡️</td>
<td>Example</td>
</tr>
</tbody>
</table>
Table of Contents

1. Business Scenario .................................................................................................................. 1

2. Background Information ...................................................................................................... 1
   2.1 Administrator Customizing/Personalization ................................................................. 1
   2.2 FPM Context Based Adaptation (CBA) ............................................................................. 1
   2.3 BAdI-Implementation for FSI (/PLMB/ES_SPI, /PLMB/EX_SPI_TRANSACTION, /PLMU/EX_FRW_APPCC_OVP) ............................................................... 2
   2.4 Shipping Cockpit BAdIs ................................................................................................. 3
   2.5 Exchange feeder class ................................................................................................. 3

3. Prerequisites & Additional Documentation ........................................................................... 4

4. Step-by-Step Procedure ........................................................................................................ 5
   4.1 Recommended Enhancements/Best-Practice ................................................................. 5
      4.1.1 Limiting the Amount of Displayed Columns in Tree UIBBs ................................ 5
   4.2 UI Layout and Appearance ............................................................................................ 9
      4.2.1 Change the UI Layout for a Specific Warehouse/Shipping Office (using context-based adaptation) ................................................................. 9
      4.2.2 Change the UI Layout for a Variant ...................................................................... 10
   4.3 How to Add New Search Criteria .................................................................................. 12
      4.3.1 Add Metadata for the Field ................................................................................... 12
      4.3.2 Add the Field to the Configuration ..................................................................... 13
      4.3.3 Prepare Selection Parameters for Query ............................................................... 14
   4.4 How to Display Additional Fields in the Tree UIBBs .................................................... 15
      4.4.1 Add the Field to the Tree Structure ................................................................. 15
      4.4.2 Add the Field to the Configuration ................................................................ 16
      4.4.3 Select and Map Data ......................................................................................... 16
   4.5 Aggregation ..................................................................................................................... 18
   4.6 Navigation ....................................................................................................................... 19
   4.7 How to Add Confirmation Dialog Boxes to Actions ..................................................... 21
   4.8 How to Extend Existing Actions .................................................................................... 23
   4.9 How to Add Own Buttons and Actions ....................................................................... 24
      4.9.1 Add Metadata for the Action ............................................................................... 24
      4.9.2 Add the Button to the Configuration ................................................................. 24
      4.9.3 Implement the Action ......................................................................................... 25
   4.10 Custom UIBB ................................................................................................................ 26
      4.10.1 Custom UIBB not integrated into FSI Framework .............................................. 27
      4.10.2 Custom UIBB integrated into FSI Framework ................................................... 27
      4.10.3 Opening Customer UIBBs as Dialog Boxes ..................................................... 29
   4.11 Side Panel ...................................................................................................................... 33
      4.11.1 General Information & Adaptation/Enhancement .......................................... 33
      4.11.2 Setup of Content for Measurement Services in Shipping Cockpit .................... 34
      4.11.3 Reuse of Sidepanel for Measurement Services in Other Transactions ............ 34
1. Business Scenario

You are working in a customer implementation of the Extended Warehouse Management system. The customer would like to use the EWM Shipping Cockpit (SCO) for outbound planning and execution and needs to adapt the application to their particular processes. This guide provides you with an overview of the main enhancement possibilities available and some step-by-step procedures for common requirements.

2. Background Information

There are multiple technical possibilities to enhance and adapt the Shipping Cockpit. The Shipping Cockpit is a Web Dynpro Application using the frameworks Floorplan Manager (FPM) and FPM Service Provider Infrastructure Integration (FSI) and Service Provider Infrastructure (SPI). These frameworks offer a plethora of enhancement and adaption options.

In this chapter, we will provide a glimpse at some of the basic possibilities these frameworks offer, as well as a list of BAdIs specifically developed for enhancing the Shipping Cockpit.

⚠️ Recommendation

We highly recommend checking out the in-detail documents about enhancement/adaption options in FPM and SPI, namely the FPM Developer's Cookbook, the guide for enhancing FPM applications and the FSI/SPI wiki. The rest of this guide assumes familiarity with basic web dynpro/FPM concepts.

2.1 Administrator Customizing/Personalization

The Floor Plan Manager (FPM) framework offers a lot of configuration possibilities, either as customizing by a system administrator or as through personalization for each individual user. Some of the features are hiding or rearranging UI Building Blocks (UIBBs), displaying or hiding additional fields or changing the layout of the application. For more details, please consult the FPM Developer's Cookbook.

2.2 FPM Context Based Adaptation (CBA)

Context Based Adaptation (CBA) is one way to change existing FPM configurations (detailing the UI Layout) modification-free. The changed configuration will be displayed based on the values of parameters specified in customizing. These parameters (called “dimensions”) can be supplied via URL parameters while calling the application or by supplying values during runtime in the backend.

⚠️ Important

CBA is only one way of changing the standard configuration. Use it if you need different configuration dependent on the current context (e.g., current warehouse number or shipping office). If you want to change the configuration for all users, please consider the other options, especially administrator customizing.

For a detailed explanation of CBA please refer to the respective chapter in the FPM Developer's Cookbook and the guide for enhancing FPM applications.
2.3 BAdI-Implementation for FSI (/PLMB/ES_SPI, /PLMB/EX_SPI_TRANSACTION, /PLMU/EX_FRW_APPCC_OVP)

The Shipping Cockpit uses the framework FSI (Floor Plan Manager Service Provider Interface Integration) as backend layer and integration with the UI framework (Floor Plan Manager). FSI uses a hierarchical model of nodes to structure backend access. For each UIBB in the shipping cockpit, a corresponding node exists for backend access.

These nodes are accessed via interface /PLMB/IF_SPI_NODE_ACCESS with methods to query, retrieve, insert, update or delete data and execute actions.

FSI provides enhancement spot /PLMB/ES_SPI to intervene at various points of the framework. For a detailed description please refer to the documentation of the individual BAdI-Definitions and the corresponding SPI documentation in SDN. The most interesting BAdIs when enhancing the shipping cockpit are:

- /PLMB/EX_SPI_APPL_ACCESS: Provides methods to change data and execute customer-specific coding before and after each call to a node of the shipping cockpit. That way you can e.g. fill fields

⚠️ CAUTION

Changing data after and especially before standard SCO calls can disrupt effective operation of the Shipping Cockpit.

When changing data after e.g. a query or action call, the safest thing is to only fill customer-own fields added to the respective export structure (see 4.4.1).

- /PLMB/EX_SPI_TRANSACTION: Provides methods to execute customer-specific coding after check before save, save and cleanup

- /PLMU/EX_FRW_APPCC_OVP: Enables you to implement a consumer application controller on the UI level. This provides methods that do additional event processing at several points in time in the transaction. For example, it can be used to add confirmation dialog boxes to actions (see 4.7).

It can also be used to override standard event processing, but this should be used with care so as to not interrupt normal program flow. Implementations of the application controller are always called before the respective methods in the feeder classes.

⚠️ CAUTION

When implementing one of the above BAdIs it is very important that you restrict the validity of the implementation to the Shipping Cockpit by using appropriate filter values. Otherwise, there might be side effects on other FSI applications.

