Icons in Body Text

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>🚨</td>
<td>Caution</td>
</tr>
<tr>
<td>🎨</td>
<td>Example</td>
</tr>
<tr>
<td>📝</td>
<td>Note</td>
</tr>
<tr>
<td>🧵</td>
<td>Recommendation</td>
</tr>
<tr>
<td>ℹ️</td>
<td>Syntax</td>
</tr>
</tbody>
</table>

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see Help on Help → General Information Classes and Information Classes for Business Information Warehouse on the first page of any version of SAP Library.

Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example text</strong></td>
<td>Words or characters quoted from the screen. These include field names, screen titles, pushbuttons 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>Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.</td>
</tr>
<tr>
<td>Example text</td>
<td>Output on the screen. This includes 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>Exact user entry. 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>
Table of Contents

Copyright........................................................................................................................2
Icons in Body Text..........................................................................................................3
Typographic Conventions ...............................................................................................3

Floorplan Manager............................................................................................................. 11

Getting Started ................................................................................................................... 11

User Interface Building Blocks ...................................................................................... 12
IF_FPM_UI_BUILDING_BLOCK Interface................................................................ 12
Creating a Simple FPM Application.............................................................................. 14
Creating a Web Dynpro Component .............................................................................. 14
Creating a Web Dynpro Application ............................................................................. 16
Using Application Parameters ....................................................................................... 16
Creating an Application Configuration with the FPM Configuration Editor................... 17
Configuring the Component and IDR Configurations.................................................. 18
Configuring the FPM_OIF_COMPONENT ................................................................ 18
Configuring the FPM_GAF_COMPONENT ................................................................ 18
Configuring the FPM_IDR_COMPONENT ................................................................ 19

Testing your FPM Application....................................................................................... 19

Application Creation Tool (ACT) – Shortcut for Creating Applications......................... 20

Configuration Editor for Floorplan Manager ................................................................ 21
Form Editor for Floorplan Manager ............................................................................. 23
List Editor for Floorplan Manager .............................................................................. 24
Search Component Editor for Floorplan Manager ....................................................... 26
Tabbed Component Editor for Floorplan Manager ...................................................... 27

Wire Model ......................................................................................................................... 28
IF_FPM_UIBB_MODEL Interface ............................................................................... 29
IF_FPM_FEEDER_MODEL Interface ......................................................................... 29
FPM on BOL................................................................................................................... 31
Creating a GUIBB on BOL ............................................................................................ 32
Creating an FPM Application on BOL .......................................................................... 32

Design Time with the FPM Configuration Editor............................................................. 33

Floorplan Instances in the FPM Configuration Editor.................................................. 34
OIF Instance .................................................................................................................... 34
GAF Instance ................................................................................................................... 34
OVP Instance .................................................................................................................. 35

Adding and Activating Sub-Steps for GAF Applications.............................................. 35

FPM Toolbar .................................................................................................................... 36
Differences between an OIF and a GAF Toolbar......................................................... 36
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding Elements to a Toolbar</td>
<td>37</td>
</tr>
<tr>
<td>Adjusting the Toolbar Dynamically</td>
<td>37</td>
</tr>
<tr>
<td>Toolbar Buttons</td>
<td>37</td>
</tr>
<tr>
<td>Toolbar Element Attributes</td>
<td>38</td>
</tr>
<tr>
<td>Toolbar Button Events</td>
<td>39</td>
</tr>
<tr>
<td>IF_FPM_CNR_GAF Interface</td>
<td>40</td>
</tr>
<tr>
<td>Accessing the API for a GAF application:</td>
<td>40</td>
</tr>
<tr>
<td>GAF Specific Parameters</td>
<td>41</td>
</tr>
<tr>
<td>IF_FPM_CNR_OIF Interface</td>
<td>43</td>
</tr>
<tr>
<td>Accessing the API for an OIF application:</td>
<td>43</td>
</tr>
<tr>
<td>OIF Specific Parameters</td>
<td>44</td>
</tr>
<tr>
<td>FPM Complete Preview</td>
<td>45</td>
</tr>
<tr>
<td>FPM Identification Region (IDR)</td>
<td>46</td>
</tr>
<tr>
<td>Adjusting the IDR Dynamically</td>
<td>46</td>
</tr>
<tr>
<td>Adding a Link to the FPM Configuration Editor in the IDR</td>
<td>46</td>
</tr>
<tr>
<td>IF_FPM_IDR Interface</td>
<td>46</td>
</tr>
<tr>
<td>Providing a Link to the FPM Configuration Editor in the IDR</td>
<td>48</td>
</tr>
<tr>
<td>Quick Help</td>
<td>49</td>
</tr>
<tr>
<td>Creating Quick Help</td>
<td>49</td>
</tr>
<tr>
<td>Procedure</td>
<td>50</td>
</tr>
<tr>
<td>Variants</td>
<td>51</td>
</tr>
<tr>
<td>Configuring Variant Selection</td>
<td>51</td>
</tr>
<tr>
<td>Initial Screen</td>
<td>52</td>
</tr>
<tr>
<td>Adding an Initial Screen</td>
<td>52</td>
</tr>
<tr>
<td>Skipping the Initial Screen</td>
<td>53</td>
</tr>
<tr>
<td>Confirmation Screen</td>
<td>54</td>
</tr>
<tr>
<td>Confirmation Screen in OIF Instances</td>
<td>54</td>
</tr>
<tr>
<td>Confirmation Screen in GAF Instances</td>
<td>54</td>
</tr>
<tr>
<td>Adding and Configuring the Confirmation Screen</td>
<td>54</td>
</tr>
<tr>
<td>FPM Event Loop</td>
<td>55</td>
</tr>
<tr>
<td>Raising Standard Events</td>
<td>55</td>
</tr>
<tr>
<td>Triggering the FPM Event Loop</td>
<td>55</td>
</tr>
<tr>
<td>Triggering Application-Specific Events</td>
<td>56</td>
</tr>
<tr>
<td>Reacting to Framework Events</td>
<td>56</td>
</tr>
<tr>
<td>Key Web Dynpro Methods</td>
<td>57</td>
</tr>
<tr>
<td>Different Categories of Web Dynpro Interfaces</td>
<td>58</td>
</tr>
<tr>
<td>Overview Page Floorplan (OVP)</td>
<td>58</td>
</tr>
<tr>
<td>Structure of an OVP</td>
<td>58</td>
</tr>
<tr>
<td>Content Areas</td>
<td>58</td>
</tr>
<tr>
<td>Sections</td>
<td>59</td>
</tr>
<tr>
<td>Personalization</td>
<td>59</td>
</tr>
<tr>
<td>Personalization Editor</td>
<td>60</td>
</tr>
<tr>
<td>UIBB Toolbar Buttons</td>
<td>60</td>
</tr>
</tbody>
</table>
OVP-Related FPM Events for Navigation ................................................................. 61
Dynamic Changes at Runtime .................................................................................. 61
   OVP CNR API ........................................................................................................ 61
   Application Controller API ................................................................................. 62
Setting a Default ALV View for a Freestyle UIBB .................................................... 63
Design Time Settings in the FPM Configuration Editor ........................................... 63
Rendering the ALV Views during Runtime .............................................................. 63
FPM Dialog Component .......................................................................................... 64
   Structure of an FPM Dialog Box ............................................................................ 64
   FPM Dialog Box Features ..................................................................................... 65
Creating and Configuring an FPM Dialog Box ......................................................... 66
Opening and Closing FPM Dialog Boxes ................................................................. 68
Event Processing in Dialog Boxes .......................................................................... 68
   The MV_IS_DIALOG_MODE Attribute ................................................................. 68
Sample Coding to Call A Dialog Box ...................................................................... 68
   Opening a Dialog Box using Direct API .............................................................. 68
   Opening a Dialog Box by Raising an FPM Event ................................................ 69
Message Manager for FPM Dialog Boxes ............................................................... 69
Error Page of an FPM Dialog Box .......................................................................... 70
FAQs on FPM Dialog Boxes .................................................................................... 70
Generic User Interface Building Block (GUIBB) ...................................................... 71
   Feeder Classes ...................................................................................................... 72
      Structure ............................................................................................................ 72
      Features ............................................................................................................. 72
Form Component (GUIBB FORM) ......................................................................... 72
   Structure .............................................................................................................. 72
   IF_FPM_GUIBB_FORM Interface ........................................................................ 73
   Form Editor for Floorplan Manager ................................................................. 77
   Using the CHECKBOX_GROUP Display Type in a Form .................................... 78
   Group Layout in a Form ...................................................................................... 79
   Add Form ............................................................................................................ 80
List Component (GUIBB LIST) .............................................................................. 82
   Structure ............................................................................................................ 82
   IF_FPM_GUIBB_LIST Interface ........................................................................ 82
   IF_FPM_GUIBB_LIST_PAGING Interface ........................................................ 86
   List Editor for Floorplan Manager .................................................................... 88
   Add List ............................................................................................................. 89
   Additional Information on the List Component ................................................ 91
   FPM Events and the List Component ............................................................... 92
   Rendering GUIBB List as ALV ......................................................................... 92
Hierarchical List Component (GUIBB TREE) ............................................................ 94
Structure ....................................................................................................................... 94
IF_FPM_GUIBB_TREE Interface ................................................................................ 95
Hierarchical List Editor for Floorplan Manager ....................................................... 100
Add Tree..................................................................................................................... 102
Additional Information on the Hierarchical List Component ....................................... 103
FPM Events and the Hierarchical List Component ...................................................... 104

Search Component (GUIBB SEARCH).......................................................................... 105
Structure ..................................................................................................................... 105
IF_FPM_GUIBB_SEARCH Interface ........................................................................... 106
Search Component Editor for Floorplan Manager ....................................................... 109
Add Search .................................................................................................................. 109

Launchpad Component (GUIBB LAUNCHPAD)....................................................... 111
Structure ..................................................................................................................... 111
IF_FPM_GUIBB_LAUNCHPAD Interface ................................................................ 114

Tabbed Component (GUIBB TABBED COMPONENT) ........................................... 116
Structure ..................................................................................................................... 116
Tabbed Component Editor for Floorplan Manager ...................................................... 116
Add Tabbed Component ............................................................................................. 117
Changing the Tabbed Component Dynamically at Runtime ........................................ 119

Drag-and-Drop ................................................................................................................. 120
Enabling Drag-and-Drop ............................................................................................. 120
Configuring Drag-and-Drop ........................................................................................ 121
Events and Event Parameters ...................................................................................... 122
Class, Methods and Parameters of Drag-and-Drop .................................................... 122
Event Processing during Drag-and-Drop ...................................................................... 123
Handling Drop in UIBBs .............................................................................................. 124

Navigation with Launchpads ........................................................................................... 124
Launchpads ................................................................................................................... 125
Creating a Launchpad with Applications .................................................................... 127
Including a Launchpad in the User Interface .............................................................. 128
Working in the Navigation Customizing ...................................................................... 128
Source Parameters and Parameter Mapping .............................................................. 129
Copying an entire Launchpad ...................................................................................... 131
Copying Applications from one Launchpad to Another ............................................. 131
Performing Searches in Launchpads of a Client ......................................................... 131
Re-displaying a SAP-Delivered Launchpad ................................................................. 132
Replacement of Switch BC-Sets .................................................................................. 132
Switching on a Whole Launchpad ............................................................................... 132
Switching Applications On and Off inside a Launchpad ............................................. 132
Replacing Existing Applications inside a Launchpad ................................................ 132
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping of Business Objects and Operations</td>
<td>132</td>
</tr>
<tr>
<td>Replacing Object-Based Navigation</td>
<td>132</td>
</tr>
<tr>
<td>Transporting a Launchpad</td>
<td>133</td>
</tr>
<tr>
<td>IF_FPM_NAVIGATION API</td>
<td>133</td>
</tr>
<tr>
<td>Integration: Navigation in the Event Loop</td>
<td>136</td>
</tr>
<tr>
<td>IF_FPM_NAVIGATE_TO API</td>
<td>137</td>
</tr>
<tr>
<td>Restarting a WD ABAP Application</td>
<td>138</td>
</tr>
<tr>
<td>Extracting Launchpad Content and Launch Service</td>
<td>138</td>
</tr>
<tr>
<td>Suspend and Resume</td>
<td>139</td>
</tr>
<tr>
<td>Suspending via Static Launchpad Customizing for URL Application Category</td>
<td>140</td>
</tr>
<tr>
<td>Suspending via Static Launchpad Customizing for Web Dynpro ABAP or Web Dynpro Java Application</td>
<td>140</td>
</tr>
<tr>
<td>Suspending via Launchpad API</td>
<td>140</td>
</tr>
<tr>
<td>Resuming a Suspended Application</td>
<td>141</td>
</tr>
<tr>
<td>Handling Dialog Boxes</td>
<td>141</td>
</tr>
<tr>
<td>Triggering a Data-Loss Dialog Box in the FPM Event Loop</td>
<td>142</td>
</tr>
<tr>
<td>Handling Application-Specific Dialog Boxes</td>
<td>142</td>
</tr>
<tr>
<td>Deferring Current Event Processing</td>
<td>142</td>
</tr>
<tr>
<td>Registering a Dialog Box</td>
<td>143</td>
</tr>
<tr>
<td>Resuming the Event</td>
<td>144</td>
</tr>
<tr>
<td>IF_FPM_WORK_PROTECTION Interface</td>
<td>145</td>
</tr>
<tr>
<td>FPM Message Management</td>
<td>146</td>
</tr>
<tr>
<td>Using the FPM Message Manager</td>
<td>147</td>
</tr>
<tr>
<td>IF_FPM_MESSAGE_MANAGER Interface</td>
<td>148</td>
</tr>
<tr>
<td>Methods for Reporting Messages</td>
<td>148</td>
</tr>
<tr>
<td>Mandatory Parameters</td>
<td>154</td>
</tr>
<tr>
<td>Methods for Raising Exception Messages</td>
<td>155</td>
</tr>
<tr>
<td>Method for Clearing Messages</td>
<td>156</td>
</tr>
<tr>
<td>Handling of FPM Message Manager in Non-FPM Dialog Boxes</td>
<td>157</td>
</tr>
<tr>
<td>FPM Message Manager FAQ</td>
<td>157</td>
</tr>
<tr>
<td>Message Mapper</td>
<td>158</td>
</tr>
<tr>
<td>Enabling Message Mapper</td>
<td>158</td>
</tr>
<tr>
<td>Message Mapping Fields</td>
<td>158</td>
</tr>
<tr>
<td>Message Context</td>
<td>158</td>
</tr>
<tr>
<td>Message Categories</td>
<td>159</td>
</tr>
<tr>
<td>Message Namespace</td>
<td>159</td>
</tr>
<tr>
<td>Message Source</td>
<td>159</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Generalization</td>
<td>159</td>
</tr>
<tr>
<td>Changing Message Types</td>
<td>162</td>
</tr>
<tr>
<td>Hiding Messages</td>
<td>162</td>
</tr>
<tr>
<td>Hiding Messages and Generalization</td>
<td>162</td>
</tr>
<tr>
<td>Logging Messages</td>
<td>162</td>
</tr>
<tr>
<td>Generalization</td>
<td>163</td>
</tr>
<tr>
<td>Mapping Message Variables</td>
<td>164</td>
</tr>
<tr>
<td>API Changes for Message Mapping</td>
<td>165</td>
</tr>
<tr>
<td>Customizing Tables for Message Mapper</td>
<td>165</td>
</tr>
<tr>
<td>Maintenance Views for Message Mapper</td>
<td>166</td>
</tr>
<tr>
<td>FPM Error Page</td>
<td>167</td>
</tr>
<tr>
<td>Structure</td>
<td>167</td>
</tr>
<tr>
<td>Features</td>
<td>167</td>
</tr>
<tr>
<td>Handling of Transactions</td>
<td>169</td>
</tr>
<tr>
<td>Using the Transaction Interface</td>
<td>170</td>
</tr>
<tr>
<td>Transaction Interface FAQ</td>
<td>170</td>
</tr>
<tr>
<td>IF_FPM_TRANSACTION Interface</td>
<td>170</td>
</tr>
<tr>
<td>Resource Management</td>
<td>173</td>
</tr>
<tr>
<td>Releasing a Component</td>
<td>173</td>
</tr>
<tr>
<td>Settings for Transient Behaviour</td>
<td>175</td>
</tr>
<tr>
<td>Setting the Transient Flag</td>
<td>176</td>
</tr>
<tr>
<td>Using IF_FPM_RESOURCE_MANAGER to Veto Release Decision</td>
<td>176</td>
</tr>
<tr>
<td>Using an FPM Application Controller</td>
<td>177</td>
</tr>
<tr>
<td>Implementing the Application Controller</td>
<td>177</td>
</tr>
<tr>
<td>IF_FPM_APP_CONTROLLER Interface</td>
<td>178</td>
</tr>
<tr>
<td>Using an Application-Specific Configuration Controller</td>
<td>178</td>
</tr>
<tr>
<td>Implementing an AppCC Component</td>
<td>178</td>
</tr>
<tr>
<td>Methods</td>
<td>179</td>
</tr>
<tr>
<td>Features</td>
<td>179</td>
</tr>
<tr>
<td>Implementing an AppCC Class</td>
<td>181</td>
</tr>
<tr>
<td>Sharing Data between UIBBs from Different Components</td>
<td>182</td>
</tr>
<tr>
<td>Using a Shared Data Component</td>
<td>182</td>
</tr>
<tr>
<td>Other Options for Sharing Data</td>
<td>183</td>
</tr>
<tr>
<td>Determining Navigation State Information at Runtime</td>
<td>183</td>
</tr>
<tr>
<td>Embedding an FPM Application</td>
<td>184</td>
</tr>
</tbody>
</table>
Constraints .................................................................................................................... 185
Embedded Analytics in Floorplan Manager ................................................................. 185
List Report on BEx Query Application ......................................................................... 186
Features ....................................................................................................................... 186
Creating Analytical and Planning Applications ............................................................ 186
Examples ...................................................................................................................... 186
Procedure ..................................................................................................................... 187
Embedding Analytics and Planning into Transactional Applications ......................... 190
Analytics List Component for Floorplan Manager ....................................................... 190
Adding an Analytics List .............................................................................................. 191
Analytics List Editor for Floorplan Manager ............................................................... 192
List Settings ................................................................................................................ 192
Toolbar Elements ........................................................................................................ 194
Search Component with Analytics Feeder Class ......................................................... 194
Adding an Analytics Search Component ...................................................................... 195
Configuring an Analytics Search Component .............................................................. 195
Procedure .................................................................................................................... 195
Configuring a Search Component ............................................................................... 195
Configuring a Search Component for Analytics ......................................................... 195
Application-Specific Analytics UIBBs ......................................................................... 196
Analytical Application Programming Interface (API) .................................................. 196
FPM Event Loop for Analytics and Planning .............................................................. 197
Floorplan Manager

Floorplan Manager (FPM) is a Web Dynpro ABAP application that provides a framework for developing new Web Dynpro ABAP application interfaces consistent with SAP UI guidelines. FPM currently supports you in creating and configuring user interfaces with the following floorplans:

- Object Instance Floorplan (OIF)
- Overview Page Floorplan (OVP)
- Guided Activity Floorplan (GAF)
- Quick Activity Floorplan (QAF)

The following floorplan areas can be configured using the FPM configuration editor:

- Identification Region (IDR)
- Message Region (MR)
- Context Navigation Region (CNR)
- Roadmap Element

Floorplan content areas must also be UI guideline compliant and FPM provides pre-defined UI building blocks (UIBBs) to support you in creating and configuring application-specific views (freestyle areas). The common UI patterns Form, List, Hierarchical List and Tabbed Area can be configured using the FPM configuration editor.

FPM includes APIs for common functions such as navigation, data-loss handling, messaging, and personalization.

FPM allows for modification-free customer adaptations.

System Requirements

This document outlines the features of Floorplan Manager as of release SAP NW 7.0 Enhancement Package 2 and SAP NW 7.1 Enhancement Package 2. Where it is necessary the system requirement is mentioned at feature level.

Getting Started

This section provides you with an overview of an FPM application and the steps required by you to create a simple Hello World example application.

Once you have created your application, you are introduced to the FPM Configuration Editor, which allows you to edit your application and to configure it at design time.

The FPM event loop and it various activities are explained to you, and finally you are presented with time-saving design templates, allowing you to create guideline compliant user-interfaces.
Assumptions

Knowledge of ABAP OO and Web Dynpro for ABAP is assumed.

User Interface Building Blocks

From an FPM perspective, UIBBs are the interface views (Web Dynpro ABAP windows) that are provided by the external application and not by FPM itself.

In order that the FPM framework recognizes a UIBB, the Web Dynpro component that provides the UIBB must implement the `IF_FPM_UI_BUILDING_BLOCK` Web Dynpro interface. The `IF_FPM_UI_BUILDING_BLOCK` interface ensures that the Web Dynpro application can take part in the FPM event loop.

For more information, see `IF_FPM_BUILDING_BLOCK` Interface.

**IF_FPM_UI_BUILDING_BLOCK** Interface

This Web Dynpro interface ensures that a Web Dynpro application and its UIBBs can take part in the FPM Event Loop.