For example, in BAdI /PLMU/EX_FRW_APPCC your implementation should only be called when the filter parameter WD_APPLICATION_NAME is '/SCWM/SCO' for the Shipping Cockpit planning view, respectively '/SCWM/SCO_EXEC' for the execution view.

In /PLMB/EX_SPI_APPL_ACCESS and /PLMB/EX_SPI_TRANSACTION, your implementation should only be valid for the filter value APPLICATION_BUILDING_BLOCK being equal to EWM_SCO.

You can use the constants SC_WD_APPL_SCO, SC_WD_APPL_SCO_EXEC and SC_ABBID from interface /SCWM/IF_SOCO_C in your implementation.
2.4 Shipping Cockpit BAdIs

The Shipping Cockpit has its own enhancement spot /SCWM/ES_SR_SCO, to offer enhancement possibilities which cannot be adequately resolved by means of the BAdIs provided by the FSI framework.

The BAdI /SCWM/EX_SR_SCO_SELECT is offered to allow you to influence the selection of TUs and the delivery query. More precisely, if you want to add new search criteria or want to display additional fields in the Tree UIBBs, in this BAdI you can add these fields to the existing queries if possible or else implement your own selection or determination for those fields. The main cases in which you can use this BAdI are following:

- **New TU search criterion**
  
  If you want to add a new search criterion for the selection of TUs, you need to implement the method CHANGE_TU_SEL_PARAM. (The new selection criterion is dynamically added to the WHERE-clause of the corresponding SELECT-statement.)

- **New delivery search criterion**
  
  If you want to add a new search criterion for the delivery query, you need to implement the method CHANGE_DLV_SEL_PARAM. (The new selection criterion is dynamically added to the delivery query.)

- **Select new TU field for the Tree UIBBs (if field available in database table)**
  
  In order to select an additional field in the TU trees, implement the method ADD_TU_SEL_FIELD. (The new TU field is dynamically added to the result set of the corresponding SELECT-statement.)

- **Select new delivery field in the Tree UIBBs or TU field that is not available in database**
  
  In order to select additional fields in the trees, implement the corresponding method FILL_CUSTOM_FIELDS_TREE_DLV (delivery tree in planning view), FILL_CUSTOM_FIELDS_TREE_TU (TU tree in planning view), FILL_CUSTOM_FIELDS_TREE_EXEC (tree in execution view).

Please confer to the extensive system documentation for more details. We will also provide a walkthrough through all necessary steps in the sections 4.3 and 4.4.

2.5 Exchange feeder class

Feeder classes form application-specific UI layer and as such are directly called by the generic UI framework (Floor Plan Manager). In the Shipping Cockpit, all feeder classes are subclasses of FSI classes (see 2.3). Thus the core functionality is handled in the FSI superclass.

There are a several interfaces that can be implemented to intervene in standard FSI processing. These are used in the Shipping Cockpit to handle e.g. communication between different UI parts (UIBBs), open dialog boxes, handling drag-and-drop etc.

If you have requirements that can only be handled on this layer have a look at 4.10.3 for an example.

⚠️ CAUTION ⚠️

Note that coding executed in the feeder class might interfere with standard SCO processing. Have a close look at what the standard SCO implementation of a given method does before doing your own implementation.

Note also that pure business logic is normally better implemented in FSI or SCO BAdIs, even though it is technically possible to call customer business logic from a feeder class.
3. Prerequisites & Additional Documentation

Guides & Documentation FPM/SPI
- FPM Developer's Cookbook
- FSI/SPI wiki
- Guide for enhancing FPM applications

Documentation Business Context Viewer/Side Panel
- SAP Help for Business Context Viewer (BCV)
- SAP Help for Side Panel (especially chapter "Enhancing/Adapting the Side Panel")
- Documentation of IMG node `EWM->Monitoring->Measurement Services->Define Sidepanel Content for Measurement Services (view cluster /SCWM/VC_MS_SC)`
- SAP Help for Launchpads
4. Step-by-Step Procedure

In this chapter, we provide a list of sample adaptions/enhancements and how it is possible to implement them in the Shipping Cockpit.

4.1 Recommended Enhancements/Best-Practice

In this chapter, we describe enhancements that we recommend doing, e.g. for performance reasons.

4.1.1 Limiting the Amount of Displayed Columns in Tree UIBBs

The Shipping Cockpit Tree UIBBs provide a large number of columns with information. In most cases, only a small fraction of these columns are of interest. Columns can be removed from display via personalization, but that way they are still processed in the backend. Removing columns from the configuration can have significantly positive effects on UI performance. We therefore recommend removing all unnecessary columns (containing data which is never needed) in the configuration. Depending on your use case, you would take one of two approaches:

1. Remove columns for all users: You remove columns which you know are never going to be used in your processed. This should be done via Administrator Customizing (see 2.1).
2. Remove columns only for specific users: Specific users (e.g. in specific warehouses or shipping offices) only use certain columns. In this case, you can for example use Context-Based Adaptations (see 2.2). We only recommend doing this if approach one does not fulfill your requirements, as CBA is additional effort compared to Administrator Customizing.

In our example, we will use Administrator Customizing to only display a few fields in the Shipping Cockpit – Execution.

First, we start the application in Administrator mode. This can be done by opening the Web Dynpro Application in transaction SE80 and choose Web Dynpro Application -> Test -> In Browser – Admin Mode from the menu (see picture below).
An orange and yellow bar at the top should now indicate that you started the application in customizing mode. Press the second wrench symbol in the upper right to enter the show customizable areas. Then, hover over the tree UIBB and press the large wrench symbol in the upper right hand of the tree UIBB.
Enter a description for your customizing and press enter. Choose whether to put your customizing into a transport request and continue.

Go to the section Tree UIBB Schema select all columns you do not need and press the delete button:
Press save. The removed columns should now be removed from the application. They cannot be added to any view even by personalization.
4.2  UI Layout and Appearance

Rearranging/hiding screen elements (buttons, UIBB, columns); changing titles/labels/help texts
Rearranging and hiding existing screen elements (buttons, UIBBs, columns) for all users or for individual users is best done using standard administrator customizing/personalization.
If you want to do more expansive changes, you can do so with one of the other options described in the guide for enhancing FPM applications: Copy Application, Enhancement, Modification or Context-Based Adaptations (CBA). The aforementioned guide contains a helpful list of advantages and disadvantages of these methods and when to use which. Most of these methods are pretty much used in the same way regardless of the specific application. As context-based adaptations are application-specified, we will describe it in more detail.

Tip
In subsequent chapters, whenever changing the configuration is mentioned as a sub step, remember you can do so using any of the aforementioned methods.

By default, three adaptation dimensions are available in the Shipping Cockpit: Warehouse number, shipping office and variant. Warehouse number and shipping office are filled by the backend from the user defaults (see screenshot). If you create an adapted configuration for specific values of these dimensions, this configuration will automatically be applied (at least on UIBB level see below).
The dimension “variant” has to either be filled either in the application URL or in the backend in customer-specific coding. Have a look at chapter 4.1 about how to do this.