**Methods**

The methods of this interface are described in the following table:

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
</table>
| FLUSH                | This is the first method called after the FPM event loop has been started.  
  In this method, the UIBB needs to transport all modified data from the views to other components the UIBB wants to communicate with later on.  
  Normally this data transport is done automatically using Web Dynpro context mapping. Therefore, you will only need to do a specific implementation of this method if you are not using these automatic mechanisms. |
| NEEDS_CONFIRMATION   | With this method, the UIBB requests that the subsequent event processing is stopped and asks the user for confirmation by way of a dialog box. Depending on the action the user takes in the dialog box, the event loop is continued or cancelled. For more details, refer to chapter Triggering a Data Loss Dialog Box. |
| PROCESS_EVENT        | Within this method the UIBB completes the following tasks:  
  - Checks for local consistency (validation, missing data, etc).  
  - Perform the actual event processing. |
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The local check is needed to inform the user of potential input errors as soon as possible. In accordance with UX guidelines, checks are to be performed continually (as long as they are not too performance-intensive). For example, when switching from one view to another view in an OIF application, the view (UIBB) which is moved away from must check for local consistency. However, this does not exempt the application from performing a complete check (including performance critical checks) before saving. This must be handled in the method IF_FPM_TRANSACTION_CHECK_BEFORE_SAVE. Besides the consistency check this method contains the actual processing of the event. For this, the current event can be identified through the attributes MV_EVENT_ID and MO_EVENT_DATA on the passed on event instance io_event. Depending on whether the event is processed successfully or not, the exporting parameter EV_RETURN must be filled with either IF_FPM_CONSTANTS<del>GC_EVENT_RESULT-OK or IF_FPM_CONSTANTS</del>GC_EVENT_RESULT-FAILED. A typical implementation of PROCESS_EVENT is shown below:</td>
</tr>
</tbody>
</table>
|                    | 1. IF io_event->mv_event_is_validating = abap_true.  
2. Do local checks and report messages if needed  
3. ENDIF  
4. CASE io_event->mv_event_id.  
5. WHEN XYZ  
6. Handle event and fill EV_RETURN accordingly with a value from IF_FPM_CONSTANTS~GC_EVENT_RESULT  
7. ENDCASE.  
If the event processing requires further user interaction (for example asking for further data in a dialog box), the event processing can be deferred by returning EV_RETURN = IF_FPM_CONSTANTS~GC_EVENT_RESULT-DEFER. |
<p>| AFTER_FAILED_EVENT | This method is called by the FPM if an event could not be processed successfully. In this case the UIBB needs to ensure that its UI reverts to the state before the user interaction occurred. Selecting an option in a ‘Lead’ field in a table triggers |</p>
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>the display of the details of a new line in another UIBB. The event could fail if the UIBB for the details contains unsaved data for the previously selected table line. As the detail form still contains the details of the original table line (after the failed event), the Lead selection must be reverted to the original table line too.</td>
</tr>
<tr>
<td></td>
<td>If the <code>PROCESS_EVENT</code> method of the current UIBB has been processed successfully, but the event processing failed due to a problem in another UIBB, the actual event processing needs to be reverted as well. The parameter <code>IV_REVERT</code> indicates this situation.</td>
</tr>
<tr>
<td>PROCESS_BEFORE_OUTPUT</td>
<td>The last method to be called on the UIBB is the <code>PROCESS_BEFORE_OUTPUT</code>. The data to be displayed is read from the model.</td>
</tr>
</tbody>
</table>

**Creating a Simple FPM Application**

Here you create a simple Hello World FPM application based on either the OIF or GAF. The OIF application will contain 2 tabs, each containing a single subview tab; the GAF application will contain 2 road steps.

This process is performed in the **Web Dynpro ABAP Workbench**.

You construct an FPM application by completing the following steps:

1. [Create a Web Dynpro Component](#) with the required UIBBs and implement the Web Dynpro interface `IF_FPM_UI_BUILDING_BLOCK`.
2. [Create a Web Dynpro Application](#) and specify parameters according to which floorplan instance you are using.
3. Using the [FPM Configuration Editor](#), create a configuration for the application.
4. Test your application.

An FPM application is composed of a number of different Web Dynpro components (most of which are instantiated dynamically at runtime). However, the following two components are always present:

- a floorplan-specific component (`FPM_GAF_COMPONENT` or `FPM_OIF_COMPONENT`)
- a component for the Header Area (`FPM_IDR_COMPONENT`)

In simple terms, the configuration of an FPM application is the configuration of these two components.

**Creating a Web Dynpro Component**

1. Open the **Web Dynpro ABAP Workbench**.
2. In the *Object Navigator*, right-click the Web Dynpro node and choose *Create Web Dynpro Component (Interface)*.

3. In the *Web Dynpro: Component/Create Interface* dialog box, enter a name, description and window name (the window name must be different from the View name).

4. Save your entry.

5. In the *Attributes* section view of the *Create Object Entry Directory* dialog box, enter the relevant Package.


7. Choose the *Implemented Interfaces* tab.

8. In the first row of the Name column, enter the FPM interface `IF_FPM_UI_BUILDING_BLOCK` and save your entry.

9. In the *Action* column, choose *Reimplement*. The icon in the *Implementation State* column indicates that your component is completely implemented.

10. Choose *Activate*.

11. In the *Activation* dialog box, select all associated, inactive components and choose *OK*.

### Adding Views to your Web Dynpro Component

When you create a component, Web Dynpro automatically creates and assigns a Window and a View to it. You may add further Windows and Views. It is recommended that you add only one View to one Window.

1. In the Object Navigator, find your new Web Dynpro component and expand its node.
   1. Expand the Views node and double-click the existing View. The View appears in the preview.
   2. In the *Layout* tab, click once on the Caption element. A blue square appears in the preview, ready to display your text.
   3. In the *Properties Section*, enter *Hello* in the *Text* property. Choose *Save* and your text appears in the preview.

2. Choose *Activate*.
   1. In the Activation dialog box, select all associated, inactive components and choose *OK*.

3. Add a second View:
   1. Right-click the *View* node and choose *Create*. Give your View a name and choose *OK*.
   2. Add a caption element and enter the text *Welcome to the world of FPM*.

4. Add this view to a new Window (which you create now):
   1. Right-click the *Windows* node and choose *Create*.
   2. In the *Web Dynpro: Create Window* dialog box, enter a Window name and choose *OK*.
   3. The preview automatically displays the *Window* tab. In the *Window Structure* column, there is a node with your new Window’s name.
   4. Drag your new View from the *Object Navigator* onto this node so that it is included in the *Window* structure (expand the node to see the new listed below it).
   5. Save and activate your new Window.
You have now created a Web Dynpro Component, implemented the required IF_FPM_UI_BUILDING_BLOCK interface and configured two views (in two separate windows) for your component.

Creating a Web Dynpro Application

Prerequisites

You have already created a Web Dynpro component with two views.

Procedure

1. In the Object Navigator, right-click your Web Dynpro Component and choose Create → Web Dynpro Application.

In the Create Web Dynpro Application dialog box, enter a name for your application and choose OK. Your new Web Dynpro Application appears in the preview.

2. Enter the following information to create either an OIF or a GAF application:
   - Component: FPM_OIF_COMPONENT / FPM_GAF_COMPONENT
   - Interface View: FPM_WINDOW
   - Plug Name: Default

3. Save your entries.

   In the Create Object Directory Entry dialog box, enter the relevant Package and choose OK.

Result

You have created a Web Dynpro application based on an OIF or GAF floorplan instance.

If you want to add parameters to your application, see Using Application Parameters.

Using Application Parameters

Application parameters are defined at Web Dynpro Application level.

To define your application parameters, proceed as follows:

1. In the Web Dynpro Object Navigator, double-click your Web Dynpro application.

2. Choose Parameters. You can add arbitrary parameters as application-specific attributes to your Web Dynpro application. During runtime, these parameters are exposed via IF_FPM->MO_APP_PARAMETER. MO_APP_PARAMETER stores an instance of IF_FPM_PARAMETER. With this interface you are able to retrieve the parameters.
Note that there is no concept of mandatory or optional parameters. For security reasons, you must never trust parameters passed by a different application. Always complete a proper validation before you use application parameters.

There are other FPM-specific parameters which you can add to your application. These are detailed in the table below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPM_SHOW_MESSAGE_LOG</td>
<td>You can turn on a log history of the messages for a particular application. When the message log is turned on, all the previously reported messages are displayed.</td>
</tr>
<tr>
<td>FPM_MAXIMUM_MESSAGE_SIZE</td>
<td>When a message is created in the application, the message area displays as many messages as possible. As soon as the visible number of messages in the message area exceeds the configured message size, a scroll bar will appear in the message area, allowing the user to read all messages. The maximum size of the message is set via configuration.</td>
</tr>
<tr>
<td>FPM_HIDE_CLOSE</td>
<td>With this parameter, you can hide the Close button on the FPM toolbar for your application.</td>
</tr>
</tbody>
</table>

Creating an Application Configuration with the FPM Configuration Editor

Prerequisites

You have already created a Web Dynpro component with two views and have created a Web Dynpro application implementing the FPM_OIF_COMPONENT or FPM_GAF_COMPONENT interface.

Procedure

1. In the Object Navigator, right-click your new Web Dynpro Application and choose Create/Change Configuration. The FPM Configuration Editor (Editor for the Web Dynpro ABAP Application Configuration) opens in a browser window.
2. Enter a name for your application’s configuration in the Configuration ID field. Note that configuration names are global; you may not use the same configuration name for different applications.
3. Choose Create. In the Create Configuration dialog box, enter the relevant Package and choose OK.
4. The application configuration window displays your new configuration. Within your configuration are the following two components:
   - FPM_OIF_COMPONENT (or FPM_GAF_COMPONENT)
   - FPM_IDR_COMPONENT
5. You will create configurations for both of these components. In the configuration column, enter names for both components and choose Save. A message appears to inform you that the components are saved, but that the configurations do not actually exist. You will create a configuration for the OIF (or GAF) component now.
6. Select the row containing your FPM_OIF_COMPONENT (or FPM_GAF_COMPONENT) and choose Go To Component Configuration.
1. Choose Create to configure the component.
2. In the Create Configuration dialog box, choose the relevant Package and choose OK.

7. The FPM Configuration Editor displays the Component Configuration window for your OIF (or GAF) component. The FPM Component Configuration window is divided into the following areas:

   - Navigation hierarchy: shows the screen elements in your application which you can configure
   - Preview: displays the element you have selected in the hierarchy and allows you to change the attributes of the element
   - Action area: allows you to add various elements to your individual screens (for example, toolbar buttons, main views or UIBBs)

For a simple application, you require only one variant, one main view and one subview. The FPM Configuration Editor automatically provides these entities (with default IDs and names).

Complete the configuration by performing the following steps below.

**Configuring the Component and IDR Configurations**

**Configuring the FPM_OIF_COMPONENT**

1. The preview of the Component Configuration window displays 1 main view containing 1 subview.
2. To add the second main view tab, choose Add Main View in the action area.
3. In the hierarchy, expand the two Main View nodes and the two Subview nodes. Note the two UIBB elements, one for each subview. Choose the UIBB element belonging to the first subview to display its attributes in the preview.
4. Set these attributes to your first window (with accompanying view) of your Web Dynpro component (containing the text 'Hello').
   1. Enter the Component name (use the input help and search function to find your component).
   2. Enter the View (once you have entered the component name, the View input help displays the list of views for that component).
5. In the hierarchy, choose the other UIBB element to display its attributes. Set these attributes to your second window (with accompanying view) of your Web Dynpro component (containing the text 'Welcome to the world of FPM').
6. Choose Save. A confirmation (or error) message appears near the top of the screen.