Important
Please note that changing the context during runtime will only cause UIBB-specific configuration changes to take place. Thus, changes on floorplan level (rearranging/deleting/adding UIBBs etc.) will only be considered when starting the Shipping Cockpit with the respective parameter values. Since the user can change “warehouse number” and “shipping office” at runtime, changes on the floorplan level should only depend on the parameter “variant”.

4.2.1 Change the UI Layout for a Specific Warehouse/Shipping Office (using context-based adaptation)

The Shipping Cockpit notifies the FPM framework about the current warehouse and shipping office on startup and every time either one is changed. Thus, as long as you only want to do changes of UIBB configuration, all you have to do is create a new adaptation of the configuration and do your changes.
Open up the application configuration for either the Planning View or Execution View (/SCWM/SCO or /SCWM/SCO_EXEC), navigate from there to the OVP configuration and select the button Adaptations & Comparisons in the menu:
Then create a new adaptation by clicking on the **Add** button. You will be asked to enter the dimension values for which you want to create the adaptation. Enter your warehouse number and/or shipping office and the namespace you want to use and then press **OK**.

You can now change any UIBB configuration and save. Whenever a user works with the Shipping Cockpit in the specified warehouse/shipping office, the changed UIBB configurations are applied. In the same way, you can also edit existing adaptations by selecting one in the Adaptations & Comparisons view.

As mentioned above, changes of floorplan configuration are currently not supported in this way. Please refer to chapter 4.2.2 if you want to such changes.

### 4.2.2 Change the UI Layout for a Variant

The dimension variant (SCO_VAR) has been added to the Adaptation Schema to enable you to maintain a different configuration triggered during startup of the Shipping Cockpit. Create your own adapted configuration as shown in the preceding chapter, entering an arbitrary value for the SCO Variant. Afterwards you can add the Shipping Cockpit Application to a role in PFCG and set the parameter SCO_VAR:
Open your role in PFCG and select the Shipping Cockpit Web Dynpro Application in the role menu in edit mode. Select the details button or right-click and choose details. On the popup, enter parameter name SCO_VAR and as value the variant name you have chosen:

Alternatively, you can trigger the context-dependent selection of a different variant in the backend. This can happen in different places in your own coding (e.g. FSI BAdI Implementations or own feeder classes), but it has to happen during the Process-Before-Output phase in the application. See below for a sample of how to trigger the variant above in the backend:

```
DATA: lo_event TYPE REF TO cl_fpm_event,
    lo_fpm  TYPE REF TO if_fpm.
CREATE OBJECT lo_event
  EXPORTING iv_event_id = if_fpm_constants=>gc_event-adapt_context
                     iv_adapts_context = abap_true.
* Set the adaptation context via event parameters
lo_event->mo_event_data->set_value(
  EXPORTING iv_key = 'SCO_VAR' iv_value = 'MY_VAR' ).
```
* Raise the event to change the context
  
  lo_fpm = cl_fpm_factory->get_instance( ).
  lo_fpm->raise_event( lo_event ).

4.3 How to Add New Search Criteria

In this section, we will show how you can easily extend the selection in the Shipping Cockpit by a new field. This could be either a custom field or a standard field. The procedure is analogous for both the Shipping Cockpit Planning and the Shipping Cockpit Execution and only requires a few straightforward steps.

As an example, you will learn how to add the document type, a delivery header field, as search criterion in the execution view of the Shipping Cockpit. There are slight differences between delivery fields and TU fields, which we will point out as we go along.

4.3.1 Add Metadata for the Field

First, the metadata for the new criterion must be declared. Implement the FSI-BAdI /PLMB/EX_SPI_METADATA and add the coding below to the method ENRICH_NODE_DEFINITION. The metadata are defined independently of the warehouse number or other parameters, but must be restricted to the relevant nodes – in our case the selection node in the execution view.

METHOD /plmb/if_ex_spi_metadata-enrich_node_definition.

DATA:
  ls_sp_qry_comp_descr TYPE /plmb/s_spi_component_descr,
  ls_sp_qry_crit_det TYPE /plmb/s_spi_criteria_details,
  ls_sp_qry_options TYPE /plmb/s_spi_query_option.

READ TABLE ct_metadata_node
  WITH KEY name = /scwm/if_sco_c=>sc_node_sel_exec
  ASSIGNING <ls_metadata_node>.

IF sy-subrc = 0.
  READ TABLE <ls_metadata_node>-queries-sp
  WITH KEY query_name = /scwm/if_sco_c=>sc_qry_sel
  ASSIGNING <ls_sp_qry>.

IF sy-subrc = 0.
  CLEAR ls_sp_qry_comp_descr.
  ls_sp_qry_comp_descr-name = 'ZDOCTYPE'.
  ls_sp_qry_comp_descr-type = '/SCDL/DL_DOCTYPE'.
  ls_sp_qry_comp_descr-as_include = abap_false.
  INSERT ls_sp_qry_comp_descr INTO TABLE <ls_sp_qry>-definition-criteria-components.
  CLEAR ls_sp_qry_crit_det.
  ls_sp_qry_crit_det-name = 'ZDOCTYPE'.
  ls_sp_qry_crit_det-supported_sign
    = /plmb/if_mdp_c=>gs_c_supported_sign-only_included.
  ls_sp_qry_crit_det-supported_entry_kind
    = /plmb/if_mdp_c=>gs_c_criteria_entry_kind-default.
  INSERT ls_sp_qry_crit_det INTO TABLE <ls_sp_qry>-definition-criteria-component_details.
4.3.2 Add the Field to the Configuration

Now you can add the field to the appropriate adaptation configuration for the Search UIBB in the Shipping Cockpit Execution (see section on adaptations for details). In the configuration tool, select the search group Delivery and press Add Search Criteria. In the case of TU fields, choose the search group Transport instead.

In the dialog box, select the new field, then click OK and save the adaptation.

In the adapted application, you should now be able to choose the new field as search criterion in the search group Delivery:
4.3.3 Prepare Selection Parameters for Query

Finally, the selection parameters entered by the user have to be translated to the language of the query. At this point, there is a difference between delivery fields and TU fields.

First, let us look into the delivery case: We need to map the name of the search criterion - here ZDOCTYPE - to a valid logical fieldname for the delivery query (see the system documentation of method QUERY of class /SCWM/CL_DLV_MANAGEMENT_PRD or the documents attached to SAP Note 1414179 “Technical documents for software development in EWM”). This may be a custom logical fieldname or a predefined logical fieldname as the ones defined in /SCDL/IF_DL_LOGFNAME_C. The mapping is done in the BAdI /SCWM/EX_SR_SCO_SELECT. Add the following code to method CHANGE_DLV_SEL_PARAM and the new search criterion will be fully functional:

METHOD /scwm/if_ex_sr_sco_select-change_dlv_sel_param.

DATA:
  ls_selection TYPE /scwm/dlv_selection_str.
FIELD-SYMBOLS:
  <ls_sel_param> TYPE /plmb/s_spi_selection_param.

LOOP AT it_sel_param
  ASSIGNING <ls_sel_param>
  WHERE fieldname = 'ZDOCTYPE'.

  CLEAR ls_selection.
  MOVE-CORRESPONDING <ls_sel_param> TO ls_selection.
  ls_selection-fieldname = /scdl/if_dl_logfname_c=>sc_doctype_h.
  APPEND ls_selection TO ct_selection.
ENDLOOP.
ENDMETHOD.

The case of TU fields is a little different. The mapping is done in the same BAdI /SCWM/EX_SR_SCO_SELECT, but in method CHANGE_TU_SEL_PARAM. This method offers three different changing parameters to which new selection parameters can be added, depending on the origin of the field:

- **CT_SELECTION_TU_SR_ACT** for fields from table /SCWM/TU_SR_ACT (e.g. custom fields in include /SCWM/INCL_EEW_TU_SR_ACT)
- **CT_SELECTION_TU_HDR** for fields from table /SCWM/TUNIT (e.g. custom fields in include /SCWM/INCL_EEW_TU_HDR)
- **CT_SELECTION_VEH_HDR** for fields from table /SCWM/VEHICLE (e.g. custom fields in include /SCWM/INCL_EEW_VEH_HDR)

Suppose, for example, that we have appended a custom field ZZFIELD to table /SCWM/TU_SR_ACT. Now we want to enhance the Shipping Cockpit to be able to select by this field. As described in the previous steps, we have to add the search criterion ZZFIELD as TU search criterion to the metadata and the adaptation configuration. Of course, the types of ZZFIELD and ZZFIELD must be compatible. Since the field ZZFIELD comes from table /SCWM/TU_SR_ACT we need to move the selection
entered by the user to the corresponding table `CT_SELECTION_TU_SR_ACT` and the selection will now take into account the new criterion:

```plaintext
METHOD /scwm/if_ex_sr_sco_select-change_tu_sel_param.

DATA:
  ls_seltab TYPE /scwm/s_seltab,
  ls_range  TYPE rstdsselopt,
  ls_sel_param TYPE /plmb/s_spi_selection_param.

LOOP AT it_sel_param INTO ls_sel_param
  WHEREfieldname = 'ZFIELD'.

  CLEAR: ls_seltab, ls_range.
  ls_seltab-fieldname = 'ZZFIELD'.
  MOVE-CORRESPONDING ls_sel_param TO ls_range.
  APPEND ls_range TO ls_seltab-rng.
  INSERT ls_seltab INTO TABLE ct_selection_tu_sr_act.
ENDLOOP.

ENDMETHOD.
```

### 4.4 How to Display Additional Fields in the Tree UIBBs

Displaying additional fields in the Tree UIBBs of the Shipping Cockpit is easy if you use the appropriate enhancement mechanisms offered by the Shipping Cockpit. In this section, we will guide you through the necessary steps necessary for displaying additional fields in the Tree UIBBs of the Shipping Cockpit.

We will add a custom field to the execution view of the Shipping Cockpit. There are slight differences between delivery fields and TU fields, which we will point out along the way.

↑ **Check Available Fields**

Many fields in the Tree UIBBs are hidden by default but can be easily displayed by using the Web Dynpro personalization or customizing features, compare the corresponding section above. Before trying to add a new field, check if it is not already available.

#### 4.4.1 Add the Field to the Tree Structure

This step can be skipped if the field is newly added to one of the following append structures:

- `/SCDL/INCL_EEW_DLV_HEAD_STR` (delivery header)
- `/SCWM/INCL_EEW_TU_SR_ACT` (TU activities)
- `/SCWM/INCL_EEW_TU_HDR` (TU header)
- `/SCWM/INCL_EEW_VEH_HDR` (vehicle header)

In these cases, the field is automatically included in the data dictionary structures for the Tree UIBBs and you can directly jump to the next step.

If you are adding an already existing field or a new field in an include different from the ones listed above, you have to append it to the corresponding data dictionary structure. If the field is a delivery field, you have to append it to structure `/SCWM/S_SCO_D_DLV_DATA`. New TU or vehicle fields have to be appended to `/SCWM/S_SCO_D_TU_DATA`. If you want to include a new field only to the execution view of the Shipping Cockpit, you can use the structures `/SCWM/S_SCO_D_STATUS_DLV` and `/SCWM/S_SCO_D_STATUS_TU` instead.
In our case, let us suppose that the field is a delivery field that should be displayed in all trees of the Shipping Cockpit, so use transaction SE11 to append a field ZZFIELD of compatible type to the structure /SCWM/S_SCO_D_DLV_DATA:

<table>
<thead>
<tr>
<th>Component</th>
<th>Typing Method</th>
<th>Component Type</th>
<th>Data Type</th>
<th>Length</th>
<th>Decl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZZFIELD</td>
<td>1 Types</td>
<td>CHAR10</td>
<td>CHAR</td>
<td>10</td>
<td>0 CI</td>
</tr>
</tbody>
</table>

### 4.4.2 Add the Field to the Configuration

Start the configuration tool to edit the adaptation configuration for the Tree UIBB, press Add Column.

**Tree UIBB Schema**

- Column: SHIP_TO
- Column: TU_NUM_EXT
- Column: DOCNO
- Column: NUMBER_OF_PRDO_ITEMS
- Column: CARRIER
- Column: TRANSMEANS_TYPE
- Column: TRANS_MODE

### 4.4.3 Select and Map Data

This step explains how to select the new fields and pass it to the UI. Any field in customer include /SCDL/INCL_EEW_DLV_HEAD_STR (delivery header) is automatically mapped to the UI structure so we can skip this step. For TU fields that are available on the database, there is a simple alternative described further below.

All other fields need to be determined in a customer implementation of the Shipping Cockpit- BAdI /SCWM/EX_SR_SCO_SELECT. For each of the trees there is a method to fill customer data in the buffer tables. In the case of the execution view, we need to implement method FILL_CUSTOM_FIELDS_TREE_EXEC:

```
METHOD /scwm/if_ex_sr_sco_select-fill_custom_fields_tree_exec.
FIELD-SYMBOLS:
```
TYPE /scwm/s_sco_dlv_data.

"[Customer determination of field ZZFIELD...]

"Assign determined values to buffer table
LOOP AT ct_dlv ASSIGNING <ls_dlv>.
  <ls_dlv>=zzfield = ...
ENDLOOP.
ENDMETHOD.

⚠️ CAUTION

Never overwrite standard fields, because this will not only impact the display of data but also the behavior of the actions in the Shipping Cockpit.

After these changes the new field will be displayed in the delivery rows in the Tree UIBB of the execution view.

🔍 Aggregation

If you want to aggregate this field to higher hierarchy levels, further steps are necessary as described in the section about aggregation 4.5

🔍 Impact on performance

Keep in mind that such an implementation can have a significant effect on the runtime of the selection in the Shipping Cockpit. It is important to make sure early in the project that the implementation is as efficient as possible and that the runtimes are acceptable for the customer.

🔍 Alternative solution: FSI-BAIds

The FSI provides BAdI /PLMB/EX_SPI_APPL_ACCESS, which lets you manipulate node data after or before actions, queries or retrieves. In some cases, this might be an alternative to the Shipping Cockpit BAdIs described above.