You have now added your component views to the application and are ready to configure the IDR Component of your application’s configuration.

**Configuring the FPM_GAF_COMPONENT**

1. The preview of the Component Configuration window displays 1 main step containing 1 UIBB. There are also two buttons, Previous and Next, which the FPM automatically displays in the toolbar for you.
2. To add the second step, choose Add Main Step in the action area.
3. In the hierarchy, expand the two Main Step nodes. Note the two UIBBs, one for each step. Choose the first UIBB element to display its attributes in the preview.
4. Set these attributes to your first window (with accompanying view) of your Web Dynpro component (containing the text ‘Hello’).
   1. Enter the Component name (use the input help and search function to find your component).
   2. Enter the View (once you have entered the component name, the View input help displays the list of views for that component).
5. In the hierarchy, choose the other UIBB element to display its attributes. Set these attributes to your second window (with accompanying view) of your Web Dynpro component.
6. Choose Save. A confirmation (or error) message appears near the top of the screen.

You have now added your component views to the application. You are now ready to configure the IDR Component of your application’s configuration.

**Configuring the FPM_IDR_COMPONENT**

Once you have created a configuration for your OIF (or GAF) component, you are then ready to create a configuration for the IDR component.

1. In the action region of the Component Configuration window, choose Configure IDR. The Configuration ID field displays the name you provided in the previous steps for your IDR component configuration.
2. Choose Create. In the Create Configuration dialog box, enter the relevant Package and choose OK. The Component Configuration window displays your IDR Configuration.
3. In the hierarchy, choose IDR Basic. The preview displays the attributes of the IDR Basic. Enter the following data:
   - Application Title
   - Tooltip (optional)
4. Choose Save. If there are error messages, they appear at the top of the window, underneath the window’s title. Note that in an OIF application there is an extra button in the Action Pane, Add IDR Extended. This provides you with the optional Extended Identification Region and its attributes.

**Result**

You have now created your first FPM application configuration. You can now test your FPM application.

**Testing your FPM Application**

**Procedure**

1. Open the Web Dynpro ABAP Workbench.
2. In the Object Navigator, locate your FPM application under the Web Dynpro Applications node.
3. Right-click your application and choose Test. Your application opens.
Note that your own component views (UIBBs) appear in the freestyle Content Area of the FPM application.

**Application Creation Tool (ACT) – Shortcut for Creating Applications**

This one-screen application tool significantly reduces the effort involved in creating a new FPM application. The application creation tool (ACT) is a WD application, provided by FPM, which allows application developers to create FPM applications and the corresponding configurations in all the three available floor plans (OIF/GAF/OVP).

To start the application, choose the `FPM_CFG_APPL_CREATION_TOOL` WD application located in the `APB_FPM_CONF` package. Choose `Test` from the context menu.

To create an FPM application using this tool, complete the following steps:

1. For the application you want to create, enter the following information:
   - Application Name
   - Description (optional)
   - Floorplan

2. Choose `Propose`. Based on the floorplan selected, FPM proposes and displays configuration names for the following components in the `Proposed Configuration Names` table:
   - Application Configuration
   - Floorplan (Component) Configuration
   - Header (IDR) Configuration
   You can edit the component names proposed by FPM and also enter descriptions for all the configurations. Choose `Propose` again if you want to change the name you first selected.

3. By default, all the checkboxes in the `Required` column are ticked; the system will create all the components for an application configuration. Only the checkbox for the IDR component is editable; the checkboxes for the application and floorplan components are read-only. However, for an OVP floorplan there is no header area and all three checkboxes are read-only.

4. Choose `Create` and enter the package and transport details. The system creates the new FPM application and the component configurations.

5. You have now created an FPM application configuration. You have the following options:
   - Choose `Test` to run the new application configuration
   - Choose `Configuration Editor` to edit the component configurations
   - Choose `New` to create a new FPM application
Configuration Editor for Floorplan Manager

You use the Floorplan Manager configuration editor to enhance application user interfaces and fit them to your business needs.

The configuration editor consists of the following work areas:

- **Navigation Path**
  
  This region is on top of the Navigation Region, the Preview and the Action Area. The Navigation Path shows all application configurations or component configurations that have been accessed previously and that are aligned hierarchically above the current configuration. The Navigation Path allows the user to navigate backward to the corresponding application respectively component configurations, whereas the user will find the same UI state (such as work mode, selected element, expanded elements) as it was left before.

- **Navigation Region**
  
  This region is divided into the following sub regions:

  - **Control Area**
    
    In this area, you select which screens you would like to configure for the selected Web Dynpro application. You can choose whether you want to see the preview of the initial screen, the main views of an application variant, or the confirmation screens of the selected application.

    In this region, you can use the *Change* or *Display* buttons to display or configure the application's global settings and variant parameters.

    You can store multiple variants of a selected floorplan for one Floorplan Manager application. A variant gives you an additional level of differentiation within Floorplan Manager. For example, you can use variants to show multiple user roles in the same application at the same time. The individual variants are separated from one another in an initial screen.

  - **Hierarchy**
    
    This region gives you a hierarchical display of the elements you can configure. The elements you can configure depend on the current configuration of the application. The hierarchy shows elements on the screen that you have selected.

- **Preview**
The preview function shows you the user interface of the application. You can use the preview function to navigate within the user interface. However, not every element can be accessed. A selected element is highlighted in color in the hierarchy view and its attributes displayed in the attribute view.

- **Action Area**

  The action area contains links to all the actions you can execute for the selected application user interface. The actions that can be selected depend on the concrete configuration of the application. This means that selection of actions can differ within a configuration.

- **Attribute View**

  When you select a configurable user interface element either in the preview or in the hierarchy, the attributes of these user interface elements are displayed in the attribute view. You can change these attributes here. The attributes you can change depend on the user interface element you selected. You can immediately see any changes made in the preview.

- **Message Area**

  In this area, potential conflicts in the configuration, such as tabs with the same name, are immediately displayed.

Every user interface element is defined and configured using its attributes. Your authorization profiles determine whether you can carry out a configuration or enhancement of user interface elements. The following two authorization profiles are of importance:

- **S_DEVELOP**

  With the authorization profile for ABAP Workbench, you can make any and all changes to a user interface developed with Web Dynpro ABAP.

- **S_WDR_P13N**

  You can use this authorization profile to make changes to a user interface if the S_DEVELOP authorization profile is not assigned to your user. It authorizes you to configure a Web Dynpro application in administrator mode.

For more information on these authorization profiles, see SAP Library for Web Dynpro ABAP under Authorization Checks in Configuration/Personalization.

You can use the configuration editor to perform Web Dynpro built-in configuration as well as component-defined configurations of user interface elements. For more information on Web Dynpro built-in and component-defined configuration, see Fitting Web Dynpro to Your Needs.

You can launch the configuration editor for Floorplan Manager in one of the following ways:
At runtime in administrator mode, from the application user interface using the Adapt Configuration link.

At runtime in expert mode, from the application user interface using the Change Configuration link.

At design time in Web Dynpro Application Configuration in the Object Navigator of the ABAP Workbench.

At design time in Web Dynpro Component Configuration in the Object Navigator of the ABAP Workbench.

**Form Editor for Floorplan Manager**

You use the form editor to adjust a form in an application to your specific business requirements. This is done by configuring form components.

The form editor consists of the following work areas:

- **Preview**
  
  In the preview, all form elements from the current configuration are displayed so as to give you a picture of the layout of the form.

- **Hierarchy**
  
  All form elements (groups, melting groups, and elements) are displayed in the hierarchy.

- **Attribute view**
  
  Attributes of the currently selected form element that can be changed using the form editor are displayed in the attribute view.

- **Action area**
  
  The action area contains links to all the actions you can execute for the form component. The actions that can be selected depend on the concrete configuration of the form. This means that it can differ within a configuration.

The form editor provides you with the following actions:

- Add Group
- Add Melting Group
- Edit Feeder Class
- Edit Parameters
- Configure Toolbar
- Configure Group

The form editor provides you with the following functions for editing a group:

- Change Group Attributes
The group name, group type, and index can be changed.

- Add New Group
- Add Melting Group
- Add Element

You can select a field from the field catalog and determine the label text and display type.

- Delete Group

The form editor provides you with the following functions for editing a melting group:

- Add Group Element

You can select a field from the field catalog. Fields are configured in more detail by changing the group element attributes.

- Change Group Element Attributes

The display type, visibility of the label, label text, and index can be changed. Any other group element attributes that can be changed depend on the display type.

- Delete Group Element

The form editor provides you with the following functions for editing a toolbar:

- Add Button
- Change Button
- Delete Button

The form editor is launched in a separate browser window. You can launch the form editor in change or display mode and save your changes at any time.

The component-defined processing view is pre-set. Make sure that this view is selected before configuring a form component.

The form editor launches from the configuration editor for Floorplan Manager automatically when you launch the configuration of an application-specific view (UIBB) that uses the FPM_FORM_UIBB Web Dynpro component.

**List Editor for Floorplan Manager**

You use the list editor to adjust a list within an application to your specific business requirements. This is done by configuring list components.
The list editor consists of the following work areas:

- **Preview**
  In the preview, the list in the current configuration is displayed so as to give you a picture of the layout of the list.

- **Hierarchy**
  All list elements (columns, toolbar, and parameters) are displayed in the hierarchy as a tree structure.

- **Attribute view**
  Attributes of the currently selected list element that can be changed using the list editor are displayed in the attribute view.

- **Action area**
  The action area contains links to all the actions you can execute for the list component. Which actions can be selected depends on the concrete configuration of the list. This means that the selection of actions can differ within a configuration.

The list editor provides you with the following actions:

- Edit Feeder Class
- Edit Parameters
- Configure Column
- Configure Toolbar

The form editor provides you with the following functions for editing a column:

- Add Column
  You can select a field from the field catalog and determine the column header and display type.

- Delete Column

The list editor is launched in a separate browser window. You can launch the list editor in change or display mode and save your changes at any time.

The component-defined processing view is pre-set. Make sure that this view is selected before configuring a list component.
The list editor launches from the configuration editor for Floorplan Manager automatically when you launch the configuration of an application-specific view (UIBB) that uses the FPM_LIST_UIBB Web Dynpro component.

**Search Component Editor for Floorplan Manager**

You use the search editor to adjust a Search within an application to your specific business requirements. This is done by configuring a Search Component.

The search editor consists of the following work areas:

- **Preview**
  
  In the preview, the search in the current configuration is displayed so as to give you a picture of the search attributes which are shown on the screen and for the result list.

- **Hierarchy**
  
  All search elements (search criteria, result list columns and parameters) are displayed in the hierarchy as a tree structure.

- **Attribute view**
  
  Attributes of the currently selected search element that can be changed using the search component editor are displayed in the attribute view.

- **Action area**
  
  The action area contains links to all the actions you can execute for the search component. Which actions can be selected depends on the concrete configuration of the search. This means that the selection of actions can differ within a configuration.