Now, let us come to the handling of TU fields. The same approach as described above can be used here, but there is a somewhat simpler way if the new field comes from one of the database tables /SCWM/TU_SR_ACT, /SCWM/TUNIT, or /SCWM/VEHICLE. In this case, you can implement BAdI /SCWM/EX_SR_SCO_SELECT, method ADD_TU_SEL_FIELD. This method offers three different changing parameters to which new fields can be added, depending on the origin of the field:

- **CT_ADD_FIELD_TU_SR_ACT** for fields from table /SCWM/TU_SR_ACT (e.g. custom fields in include /SCWM/INCL_EEW_TU_SR_ACT)
- **CT_ADD_FIELD_TU_HDR** for fields from table /SCWM/TUNIT (e.g. custom fields in include /SCWM/INCL_EEW_TU_HDR)
- **CT_ADD_FIELD_VEH_HDR** for fields from table /SCWM/VEHICLE (e.g. custom fields in include /SCWM/INCL_EEW_VEH_HDR)

Suppose, for example, that we have appended a custom field ZZZT_FIELD to table /SCWM/TU_SR_ACT. Now we want this field to be selected and displayed in the Shipping Cockpit. As described in the previous steps, we have to add a field ZZUI_FIELD to the dictionary structure and to the adaptation configuration. Of course, the types of ZZZT_FIELD and ZZUI_FIELD must be compatible. Since the field ZZZT_FIELD comes from table /SCWM/TU_SR_ACT we need to add the
mapping to the corresponding table CT_ADD_FIELD_TU_SR_ACT. In the BAdI method, we need to provide the field name ZZTU_FIELD and the field alias ZZUI_FIELD, i.e. the name of the field in the UI structure /SCWM/S_SCO_D_TU_DATA. If the two names are identical, the field alias can be left empty.

METHOD /scwm/if_ex_sr_sco_select-add_tu_sel_field.

DATA:
  ls_field_name TYPE /scwm/s_sco_sel_field_name.

  ls_field_name-field_name = 'ZZTU_FIELD'.
  ls_field_name-field_alias = 'ZZUI_FIELD'.
  APPEND ls_field_name TO ct_add_field_tu_sr_act.

ENDMETHOD.

Now the new field will be selected and appear in the Tree UIBB.

4.5 Aggregation

If new fields are introduced to the Tree UIBBs in the Shipping Cockpit (as described in Section 4.4), they are not automatically aggregated to higher hierarchy levels. In order to do this, the aggregation rule for the new field needs to be implemented in the Shipping Cockpit-BAdI /SCWM/EX_SR_SCO_AGGREGATE. The BAdI has a separate method for each of the trees in the Shipping Cockpit, so the aggregate can (or must) be defined for each tree (please have a look at the BAdI implementation for more detailed information).

Let us suppose that the customer is not satisfied with the aggregation logic of the field “Carrier”. By default, if the carriers in the child nodes are not all the same, then “* * *” is displayed in the parent node. However, the customer would prefer to see the first carrier in the list.

⚠️ CAUTION

Never change the aggregation logic for standard fields, because this will not only impact the display of data but also the behavior of the actions in the Shipping Cockpit. Also keep in mind that the aggregation logic is called every time the tree is build and is called for each hierarchy node, so only simple aggregation rules should be used. Complex rules or determinations in the aggregation rules will very likely have a negative impact on performance.

So we are not allowed to change the aggregation logic of the field “Carrier”, but we can add a new field “ZZCARRIER” as described in Section 4.4 and then copy the field value by implementing the method FILL_CUSTOM_FIELDS_TREE_EXEC of BAdI /SCWM/EX_SR_SCO_SELECT.

METHOD /scwm/if_ex_sr_sco_select-fill_custom_fields_tree_exec.

FIELD-SYMBOLS:
  <ls_dlv> TYPE /scwm/s_sco_dlv_data.

LOOP AT ct_dlv ASSIGNING <ls_dlv>.
  "Copy carrier to customer field
  <ls_dlv>-zzcarrier = <ls_dlv>-carrier.
ENDLOOP.
Now we implement method AGGREGATE_TREE_EXEC of BAdI /SCWM/EX_SR_SCO_AGGREGATE to define the new aggregation rule:

METHOD /scwm/if_ex_sr_sco_aggregate-aggregate_tree_exec.

FIELD-SYMBOLS:
  <ls_tree_exec> TYPE /scwm/s_sco_tree_exec.

LOOP AT it_child ASSIGNING <ls_tree_exec>
  WHERE zzcarrier IS NOT INITIAL.
    "Take first non-initial value
    cs_parent-zzcarrier = <ls_tree_exec>-zzcarrier.
    RETURN.
  ENDLOOP.
ENDMETHOD.

Now the old field can be hidden from the (adapted) component configuration of the Tree UIBB and the new field can be included. If you want to define navigation for the new field, please have a look at Section 4.6.

4.6 Navigation

We continue our example from the previous section by defining navigation for a new carrier field to transaction BSSR_BOR_OBJECT. In the screen shots the field is called ZFIELD, but this could just as well be ZZCARRIER in your system. First, the attributes of the column have to be changed in the adaptation of the configuration of the tree UIBB: The display type needs to be set to “Link to Action” and the link type to “Navigation”:

![Attributes of Column: ZFIELD](image)
Now the entries in the new tree column will be displayed as links (i.e. blue and underlined). Now we need to define the behavior of the navigation. This is done by customizing the launchpad GENERAL NAVIGATION (role EWM_SCO) in transaction LPD_CUST. In our example, we do not need to define a new navigation behavior, we only want to copy the behavior of the navigation already defined for the standard field “Carrier”. Double-click on the target “Display Carrier”:

Copy the navigation target (right click → copy) and name the new target, for example as “Display Carrier for custom field”. Then change the application alias by appending the prefix ALIAS_ to the field name, e.g. ALIAS_ZFIELD if your field is called ZFIELD or ALIAS_ZZCARRIER if the name is ZZCARRIER:

Now navigate to the parameter mapping to define which values are passed to navigation:
In the popup, add the field NAVIGATION_VALUE to the list of parameters. The field NAVIGATION_VALUE is a generic value which always contains the value of the cell on which the user has clicked. Moreover, there are other parameters available (most importantly, the warehouse number), which can be mapped to the target application.

Save the customizing. The newly defined navigation is now working.

Please note that the navigation via launchpads offers many more customizing options than the ones shown in the example. Further information on launchpads can be found in the FPM Developer’s Cookbook or in the SAP Help for Launchpads. The two launchpads used in the Shipping Cockpit are GENERAL_NAVIGATION and RELATED_LINKS, which belong to role EWM_SCO.

### 4.7 How to Add Confirmation Dialog Boxes to Actions

For some actions, you might want the user to get a confirmation dialog box. This is a popup window asking the user to confirm that he or she wants to execute the action. It is usually used for actions that make business critical changes that are hard to reverse (for example, deleting something). In the standard delivery, only the action **Delete TU** and actions performed for all TUs assigned to a vehicle have confirmation dialog boxes (implemented in the feeder classes).
Adding a confirmation dialog box to an action is best done in an implementation of business add-in (BAI) /PLMU/EX_FRW_APPCC_OVP in enhancement spot /PLMU/ES_FRW_CONSUMER_APPCC. You have to perform the following steps:

1. Create your own implementation of BAdI /PLMU/EX_FRW_APPCC_OVP. Use the name of the Web Dynpro application name as a filter. The Web Dynpro application names for Shipping Cockpit Planning and Shipping Cockpit Execution are /SCWM/SCO and /SCWM/SCO_EXEC. See below for an example for a BAdI implementation valid for both Planning and Execution.

<table>
<thead>
<tr>
<th>Filter Combinations</th>
<th>Filter Value 1</th>
<th>Compare</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>/SCWM/SCO</td>
<td>=</td>
<td>W0_APPLICATION_NAME</td>
</tr>
<tr>
<td>m</td>
<td>/SCWM/SCO_EXEC</td>
<td>=</td>
<td>W0_APPLICATION_NAME</td>
</tr>
</tbody>
</table>

2. Implement method NEEDS_CONFIRMATION. This method is called before the processing of any action. Exported object EO_CONFIRMATION_REQUEST has to be filled with the information about the dialog box. See below for sample coding.

Make sure to check the event ID so that you get confirmation dialog boxes for specific actions only. You can find the event ID for standard actions in the Web Dynpro configuration, and use the constants starting with SC_ACT in interface /SCWM/IF_SCO_C in the coding. You can set the text for the approve and cancel buttons, the title and a text for the dialog box itself.

In the example, the action used is action CLOSE TU. We use the text in the dialog box to describe to consequences of executing the action.

```
METHOD /plmu/if_ex_frw_appcc_ovp-needs_confirmation.
  DATA:
    lt_confirmation_text  TYPE string_table,
    lv_window_title       TYPE string,
    lv_button_text_approve TYPE string,
    lv_button_text_reject TYPE string.
  IF io_event->mv_event_id = /scwm/if_sco_c=>sc_act_end_load_tu.
    "set pop-up attributes
    lv_window_title = 'TU will be closed'.
    lv_button_text_approve = 'Close TU'.
    lv_button_text_reject = 'Cancel'.
    APPEND 'Selected TU will be closed.' TO lt_confirmation_text.
    APPEND 'Not loaded HUs/delivery items will lose their TU ass.'
    TO lt_confirmation_text.
    APPEND 'Status Loading Finished will be set for the TU.'
    TO lt_confirmation_text.
    APPEND 'Open Loading WTs will be cancelled' TO lt_confirmation_text.
  CREATE OBJECT co_confirmation_request
    EXPORTING
      it_confirmation_text     = lt_confirmation_text
      iv_window_title          = lv_window_title
      iv_button_text_approve   = lv_button_text_approve
      iv_button_text_reject    = lv_button_text_reject.
 ENDIF.
ENDMETHOD.
```
4.8 How to Extend Existing Actions

In some cases, it might be necessary to introduce additional checks before an existing action or perform follow-up actions afterwards. In these cases, you can use the BAdIs provided by the FSI framework.

In previous sections, we have added a customer field ZZFIELD to the UI. Let us continue with this example by supposing that this field is filled in a custom-developed transaction (e.g. an adapted RF-transaction) during the packing step. It is a mandatory field and it needs to be checked that the field is correctly filled before the deliveries can be posted goods issue.

This may be solved by implementing the BAdI /PLMB/EX_SPI_APPL_ACCESS as shown here:

METHOD /plmb/if_ex_spi_appl_access-before_action.

DATA: lo_fpm TYPE REF TO if_fpm,
  lv_msg TYPE string.

IF iv_node_name = /scwm/if_sco_c=>sc_node_tree_exec AND
  iv_action_name = /scwm/if_sco_c=>sc_act_post_gi.

TRY.
  "====================================================
  Here custom logic determines whether the customer
  " field is correctly filled
  "====================================================
  CATCH /scwm/cx_sr_error.
  MESSAGE e000(/scwm/shp_rcv) WITH 'ZZFIELD not filled' INTO lv_msg.
  io_collector->add_system_message( ).
  cv_skip_standard = abap_true.
  cv_failed = abap_true.
  ENDTry.
ENDIF.
ENDMETHOD.

Note

In your own coding you should of course create your own message. The message call above is not meant to be an example of good programming practice, but to provide you with a functioning and transparent template.

If you need to perform any follow-up actions (e.g. updating affected data in custom tables), you can do this in an analogous fashion by implementing the method AFTER_ACTION of the same BAdI /PLMB/EX_SPI_APPL_ACCESS.
4.9 How to Add Own Buttons and Actions

It is possible to add own buttons to the Tree UIBB and link newly defined actions to those buttons. Let us assume that the Shipping Office clerk needs to be able to manually print certain documents to hand over to the driver.

4.9.1 Add Metadata for the Action

First, the metadata for the new action must be declared. Implement the FSI-BAdI /PLMB:EX_SPI_METADATA and add the coding below to the method ENRICH_NODE_DEFINITION. The metadata are defined independently of the warehouse number or other parameters, but must be restricted to the relevant nodes – in our case the tree node in the execution view.

METHOD /plmb/if_ex_spi_metadata-enrich_node_definition.

DATA: ls_metadata_action TYPE /plmb/s_spi_metadata_action.
FIELD-SYMBOLS: <ls_metadata_node> TYPE /plmb/s_spi_metadata_node.

READ TABLE ct_metadata_node
   WITH KEY name = /scwm/if_sco_c=>sc_node_tree_exec
   ASSIGNING <ls_metadata_node>.

IF sy-subrc = 0.
   "add customer action for printing documents
   CLEAR ls_metadata_action.
   ls_metadata_action-name = 'ZPRINT'.
   ls_metadata_action-not_save_rel = abap_true.
   ls_metadata_action-sideeffect = /plmb/if_mdp_c=>gs_c_sideeffect-none.
   ls_metadata_action-structure = space.
   INSERT ls_metadata_action INTO TABLE <ls_metadata_node>-actions.
ENDIF.

ENDMETHOD.

Note

In this example, the action requires no subsequent saving of the business objects. If this were the case, you would have to set the flag NOT_SAVE_REL to false.

4.9.2 Add the Button to the Configuration

Start the configuration tool to edit the adaptation configuration for the Tree UIBB in the Shipping Office execution view, press Add Toolbar Element on the tab Toolbar Schema.

In the dialog box, select your new action ZPRINT and press OK.
In the attributes for the new button, add the description text for the button. Then save your changes.

When you restart the Shipping Cockpit execution view, the new button now appears on the UI.

4.9.3 Implement the Action

In this last step, you implement the actual action. You can do this in the method BEFORE_ACTION of the BAdI /PLMB/EX_SPI_APPL_ACCESS.

METHOD /plmb/if_ex_spi_appl_access-before_action.

DATA:
  lo_data_access_exec TYPE REF TO /scwm/cl_sco_data_access_exec,
  lt_node_data TYPE /scwm/tt_sco_tree_exec.