The search component editor provides you with the following actions:

- Edit Feeder Class
- Edit Parameters
- Edit Search Attributes
- Exclude Operators
- Edit Result List Field Catalog

In the hierarchy you can select the *Search* node on the top level of a search and in the attribute view you can see the feeder class which is used. You can decide how many search lines shall be visible providing a value for the *Number of Search Lines* attribute and you can decide how many rows shall be visible in the result list providing a value for the *Number of Result Rows* attribute. Here you can also define whether a search can be saved or not.

The search component editor provides you with the following functions for editing a search attribute:

- Edit Search Attribute
You can select an attribute from the field catalog and determine the label.

- **Delete Search Attribute**

With the *Exclude Operators* action you can decide at configuration level which operators shall become non-selectable at runtime for every single search attribute.

The search component editor provides you with the following functions for editing a result list attribute:

- **Edit Search Result Attribute**

You can select an attribute from the field catalog and determine the label and the display type.

The search component editor is launched in a separate browser window. You can launch the search component editor in change or display mode and save your changes at any time.

The component-defined processing view is pre-set. Make sure that this view is selected before configuring a search component.

The search component editor launches from the configuration editor for Floorplan Manager automatically when you launch the configuration of an application-specific view (UIBB) that uses the *FPM_SEARCH_UIBB* Web Dynpro component.

**Tabbed Component Editor for Floorplan Manager**

You use this editor to adjust a [tabbed component](#) within an application to your specific business requirements. This is done by configuring the component.

The editor consists of the following work areas:

- **Preview**

In the preview, all application-specific views (UIBBs) from the current configuration are displayed so as to give you a picture of the layout of the tabbed component.

- **Layout**

In this area, you determine whether the tabbed component should be arranged horizontally or vertically.

- **Hierarchy**

All application-specific views (UIBBs) are displayed in the hierarchy as a tree structure.
• Attribute view

Attributes of the currently selected application-specific view (UIBB) that can be changed using the editor are displayed in the attribute view.

• Action area

The action area contains links to all the actions you can execute for the tabbed component.

The editor for a tabbed component provides you with the following actions:

- Add Master Component (technical name: MASTER UIBB)
- Add Tab (technical name: TAB)
- Add Application-Specific View to Tab (technical name: TAB UIBB)

The editor for a tabbed component is launched in a separate browser window. You can launch the editor in change or display mode and save your changes at any time.

The component-defined processing view is pre-set. Make sure that this view is selected before configuring a tabbed component.

The editor for a tabbed component launches from the configuration editor for Floorplan Manager automatically when you launch the configuration of an application-specific view (UIBB) that has the FPM_TABBED_UIBB Web Dynpro component.

Wire Model

The wire model can be used to create running FPM application by pure configuration or at least with minimal coding effort. The runtime interdependencies between UIBBs are defined by configuration entities called “wires” which are based on reusable “connector” classes implementing the dependency semantics. The primary use cases for the wire model are object models with generic access interfaces (for example, ESF, BOPF, or BOL).

A wire controls the runtime interdependencies between two UIBBs; that is, they determine the data content of the target UIBB depending on user interaction changing the “outport” of the source UIBB. Outports can be of type lead selection, selection or collection”. For example, the execution of a search on a Search GUIBB will change its collection outport and may therefore change the data content of a result list displayed in a separate List GUIBB. Similarly, changing the lead selection in a list of sales orders may change the data content of another list displaying the associated sales order items.

In order to be part of a wire model, a UIBB needs to implement a certain Web Dynpro interface which in turn provides a feeder model implementation. The FPM GUIBBs are automatically integrated if their feeder classes implement the feeder model interface.
Application areas or object models define their own namespaces for which their connector classes, feeder model classes can be reused. Moreover, they typically need to provide a transaction handler class which manages transaction events like “save”, “modify” or “check” and global message handling.

Wires are defined on the level of the floorplan configuration. For each model UIBB contained in the floorplan configuration, a source UIBB with specified outport can be defined. Furthermore, a connector class and, potentially, connector parameters must be maintained.

If the floorplan contains composite components (tabbed components), the model UIBBs contained in the tabbed components can also be wired. However, in order to provide better reusability of composite components, it is also possible to define intrinsic wiring for tabbed components. A tabbed component can define a model UIBB as a “wire plug” (this is usually a master UIBB), which serves as an entry point for the wiring of the tabbed component from the enveloping floorplan component. If a wire plug is configured for a tabbed UIBB, only the wire plug UIBB can be wired from outside.

**IF_FPM_UIBB_MODEL Interface**

This Web Dynpro interface needs to be implemented by freestyle UIBBs which shall be sources or targets of wires. You only need to implement the method GET_MODEL_API such that it returns a bound reference to the ABAP OO feeder model interface.

**IF_FPM_FEEDER_MODEL Interface**

The feeder model interface comprises methods which are called by the FPM framework.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_NAMESPACE</td>
<td>Returns the namespace of the underlying application area. Method is called at design time.</td>
</tr>
<tr>
<td>SET_CONNECTOR</td>
<td>Called upon instantiation of a UIBB. It hands over the connector (reference to IF_FPM_CONNECTOR_RUN) which can be accessed for data retrieval at PBO.</td>
</tr>
<tr>
<td>GET_IMPORTED_KEY</td>
<td>Returns a reference to an object key which characterizes the meta data type expected at the import (for example the business object node). Method is called at design time.</td>
</tr>
<tr>
<td>GET_OUTPORTS</td>
<td>Provides a table of outports comprising the object key, the port type an identifier and a descriptive text. Method is called at design time.</td>
</tr>
<tr>
<td>GET_OUTPORT_DATA</td>
<td>Returns an object reference carrying the actual data identifier for a certain port. Method is called at runtime.</td>
</tr>
</tbody>
</table>
The actual object type and the type of data it contains (keys, GUIDs or entity references etc.) is left to the design of the actual namespace. It is however important that there is a consistent handling inside a namespace and that the data identifier allow each feeder model to uniquely identify the runtime data to be accessed.

The IF_PPM_CONNECTOR connector interface comprises an interface, IF_PPM_CONNECTOR_DEF, defining the access by the FPM framework and an interface IF_PPM_CONNECTOR_RUN for runtime access by the application feeder model. The definition interface possesses a static attribute, SV_NAMESPACE, which should be filled with the namespace in the class constructor of a connector implementation (for example in a common superclass).

Methods of the connector interface: framework access part

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_PARAMETER_LIST</td>
<td>Connector classes can be parameterized to flexibly control their runtime behavior. The parameter values are maintained for the wires in the FPM configuration editor. A parameter is defined by a name, its data type and a descriptive text.</td>
</tr>
<tr>
<td>GET_PARAMETER_VALUE_SET</td>
<td>With this method, a connector implementation can provide a value set for each parameter. For example, in an object model a parameter may carry the association name. For a wire between specified UIBBs, the method may provide a list of all associations between the source and target business object node.</td>
</tr>
<tr>
<td>INITIALIZE</td>
<td>With this method the connector is initialized with the parameter values. This method is called by the FPM runtime upon UIBB instantiation.</td>
</tr>
<tr>
<td>SET_INPUT</td>
<td>Receives an object reference carrying the actual data of the connected outport. This method is called before the UIBB’s PBO by the FPM runtime.</td>
</tr>
</tbody>
</table>

The runtime interface contains all the methods which are concurrently called in the request-response cycles at runtime.

Methods of the connector interface: feeder model access part

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_OUTPUT</td>
<td>Returns an object reference carrying the actual data to be displayed by a UIBB. This method can be called by the UIBB at PBO for example in the GET_DATA method of a feeder class.</td>
</tr>
<tr>
<td>CREATE_ENTITY</td>
<td>Creates and returns a data entity which can be arbitrarily typed. This method can be called by an action handler of the</td>
</tr>
</tbody>
</table>
The transaction interface provides methods for handling global and transactional events. In the FPM configuration editor, one transaction handler implementation can be assigned on the level of the wire model.

### Methods of the transaction handler interface

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>START</strong></td>
<td>Receives basic data like the FPM message handler and application parameters. This method is called once at application start.</td>
</tr>
<tr>
<td><strong>AFTER_FLUSH</strong></td>
<td>This method is called after <strong>FLUSH</strong> has been called for all current UIBBs. It can be used to flush buffers.</td>
</tr>
<tr>
<td><strong>AFTER_PROCESS_EVENT</strong></td>
<td>This method is called after <strong>PROCESS_EVENT</strong> has been called for all current UIBBs. It can be used for handling transactional events for example <strong>SAVE</strong> or <strong>CHECK</strong>. Moreover, it can be used to collect messages which here not handled inside UIBBs and to forward them to the FPM message handler.</td>
</tr>
<tr>
<td><strong>AFTER_PROCESS_BEFORE_OUTPUT</strong></td>
<td>This method is called after <strong>PBO</strong> has been called for all current UIBBs. It can be used to collect messages at the latest possible point in time before screen output.</td>
</tr>
<tr>
<td><strong>AFTER_NEEDS_CONFIRMATION</strong></td>
<td>This method is called after <strong>NEEDS_CONFIRMATION</strong> has been called for all UIBBs. It can be used to analyze and add confirmation requests.</td>
</tr>
<tr>
<td><strong>IS_DIRTY</strong></td>
<td>This method can be used to indicate a dirty state for the work protection mode.</td>
</tr>
</tbody>
</table>

**FPM on BOL**

For the CRM Business Object Layer (BOL), there is a complete implementation of feeder classes for the GUIBBs Form, List and Search and connector classes for trivial connections (“identity connector” used, for example for a master-detail pattern) and BOL relations as well.
as application entry via URL parameters with implicit query execution. There is also a BOL transaction handler class and assistance base classes for freestyle UIBBs.¹

**Creating a GUIBB on BOL**

**Prerequisites**

There is already a BOL component implementing the business logic.

**Procedure**

1. Start the configuration editor for the GUIBB component (FPM\_FORM\_UIBB, FPM\_LIST\_UIBB or FPM\_SEARCH\_UIBB).
2. Choose feeder class CL\_GUIBB\_BOL\_FORM for a form, CL\_GUIBB\_BOL\_QUERY for a query form, CL\_GUIBB\_BOL\_LIST for a list or CL\_GUIBB\_BOL\_DQUERY for a search GUIBB.
3. Maintain the feeder parameters. You have to specify the BOL component and the BOL object which specifies the object name, the query or the dynamic query.
4. After confirming the parameters, you will already retrieve the feeders default configuration. You may adjust it according to your needs.

You can also create your own feeder classes inheriting form the above BOL feeder classes. This way you can adjust their behavior according to your needs with minimal coding effort.

**Creating an FPM Application on BOL**

**Prerequisites**

You have created UIBBs on BOL.

**Procedure**

1. Create a configuration for the floorplan component (FPM\_OIF\_COMPONENT, FPM\_GAF\_COMPONENT or FPM\_OVP\_COMPONENT).
2. Assemble the UIBBs on BOL according to your needs.
3. Navigate to the “Wire Model” node in the hierarchy (below the “Variant” node for OIF and GAF).
4. Choose the “BOL Transaction Handler (CL\_FPM\_BOL\_TRANSACTION)” as transaction handler.
5. Use the “Add wire” button in the action region to create new wires or navigate to the “Wire” node in the hierarchy to maintain existing wires.