IF iv_node_name = /scwm/if_sco_c=>sc_node_tree_exec AND
  iv_action_name = 'ZPRINT'.
  cv_skip_standard = abap_true.
First retrieve data of selected lines
lo_data_access_exec ?= /scwm/cl_sco_data_access=>get_instance( ).

lo_data_access_exec->retrieve_tree(  
    EXPORTING  
        it_tree_exec_id = it_node_id  
    IMPORTING  
        et_tree_exec = lt_node_data ).

TRY.
"======================================================================
 " Here custom logic triggers the printing of the
 " documents for the selected TUs.
 "======================================================================
CATCH /scwm/cx_sr_error.  
MESSAGE e000(/scwm/shp_rcv) WITH 'Printing error' INTO lv_msg.  
io_collector->add_system_message( ).  
cv_failed = abap_true.  
ENDTRY.
ENDIF.
ENDMETHOD.

Note

In your own coding you should of course create your own message. The message call above is not meant to be an example of good programming practice, but to provide you with a functioning and transparent template.

In this example, you do not need to save, but for other actions that require saving the Shipping & Receiving business objects, you need to raise an FPM event that will trigger the call of the method SAVE of the service provider. You can use the following code snippet:

```
DATA:
  lo_fpm TYPE REF TO if_fpm,
  lo_event TYPE REF TO cl_fpm_event.

"create SAVE-event and raise it
lo_event => cl_fpm_event=>create_by_id( cl_fpm_event=>gc_event_save ).
lo_fpm => cl_fpm_factory=>get_instance( ).
lo_fpm->raise_event( lo_event ).
```

4.10 Custom UIBB

Customer-Specific UIBBs can easily be added to the Shipping Cockpit using Context-Based Adaptation. You can either add standard FPM UIBBs (see FPM Developer's Cookbook on how implement such) or add your own FSI UIBBs. The advantage of the latter might be that you keep the general architecture of the Shipping Cockpit.
4.10.1 Custom UIBB not integrated into FSI Framework

4.10.1.1 Create your own UIBB
Create a feeder class for your UIBB (freestyle UIBB or GUIBB) as documented in the FPM Developer's Cookbook or the guide for enhancing FPM applications. Implement action handling, data retrieval etc. as you see fit.

4.10.1.2 Add the UIBB to the SCO configuration
Create an adaptation for the Overview Page (OVP) configuration of either Planning or Execution View (see 4.2.1) or choose an existing adaptation. In the Overview Page Schema in the configuration, add a new UIBB of your chosen type (see image below, highlighted form has been added via adaptation).

Rearrange the UIBBs as you like and configure the UIBB as usual. Your UIBB will be called as usual by the standard Floorplan Manager transaction logic.

4.10.2 Custom UIBB integrated into FSI Framework

4.10.2.1 Add a New Node to the SCO Meta Model
Implement method /PLMB/IF_EX_SPI_METADATA~ENRICH_NODE_DEFINITION of BAdI /PLMB/EX_SPI_METADATA. Add a new customer node to the meta model. In the example below, a form UIBB to display inspection data for TUs in the Execution View is added.

```
METHOD /plmb/if_ex_spi_metadata~enrich_node_definition.
    DATA:
        ls_metadata_node LIKE LINE OF ct_metadata_node,
        ls_metadata_action TYPE /plmb/s_spi_metadata_action.

    * Metadata for Inspection Form
        ls_metadata_node-name = 'INSPECTION_DATA',
        ls_metadata_node-data_struc = 'Z_S_INSPECTION',
        ls_metadata_node-id_struc = 'Z_S_K_INSPECTION'.

        ls_metadata_action-name = 'SET_TO_COMPLETED'.
        APPEND ls_metadata_action TO ls_metadata_node-actions.
        APPEND ls_metadata_node TO ct_metadata_node.

ENDMETHOD.
```

If you want buttons on your UIBB, add corresponding actions to the meta model as shown in the example.
4.10.2.2 (Optional) Implement Feeder Class

If you have no requirements on the UI layer, you can use the FSI standard feeder classes. Otherwise, create a new customer feeder class as a subclass of one of the FSI feeder classes (/PLMU/CL_FRW_G_FEEDER_FORM, /PLMU/CL_FRW_G_FEEDER_LIST, ...).

4.10.2.3 Implement Business Logic

Implement BAdI /PLMB/EX_SPI_APPL_ACCESS. Depending on the type of operation you want to support (queries, actions, retrieve, delete etc.) with your node, you have to implement the corresponding methods. Make sure to set parameter CV_SKIP_STANDARD for each method. Otherwise the call will be forwarded to the standard SCO processing and will cause a short dump as the SCO backend does not recognize your node.

In the example, method BEFORE_ACTION is implemented to handle the action defined in the meta model:

METHOD /plmb/if_ex_spi_appl_access-before_action.

    IF iv_node_name = 'INSPECTION_DATA' AND
        iv_action_name = 'SET_TO_COMPLETED',
        set_to_completed( it_tu_id = it_node_id ).
    ENDF.

    " Prevent call of SCO standard processing
    cv_skip_standard = abap_true.
ENDMETHOD.

4.10.2.4 Add the FSI UIBB to the SCO Configuration

Create an adaptation for the OVP configuration of either Planning or Execution View (see 4.2.1) or choose an existing adaptation. In the Overview Page Schema in the configuration, add a new UIBB of your chosen type (see 4.10.1.2)

Configure the UIBB and set the feeder class to either the standard FSI feeder class or your own (see 4.10.2.2).
Fill the feeder class parameters, in the example:

**Edit Parameters**

*Feeder Class: ZSK_CL_FEED_INSPECT_FORM*  
Feeder for TU inspection form

Parameters

- **Application Building Block ID:** EWM_SCO
- **Node Name:** INSPECTION_DATA
- **RETRIEVE by Association:**

Refer to the [FSI/SPI wiki](#) on how to fill the parameters of other feeder classes.

### 4.10.3 Opening Customer UIBBs as Dialog Boxes

It is possible to open customer UIBBs as dialog boxes, triggered by actions of standard SCO UIBBs. This is one of the possible applications of exchanging the feeder class.
4.10.3.1 Create Customer Feeder Class

Create a new feeder class as a subclass to one of the SCO feeder classes (/SCWM/CL_SCO_FEED*). You can see the correct feeder class in the configuration of the UIBB from which you want to trigger the opening of the dialog box. In the following example, we replace the feeder class for the Transportation Units tree in the Shipping Cockpit Planning. We create a customer feeder class as a subclass of /SCWM/CL_SCO_FEED_TREE_TU.

4.10.3.2 Redefine Feeder Class Method

Redefine one of the methods of the feeder class, or implement a new FSI feeder class callback interface (see FSI/SPI wiki for all available interfaces). Callback interfaces are used to implement custom coding in the otherwise generic FSI feeder classes. Appropriate methods in many cases are:

- /PLMU/IF_FRW_G_ACTIONS~PROCESS_ACTION_EVENT which processes events triggered by FSI meta model actions
- /PLMU/IF_FRW_G_GLOBAL_EVENTS~PROCESS_GLOBAL_EVENT which processes events triggered by or in FPM

**CAUTION**

When redefining an existing method, you should always make sure to call the superclass in your own implementation to ensure that standard SCO processing is not affected.