¹ The objects belong to package APB\_FPM\_BOL\_CORE in the software component WEBCUIF. Sample applications can be found in package APB\_FPM\_BOL\_TEST.
6. Maintain the UIBB instance key of the (target) UIBB which shall receive the data from another UIBB. (You may use the value help for any field of the instance key.)
7. Maintain the UIBB instance key of the source UIBB. (You may use the value help for any field of the instance key.)
8. Maintain the output of the source UIBB. (You may use the value help.)
9. Maintain the connector class. (You may use the value help.)
10. Maintain the connector parameters if the connector defines parameters. If there is only a unique value, it is automatically filled (for example if the relation name is unique between to BOL objects).
11. Repeat step 5 to 10 for all the dependencies of your UI.
12. Save and configuration.
13. Create or reuse a Web Dynpro application and create an application configuration referencing your new floorplan component configuration.

You can also create your own feeder classes inheriting from the above BOL feeder classes. Similarly you can create your own connector classes inheriting from the above BOL connector classes. This way you can adjust their behavior according to your needs with minimal coding effort.

Design Time with the FPM Configuration Editor

Within the FPM Configuration Editor, the Component Configuration windows (for your OIF, GAF, OVP and IDR components) help you to design the layout of your application.

Within the component configuration windows, you can perform the following tasks:

- Add extra steps or views (depending on your floorplan instance), including substeps and sub views
- Configure the toolbar with predefined buttons and navigation menus and attach events to these elements
- Attach your UIBBs to the relevant steps or views (or attach the FPM predefined GUIBBs)
- Define the layout for a step or view.

The standard layout is to have each UIBB underneath each other. It is now possible to arrange the UIBBs in a Two-Column layout and in a Row layout. If a UIBB is added in the configuration editor it is by default put in the first row or column within the new layout feature. It can be easily put in the other column or rows by changing the Row respectively Column attribute and the Index attribute.

Within the actions area there is a button choice Add UIBB. It has the entries for adding a form (FORM UIBB), a list (LIST UIBB), a tabbed component (Tabbed UIBB), and a search (SEARCH UIBB). If you select one of those entries the Component and Window fields are prefilled. It is only necessary to add the configuration ID.
• Configure Quick Help for your application
• Configure an initial screen, a confirmation screen and extra variants for your application
• Change the global settings for your application and set variant parameters
• Activate the preview of UIBBs

Above the preview area, a new button *Show UIBB Preview* is added. If the button is active you have the possibility to see the application how it looks like at runtime.

The interface view of your application is the smallest unit of application UI that can be configured in the FPM. By assigning the interface view as a UIBB you are in effect composing how your application content area will look when the application runs within FPM.

**Floorplan Instances in the FPM Configuration Editor**

What you see in the hierarchy of the *FPM Configuration Editor* depends on the type of Floorplan instance you are using in your application.

**OIF Instance**

In an OIF application, FPM displays your UIBBs in multiple tabs within the *Content Area*. The hierarchy displays the following types of views:

- *Main View*: These represent a single tab within the *Content Area* of your application. Attributes allow you to name and identify the individual tabs. Each Main View contains one or more sub-views.

  To add more tabs, choose *Add Main View*.

- *Sub-View*: You add your UIBBs to the sub views. An FPM application must have at least one UIBB for each sub view. The *FPM Configuration Editor* automatically provides this, but you can add your own predefined UIBBs from your application. These UIBBs will be rendered one beneath the other. As well as containing UIBBs, sub views enable you to further divide your tabs for more complex applications. You can configure headings for both main- and sub-views. However, if you create only one main view with only one subview, then no tabs are displayed at all.

  Adding UIBBs to a Sub View:

  1. In the hierarchy, select the sub view in which you want to add your UIBB(s) and choose *Add UIBB*.
  2. In the attribute view, enter the following details:
     - Component (the name of the Web Dynpro component implementing the FPM interface IF_FPM_UI_BUILDING_BLOCK)
     - View (the name of the above Web Dynpro component’s Window containing the UIBB)

**GAF Instance**
In a GAF application, FPM displays your UIBBs as individual steps in the overall Roadmap. For GAF applications, the hierarchy displays the following types of steps:

- **Main Step**: Each main step in the hierarchy represents one roadmap step. An FPM application must have at least one UIBB for each main step. The *FPM Configuration Editor* automatically provides this but you can add your own predefined UIBBs from your application. Attributes allow you to name and identify the individual main steps.

- **Sub Step**: A substep is a step that appears between two main steps. Attributes allow you to name and identify the individual substeps. Like a main step, substeps must have at least one UIBB. You add UIBBS to a substep in the same way you add them to a sub view.

Substeps are not visible at startup, but all main steps that are a possible starting point for substeps are indicated as such on the Roadmap Element at runtime. Whether a substep is completed or not at runtime, depends on the application context and the user input. Therefore, substeps are statically declared but activated at runtime by the application (via the FPM API).

For more information on adding substeps and dynamically activating them, see Adding and Activating Substeps for GAF Applications.

**OVP Instance**

For information on the OVP floorplan, see Overview Page Floorplan.

**Adding and Activating Sub-Steps for GAF Applications**

**Procedure**

The configuration of substeps is similar to that of main steps. You can add one or more substeps to a main step and each substep can contain one or more UIBBs.

To add substeps, perform the following steps:

1. In the *FPM Configuration Editor*, open the Component Configuration window.
2. In the hierarchy, select the main step to which you want to add a substep. Choose Add Sub-Step from the action region.
3. In the attribute view, enter the following data:
   - SUBSTEP: Enter the Component ID and Name. The name will be shown at runtime beneath the corresponding substep.
   - SUBSTEP_UIBB: Enter the Component and Window. In Window enter the name of the Web Dynpro window of the interface view (not the name of the Web Dynpro view itself).

After a substep has been configured statically, you may invoke it at runtime via the FPM API. This is done by raising a special FPM event. Before raising this event, the event parameters are populated with the corresponding substep ID that you want to use. This is shown in the sample code below:
DATA: lo_fpm TYPE REF TO if_fpm,
     lr_event TYPE REF TO cl_fpm_event.
* get reference to FPM API
lo_fpm = cl_fpm_factory=>get_instance( ).
* create event
lr_event = cl_fpm_event=>create_by_id( cl_fpm_event=>gc_event_change_step ).
* fill event parameters
lr_event->mo_event_data->set_value( iv_key = cl_fpm_event=>gc_event_param_mainstep_id
     iv_value = <ID of Main Step> ).
lr_event->mo_event_data->set_value( iv_key = cl_fpm_event=>gc_event_param_substep_id
     iv_value = <ID of Sub-Step> ).
lr_event->mo_event_data->set_value( iv_key = cl_fpm_event=>gc_event_param_subvariant_id
     iv_value = <ID of Sub-Step variant> ).
* now raise event
Web Dynpro_this->fpm->raise_event( io_event = lr_event )

FPM Toolbar

FPM allows you to construct toolbars according to the SAP UI Guidelines. You choose which toolbar elements you require and FPM positions them in a predetermined location. FPM allows you to configure the following toolbar elements:

- Standard function buttons: buttons such as Save, Edit, Finish, Read-Only
- Application-specific buttons: buttons to which you add your own code
- Button choices: buttons which offer the user a dropdown menu with a list of further options. You can define the individual menu options in a button-choice and attach events to them. FPM provides no predefined events for these menu options but allows you to attach your own events instead. To attach your own predefined event to a button, enter a menu option name (Label) and the event ID. When the menu option is selected during run-time, the FPM will call up the attached event. A button choice is indicated in the Add Toolbar Element dialog box by a small arrow in the bottom right-hand corner of the button.
- Navigation menus

The Close button appears automatically on the FPM toolbar but you cannot configure it like the above standard function buttons. You can hide it by using the CNR API or with an application parameter FPM_HIDE_CLOSE=X.

Differences between an OIF and a GAF Toolbar

OIF Application

There is only one toolbar in every OIF variant. This toolbar contains more standard buttons than the GAF toolbar. Additionally, the OIF toolbar has two more options to create
application-specific buttons. FPM automatically adds a *Save* button to an OIF toolbar when you create the component configuration. As the *Save* button belongs to the category Activation Function, you can configure it (for example with a tooltip, label or event).

**GAF Application**

In a GAF application, every main step and substep inside a variant has its own toolbar. This enables you to have a different toolbar configuration at each step in the roadmap. FPM automatically adds the *Next* and *Previous* buttons to a GAF toolbar when you create the component configuration.

There is no ‘main’ toolbar in a GAF application. If you require a particular button on the toolbar at each step in the roadmap, you add it to each substep toolbar.

**Activities**

**Adding Elements to a Toolbar**

1. In the *FPM Configuration Editor*, locate the OIF or GAF component of your application and choose *Change*. This opens the OIF or GAF component configuration in edit mode.
2. To add an element to a toolbar, choose *Add Toolbar Element* in the action area. The *Add Toolbar Element* dialog box appears.
3. Select a button and choose *OK*. The button now appears in the hierarchy under *Toolbar* and the button’s editable attributes are visible in the preview.

**Adjusting the Toolbar Dynamically**

During runtime the content and visibility of the OIF and GAF toolbars may be changed via the *Context Navigation Region (CNR) APIs*. Note that there are different APIs for each floorplan type.

With these APIs you can dynamically change the FPM toolbars of both the initial screen and the main screen.

**Toolbar Buttons**

The following table describes the non self-explanatory toolbar buttons.

<table>
<thead>
<tr>
<th>Toolbar Button Name</th>
<th>Toolbar Button Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Activation Function</em></td>
<td>This button is intended primarily to be used as a <em>Save</em> button. As most applications require a <em>Save</em> button, the <em>FPM Configuration Editor</em> automatically adds this button to your configuration by default. The FPM Event <em>FPM_Save</em> is set as the default FPM Event ID but you can edit this.</td>
</tr>
<tr>
<td><em>Alternate</em></td>
<td>Use this button when you need to call an application-specific function from</td>
</tr>
</tbody>
</table>
**Function**

Your application screen. You can add your own application-specific event to it. This button appears in the same toolbar region as the Activation (Save) button.

**Other Function**

Use this button when you need to call an application-specific function from your application screen. You can add your own application-specific event to it. This button appears in a toolbar region separated from the Activation (Save) button. Additionally, it has attributes for Explanation and Button Design.

**You can Also/Related Links**

These two toolbar elements provide navigation links away from the FPM. These elements require a Role and an Instance, both of which are taken from a launchpad which you must first create and configure.

**Close**

FPM provides GAF applications with a Close button. You cannot configure this button (when you select it at design time, you will see no attributes). For technical reasons, this button is not visible in every system (see CSN Note #1234843). If you need to hide this button – for example your application is executed within an iView on a Portal page, please refer to chapters Using Application Parameters and Adjusting the toolbar using the CNR API.

**Exit to Main Step**

This is available only to sub-steps. If you click this button during run-time you return to the Main Step to which the button is assigned.

**Finish**

This is available only to main steps. If you click this button during run-time, the roadmap is executed sequentially; the FPM will navigate automatically through the roadmap as far as the last screen (before the confirmation screen) or will stop prematurely if it encounters an error.

**Next Step/Final Action**

Extra attributes are available for Next Step in the final roadmap step. These attributes (Label, Event ID) allow you to execute your own predefined final action before the confirmation screen. Each variant of the roadmap can have one Final Action.

### Toolbar Element Attributes

Toolbar elements have a variety of attributes and not every element has the same attributes. The table lists some of the non self-explanatory toolbar button attributes.

<table>
<thead>
<tr>
<th>Toolbar Element Attribute</th>
<th>Toolbar Element Attribute Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element ID</strong></td>
<td>Enter an Element ID if you want to change the properties of a toolbar element dynamically during runtime.</td>
</tr>
<tr>
<td><strong>Duplicate Toolbar</strong></td>
<td>This allows you to display a copy of the toolbar at the bottom of your application screen.</td>
</tr>
<tr>
<td><strong>Sequence Index</strong></td>
<td>This allows you to choose the order in which your application-specific UI elements (for example Other Function buttons, Main Steps) appear on the toolbar or in the hierarchy. The toolbar elements which FPM automatically adds to the toolbar cannot be rearranged using this attribute.</td>
</tr>
<tr>
<td><strong>Repeat Sel. Action</strong></td>
<td>This is available for button-choice elements. If you tick this checkbox, the menu option that is selected from a button-choice at run-time will then be</td>
</tr>
</tbody>
</table>
Toolbar
Element
Attribute | Toolbar Element Attribute Description
--- | ---
Select Action | visible as the button choice title for the current session. If the user wishes to select the same option next time, he must click only the button and not scroll through the list of menu options.

Enabled | This grays out a toolbar element; it renders a toolbar element unusable if the checkbox is not ticked.

Visibility | If you check the visibility attribute of both the button and the button-choice, only the button is visible in the toolbar.

### Toolbar Button Events

Every Standard Function button is attached to an FPM event (for example, *Edit* is connected to the FPM event `gc_event_edit`). The connection to these raised FPM events is hard-coded and cannot be changed. The event can, of course, be changed dynamically by calling other events. Some button events are pre-configured by the FPM (for example, the *Previous* and *Next* navigation button events and the *Save* button event) and require no extra code, but generally the application must provide the event processing. In general, the FPM ensures only that all affected UIBBs are informed. For example, although the FPM provides a *Print* button, there is no print support in FPM. FPM provides this button only to ensure that it is rendered according to the SAP UI Guidelines. The application must provide the necessary print functions.

The table below lists the toolbar buttons (and button-choices) and the events raised by them.

<table>
<thead>
<tr>
<th>Toolbar Button</th>
<th>Event Raised</th>
<th>Floorplan Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Activation Function</strong></td>
<td>self-defined via configuration</td>
<td>OIF</td>
</tr>
<tr>
<td><strong>Alternate Activation</strong></td>
<td>self-defined via configuration</td>
<td>OIF</td>
</tr>
<tr>
<td><strong>Check</strong></td>
<td><code>gc_event_check</code></td>
<td>OIF</td>
</tr>
<tr>
<td><strong>Close</strong></td>
<td><code>gc_event_close</code></td>
<td>OIF and GAF</td>
</tr>
<tr>
<td><strong>Delete Object</strong></td>
<td><code>gc_event_delete_current_object</code></td>
<td>OIF</td>
</tr>
<tr>
<td><strong>Edit</strong></td>
<td><code>gc_event_edit</code></td>
<td>OIF</td>
</tr>
<tr>
<td><strong>Exit to Main Step</strong></td>
<td><code>gc_event_exit_to_main_step</code></td>
<td>GAF</td>
</tr>
<tr>
<td><strong>Load Draft</strong></td>
<td><code>gc_event_load_draft</code></td>
<td>OIF</td>
</tr>
<tr>
<td><strong>New</strong></td>
<td><code>gc_event_new</code></td>
<td>OIF</td>
</tr>
<tr>
<td><strong>Next Object</strong></td>
<td><code>gc_event_next_object</code></td>
<td>OIF</td>
</tr>
<tr>
<td><strong>Next Step</strong></td>
<td>If no final action is defined: <code>gc_event_next_step</code></td>
<td>GAF</td>
</tr>
<tr>
<td></td>
<td>If a final action is defined: the self-configured event in the final action node and the next step event are raised</td>
<td></td>
</tr>
</tbody>
</table>
### Toolbar Button-Choice 

<table>
<thead>
<tr>
<th>Event Raised</th>
<th>Floorplan Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>self-defined via configuration</td>
<td>OIF</td>
</tr>
</tbody>
</table>

### IF_FPM_CNR_GAF Interface

This interface provides you with methods to dynamically change the FPM toolbar of an initial screen or main screen.

The interface is accessed via the `CL_FPM_SERVICE_MANAGER`, as the code below shows:

#### Accessing the API for a GAF application:

```php
DATA: lo_cnr_gaf TYPE REF TO if_fpm_cnr_gaf,
     lo_fpm TYPE REF TO if_fpm.
lo_fpm = cl_fpm_factory=>get_instance( ).
lo_cnr_gaf ?= lo_fpm->get_service(
    cl_fpm_service_manager=>gc_key_cnr_gaf ).
```
### Methods

This interface provides you with the methods described in the table below.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE_BUTTON</td>
<td>With this method either standard buttons or application-specific buttons can be created and edited. The parameter IV_FUNCTION defines the button type (see IF_FPM_CONSTANTS=&gt;gc_button). The ELEMENT_ID is needed if application-specific buttons must be changed subsequently.</td>
</tr>
<tr>
<td>DEFINE_BUTTON_CHOICE</td>
<td>With this method either standard button-choices or application-specific button-choices can be created and edited. The parameter IV_FUNCTION defines the button-choice type (see IF_FPM_CONSTANTS=&gt;gc_button_choice). The ELEMENT_ID is needed if application-specific buttons must be changed subsequently.</td>
</tr>
<tr>
<td>CREATE_SEPARATOR</td>
<td>Use this method to create a separator at runtime in the OTHER_FUNCTIONS area (application-specific).</td>
</tr>
<tr>
<td>SET_DUPLICATE_TOOLBAR</td>
<td>Use this method to activate or deactivate the duplication of the toolbar.</td>
</tr>
<tr>
<td>DEFINE_YOU_CAN_ALSO</td>
<td>Use this method to define launchpads for the You Can Also menu bar for (see Navigation API chapter).</td>
</tr>
<tr>
<td>DEFINE_RELATED_LINKS</td>
<td>Use this method to edit the menu bar for RELATED_LINKS (see Navigation API chapter).</td>
</tr>
<tr>
<td>GET_BUTTONS</td>
<td>This method determines which buttons (and their configurations) are to be shown in the toolbar.</td>
</tr>
<tr>
<td>GET_BUTTON_CHOICES</td>
<td>This method determines which button-choices (and their configurations) are to be shown in the toolbar.</td>
</tr>
<tr>
<td>GET_SEPARATORS</td>
<td>This method determines the positions of the separators in the toolbar (only in the Other Functions area).</td>
</tr>
<tr>
<td>GET_RELATED_LINKS</td>
<td>This method determines the contents of the Related Links menu in the toolbar.</td>
</tr>
<tr>
<td>GET_YOU_CAN_ALSO</td>
<td>This method determines the contents of the You Can Also menu in the toolbar.</td>
</tr>
</tbody>
</table>

### GAF Specific Parameters

Depending on the location of the UI elements that you wish to define, the following parameters (outlined in the table below) are passed with every GAF CNR API method:

<table>
<thead>
<tr>
<th>Location of UI Elements</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Step</td>
<td>• VARIANT_ID</td>
</tr>
<tr>
<td></td>
<td>• MAINSTEP_ID</td>
</tr>
</tbody>
</table>
An example of method calls to change the CNR of the GAF at runtime is shown below:

```abap
DATA: lo_cnr_gaf TYPE REF TO if_fpm_cnr_gaf,
  lo_fpm TYPE REF TO if_fpm.
lo_fpm = cl_fpm_factory=>get_instance( ).
lo_cnr_gaf ?= lo_fpm->get_service(
  cl_fpm_service_manager=>gc_key_cnr_gaf ).
lo_cnr_gaf ->define_button(
  EXPORTING
    iv_variant_id     = < optional; e.g. 'variant_1'; current variant if
    iv_mainstep_id    = < optional; 'mainstep_1'; current mainstep if
    iv_subvariant_id  = < optional; 'subvariant_xyz'>
    iv_substep_id     = < optional; 'substep_99'>
    iv_function       = < e.g. EXIT_TO, FINISH, OTHER_FUNCTIONS (appl-
                         specific buttons), SAVE_DRAFT, NEXT_STEP) see also
    IF_FPM_CONSTANTS=>gc_button >
    iv_screen         = < optional; the screen where the UI-Element has
                         to be changed (INIT, MAIN) >
  
    iv_element_id     = < optional; only if you want to change the
    iv_sequence_id    = < optional; only if you use OTHER_FUNCTIONS;
    determines the place where to insert this button >
    iv_design         = < optional; Button-Design >
    iv_enabled        = < optional; Button-Enabling >
    iv_explanation    = < optional; Button-Explanation >
    iv_on_action      = < optional; determines the Event-Id for a
    button; not possible with standard buttons >
    iv_text           = < optional; Button-Label >
    iv_tooltip        = < optional; Button-Tooltip >
    iv_visibility     = < optional; Button-Visibility >
    iv_default_button = < optional; only for NEXT button; by pressing
                         enter within an application triggers the action of this button>  ).

    iv_hotkey         = < optional; key-combination for activating the
    event of this button> 
```

<table>
<thead>
<tr>
<th>Location of UI Elements</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| Sub-Step                | • VARIANT_ID  
                        | • MAINSTEP_ID  
                        | • SUBVARIANT_ID  
                        | • SUBSTEP_ID  |
| Initial Screen          | • Screen    |
IF_FPM_CNR_OIF Interface

This interface provides you with methods to dynamically change the FPM toolbar of an initial screen or main screen.