In the two examples, a dialog box is opened reacting on a selected row in the tree UIBB (example 1) and a customer button (example 2, see 4.9 on how to add a customer button).

**METHOD** /plmu/if_frw_g_global_events~process_global_event.

**DATA:**
- lo_fpm TYPE REF TO if_fpm,
- lo_event TYPE REF TO cl_fpm_event,
- lt_tree_tu TYPE /scwm/tt_sco_tree_tu.

* Call standard event processing

```
super~/plmu/if_frw_g_global_events~process_global_event(
  EXPORTING
  io_event = io_event
  iv_raised_by_own_ui = iv_raised_by_own_ui
  IMPORTING
  ev_skip_default = ev_skip_default
  ev_result = ev_result
  et_messages = et_messages )
```

* Open dialog box

```
CASE io_event->mv_event_id.
  WHEN if_fpm_guiibb_tree=>gc_fpm_event_on_lead_sel.
```

* Get selected rows of tree

```
mo_context~/plmu/if_frw_context~get_selection(
  IMPORTING
  et_selection = lt_tree_tu )
```

* Create event and fill parameters

```
lo_fpm = cl_fpm_factory~/get_instance( ).
CREATE OBJECT lo_event
  EXPORTING
  iv_event_id = cl_fpm_event~/gc_event_open_dialog_box.
```
This first example also shows how data can be transferred to a custom UIBB. The standard event processing is called at the start of the method, as opening the dialog box here is independent of standard processing.

METHOD /plmu/if_frw_g_actions~process_action_event.
DATA: lo_fpm TYPE REF TO if_fpm.

CASE lo_event~mv_event_id.
  WHEN 'CUST_ACTION'.
    " For customer action, open dialog box
    lo_fpm = cl_fpm_factory~get_instance( ).
    lo_fpm~open_dialog_box( 
      iv_dialog_box_id = 'CUST_DIALOG_BOX' ).
  " Skip call to Service Provider
  ev_skip_default = abap_true.
  WHEN OTHERS.
    " For all other actions, call standard processing
    super~/plmu/if_frw_g_actions~process_action_event( 
      EXPORTING 
      io_event = io_event
      iv_index = iv_index
      IMPORTING 
      ev_skip_default = ev_skip_default 
      ev_result = ev_result
      et_messages = et_messages ).
  ENDCASE.
ENDMETHOD.

In the second example, no data is transferred to the dialog box, thus we can use a shortened call to open it. By the way, the default processing is skipped. As we only want to open a dialog box, we do not have to call down to the SPI Service Provider Layer.

4.10.3.3  Exchange the Feeder Class in Web Dynpro Configuration

Create a context-based adaptation (see 4.2.1 ) for the UIBB where you want to exchange the feeder class. All configurations in SCO start with /SCWM/WDCC_SCO*. Note again that you have to go to the UIBB configuration via the OVP/application configuration to create a new adaptation.
In the UIBB configuration, press button **Feeder Class** and enter your newly created feeder class.

Confirm the warning message, press button **Edit Parameters** and leave the parameters as they are.
Now, you can set an external break-point at your redefined or newly implemented method to verify that your coding is called.

### 4.11 Side Panel

#### 4.11.1 General Information & Adaptation/Enhancement

The Side Panel is a way to display contextual information from the Business Context Viewer (BCV) in a Panel on the side of an application (see screenshot below). There are numerous ways to adapt the existing content of a Side Panel and create new content. For general ways to do this please refer to the [SAP Help for Business Context Viewer (BCV)](https://help.sap.com) and [Side Panel for Business Suite](https://help.sap.com).

Additionally for EWM, we deliver content to display the value of Measurement Services in the Side Panel. For this content, there are EWM-specific ways for adaptation, which will be described in the following chapter.
4.11.2 Setup of Content for Measurement Services in Shipping Cockpit

You can setup the measurement services that shall be displayed in the sidepanel of the Shipping Cockpit in Customizing Extended Warehouse Management->Monitoring->Measurement Services-> Define Side Panel Content for Measurement Services.

4.11.3 Reuse of Sidepanel for Measurement Services in Other Transactions

You have already defined two tailored measurement services for your warehouse W001.

For example, you have defined the following measurement services:

- M001: Warehouse orders (WOs) created today, not yet confirmed
- M002: WOs created today, confirmed


You now want to have the data displayed in a sidepanel next to transaction /SCWM/WAVE for users assigned to user role <ZROLE> which as well contains transaction /SCWM/WAVE.

You define the following entries in Customizing Extended Warehouse Management-> Monitoring-> Measurement Services-> Define Side Panel Content for Measurement Services:

**Define Context for Measurement Service Usage:**

<table>
<thead>
<tr>
<th>MS_SC_CONTEXT</th>
<th>MS_SC_CONTEXT_TXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SCWM/WAVE</td>
<td>Wave Management</td>
</tr>
</tbody>
</table>

**Define Measurement Service Groups:**

<table>
<thead>
<tr>
<th>LGNUM</th>
<th>MS_SC_GRP</th>
<th>MS_SC_GRP_TXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>W001</td>
<td>G001</td>
<td>Wave Statistics</td>
</tr>
</tbody>
</table>

**Assign Measurement Service Groups:**

<table>
<thead>
<tr>
<th>LGNUM</th>
<th>MS_SC_GRP</th>
<th>MS_SERV</th>
<th>MS_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>W001</td>
<td>G001</td>
<td>M001</td>
<td>T</td>
</tr>
<tr>
<td>W001</td>
<td>G001</td>
<td>M002</td>
<td>T</td>
</tr>
</tbody>
</table>

**Set Default Measurement Service Groups:**

<table>
<thead>
<tr>
<th>LGNUM</th>
<th>MS_SC_CONTEXT</th>
<th>MS_SC_GRP_TYPE</th>
<th>MS.SC_GRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>W001</td>
<td>/SCWM/WAVE</td>
<td>01</td>
<td>G001</td>
</tr>
</tbody>
</table>

You have already activated BC Set /SCWM/SR_SCO_BCV.
In view NWBC_VC_GUI_TAG you maintain the following entries (see background information for more details)

In transaction PFCG for role \(<ZROLE>\) you create new entries

Maintain parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDSIDEPANELRENOTECONSUMER</td>
<td>X</td>
</tr>
<tr>
<td>WDSIDEPANELRENOTEPRODUCER</td>
<td>X</td>
</tr>
</tbody>
</table>

Create an application configuration for Web Dynpro Application WDR_CHIP_PAGE and configure the application configuration as follows: (In order to be able to create the chip page you need authorization for object S_START for application WDR_CHIP_PAGE).

Drag&Drop the Current Data BCV-CHIP to the content of the page
Save the configuration
In the application configuration set

If you now logon with a user assigned to role ZROLE the transaction /SCWM/WAVE is enabled for sidepanel and the defined measurement services for the context are displayed in the sidepanel.