The interface is accessed via the CL_FPM_SERVICE_MANAGER, as the code below shows:

Accessing the API for an OIF application:

```plaintext
DATA: lo_cnr_oif TYPE REF TO if_fpm_cnr_oif,
     lo_fpm TYPE REF TO if_fpm.
lo_fpm = cl_fpm_factory=>get_instance( ).
lo_cnr_oif ?= lo_fpm->get_service(cl_fpm_service_manager=>gc_key_cnr_oif).
```

Methods

This interface provides you with the methods described in the table below.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE_BUTTON</td>
<td>With this method either standard buttons or application-specific buttons can be created and edited. The parameter IV_FUNCTION</td>
</tr>
<tr>
<td>Method Name</td>
<td>Method Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DEFINE_BUTTON_CHOICE</td>
<td>With this method either standard button-choices or application-specific button-choices can be created and edited. The parameter IV_FUNCTION defines the button-choice type (see IF_FPM_CONSTANTS=&gt;gc_button_choice). The ELEMENT_ID is needed if application-specific buttons must be changed subsequently.</td>
</tr>
<tr>
<td>CREATE_SEPARATOR</td>
<td>Use this method to create a separator at runtime in the OTHER_FUNCTIONS area (application-specific).</td>
</tr>
<tr>
<td>SET_DUPLICATE_TOOLBAR</td>
<td>Use this method to activate or deactivate the duplication of the toolbar.</td>
</tr>
<tr>
<td>DEFINE_YOU_CAN_ALSO</td>
<td>Use this method to define launchpads for the You Can Also menu bar for (see Navigation API chapter).</td>
</tr>
<tr>
<td>DEFINE_RELATED_LINKS</td>
<td>Use this method to edit the menu bar for RELATED_LINKS (see Navigation API chapter).</td>
</tr>
<tr>
<td>GET_BUTTONS</td>
<td>This method determines which buttons (and their configurations) are to be shown in the toolbar.</td>
</tr>
<tr>
<td>GET_BUTTON_CHOICES</td>
<td>This method determines which button-choices (and their configurations) are to be shown in the toolbar.</td>
</tr>
<tr>
<td>GET_SEPARATORS</td>
<td>This method determines the positions of the separators in the toolbar (only in the Other Functions area).</td>
</tr>
<tr>
<td>GET_RELATED_LINKS</td>
<td>This method determines the contents of the Related Links menu in the toolbar.</td>
</tr>
<tr>
<td>GET_YOU_CAN_ALSO</td>
<td>This method determines the contents of the You Can Also menu in the toolbar.</td>
</tr>
</tbody>
</table>

**OIF Specific Parameters**

Since a toolbar exists for every OIF variant, only the VARIANT_ID must be passed with every OIF CNR API method.

**Example**

An example of method calls to change the CNR of the OIF at runtime is shown below:

```plaintext
DATA: lo_cnr_oif TYPE REF TO if_fpm_cnr_oif,
     lo_fpm TYPE REF TO if_fpm.
lo_fpm = cl_fpm_factory=>get_instance( ).
lo_cnr_oif = lo_fpm->get_service( cl_fpm_service_manager=>gc_key_cnr_oif ).
lo_cnr_oif->define_button( EXPORTING
```
iv_variant_id = < optional; e.g. 'variant_1'; current variant if skipped >
iv_function = < e.g. ACTIVATION_FUNCTIONS (appl-specific buttons), ALTERNATE_FUNCTIONS (appl-specific buttons), CHECK, DELETE_OBJECT, EDIT, LOAD_DRAFT, NEW, NEXT_OBJECT, OTHER_FUNCTIONS (appl-specific buttons), REVIOUS_OBJECT, PRINT, PRINT_PREVIEW, READ_ONLY, REDO, REFRESH, SAVE_AS, SAVE_DRAFT, SEND, START_OVER, UNDO, see also IF_FPM_CONSTANTS=>gc_button >
iv_screen = < optional; the screen where the UI-Element has to be changed (INIT, MAIN) >
iv_element_id = < optional; only if you want to change the button properties afterwards >
iv_sequence_id = < optional; only if you use OTHER_FUNCTIONS; determines the place where to insert this button >
iv_design = < optional; Button-Design >
iv_enabled = < optional; Button-Enabling >
iv_explanation = < optional; Button-Explanation >
iv_on_action = < optional; determines the Event-Id for a button; not possible with standard buttons >
iv_text = < optional; Button-Label >
iv_tooltip = < optional; Button-Tooltip >
iv_visibility = < optional; Button-Visibility >
iv_default_button = < optional; only for buttons CHECK and REFRESH; by pressing enter within an application triggers the action of this button >
iv_hotkey = < optional; key-combination for activating the event of this button >
lo_cnr_oif->define_button_choice(
EXPORTING
iv_variant_id = < optional; e.g. 'variant_1'; current variant if skipped >
iv_function = < e.g. NEW, OTHER_FUNCTIONS (appl-specific button-choices), PRINT, PRINT_PREVIEW, SEND, see also IF_FPM_CONSTANTS=>gc_button_choice >
iv_screen = < optional>; the screen where the UI-Element has to be changed (INIT, MAIN) >
iv_element_id = < optional; only if you want to change the button-choice properties afterwards >
iv_sequence_id = < optional; only if you use OTHER_FUNCTIONS; determines the place where to insert this button-choice >
iv_enabled = < optional; Button-Choice-Enabling >
iv_text = < optional; Button-Choice-Label >
iv_tooltip = < optional; Button-Choice-Tooltip >
iv_visibility = < optional; Button-Visibility >
it_menu_action_items = < menu elements of a Button-Choice >

FPM Complete Preview

In order to enable the preview feature for a self-developed UIBB you must implement the FPM Web Dynpro IF_FPM_CFG_UIBB_PREVIEW interface. The interface provides the UIBBB_PREVIEW method with the IV_INTERFACE_VIEW importing parameter, which is the actual visible view, and the EV_PREVIEW_WINDOW exporting parameter.

The application itself needs to check that the window which is returned is defined and will work at design time.
FPM Identification Region (IDR)

The Identification Region (IDR) consists of the following three areas:

- Header area (IDR Basic)
- Ticket area (IDR Extended)
- Items area

This is illustrated in the figure below:

Both the header and the ticket areas can be configured at design-time in the Component Configuration window for the IDR configuration.

Note the following points regarding the ticket area:

- The ticket area is only available for OIF applications.
- To configure the ticket area, choose Add IDR Extended.

Attributes for Ticket Top and Ticket Bottom appear. These attributes can be called dynamically to add label/value pairs, label/navigation link pairs and label/icon pairs to the ticket area.

Adjusting the IDR Dynamically

During runtime, use the IDR API to make changes to the individual IDR areas. This API consists of the methods encapsulated in the IF_FPM_IDR interface.

Adding a Link to the FPM Configuration Editor in the IDR

You can provide your application with a link to the FPM Configuration Editor from the IDR. For more information, see Providing a Link to the FPM Configuration Editor.

IF_FPM_IDR Interface

This interface provides you with methods to change the IDR dynamically at run-time.

The sample code below shows you how to access this interface:
DATA: lo_idr TYPE REF TO if_fpm_idr,
    lo_fpm TYPE REF TO if_fpm.
lo_fpm = cl_fpm_factory=>get_instance( ).
lo_idr := lo_fpm->get_service( cl_fpm_service_manager=>gc_key_idr ).

There are methods available for each of the following IDR areas:

- Header Area
- Ticket Area
- Items Area

### Methods for IDR Header Area

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_APPLICATION_TITLE</td>
<td>Retrieves the title text and its tooltip.</td>
</tr>
<tr>
<td>SET_APPLICATION_TITLE</td>
<td>Displays a new title text and tooltip in the header area.</td>
</tr>
<tr>
<td>SET_HEADER_VISIBILITY</td>
<td>Makes the header area visible or invisible.</td>
</tr>
</tbody>
</table>

### Methods for IDR Ticket Area

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_TICKET</td>
<td>Retrieves the texts of the ticket top, ticket bottom and their tooltips.</td>
</tr>
<tr>
<td>SET_TICKET</td>
<td>Displays new texts of the ticket top, ticket bottom and their tooltips.</td>
</tr>
<tr>
<td>SET_TICKET_VISIBILITY</td>
<td>Makes the ticket area visible or invisible.</td>
</tr>
</tbody>
</table>

### Methods for Items Area

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_ITEM_GROUP_BY_VAL</td>
<td>Adds a new item group to the item area. One item consists of a label, its tooltip, a value and the value’s tooltip. A group of items consists of an arbitrary amount of such items. With this method, you can add items to the IDR as simple static text strings. Therefore, if the value of an item needs to be changed at a later point in time, you will need to call method CHANGE_ITEM_GROUP_BY_VAL. The method ADD_ITEM_GROUP_BY_REF can also be used to pass references to Web Dynpro context nodes to the IDR. In this case, the value changes automatically when the value of the corresponding attribute in the context node changes.</td>
</tr>
<tr>
<td>CHANGE_ITEM_GROUP_BY_VAL</td>
<td>Changes the label and values that were passed to the IDR via the method ADD_ITEM_GROUP_BY_VAL.</td>
</tr>
<tr>
<td>ADD_ITEM_GROUP_BY_REF</td>
<td>Similar to add_item_group_by_val. Adds label/value items to the IDR. In this case, the value is not passed as a static text but as reference to a Web Dynpro context node attribute.</td>
</tr>
<tr>
<td>Method Name</td>
<td>Method Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>The advantage here is that the value can be of a type other than string. In addition, updating the value happens automatically; whenever the attribute of the context node changes, the IDR changes the visible value. It is also possible for the IDR to show the unit of the value. Do this using a flag; the actual unit is taken from the DDIC information of the value’s type. Therefore, this feature will only work if the type of the attribute in the context node, (which is passed to the IDR) has a defined DDIC unit.</td>
</tr>
<tr>
<td>ADD_NAVIGATION_ITEM</td>
<td>Adds a pair of label/navigation links to the IDR. The link itself is provided by the report launchpad. It makes no difference whether the link in the report launchpad is supplied by the database or is created dynamically during runtime via the report launchpad API. For more information about the report launchpad, refer to the report launchpad documentation. You specify the launchpad via the parameters instance and role. Since one launchpad may contain several targets (and this method is used to add only one target), use an additional parameter to specify the single target. The additional parameter is either the application alias or the navigation key.</td>
</tr>
<tr>
<td>CHANGE_NAVIGATION_ITEM</td>
<td>Use this method to edit a pair of label / navigation links that you added using the method ADD_NAVIGATION_ITEM. It is possible to change only the label and the link text with this method. If you want to change the target itself, use the report launchpad API.</td>
</tr>
<tr>
<td>ADD_IMAGE_ITEM</td>
<td>Adds pairs of label/icons to the IDR.</td>
</tr>
<tr>
<td>CHANGE_IMAGE_ITEM</td>
<td>Edits a label/icon pair that you added using the method ADD_IMAGE_ITEM.</td>
</tr>
<tr>
<td>CONTAINS_ITEM_GROUP</td>
<td>Checks whether a certain item group exists within the IDR.</td>
</tr>
<tr>
<td>REMOVE_ITEM_GROUP</td>
<td>Removes a certain item group from the IDR.</td>
</tr>
<tr>
<td>INITIALIZE_ITEMS</td>
<td>Clears all items from the IDR.</td>
</tr>
<tr>
<td>SET_ITEMS_VISIBILITY</td>
<td>Edits the visibility of the item area (the visibility status of all items, not just single items).</td>
</tr>
</tbody>
</table>

### Providing a Link to the FPM Configuration Editor in the IDR

There are currently two options to provide a link (in the IDR header area of your FPM application) which points to the *FPM Configuration Editor*:

- Using transaction SU3.

To do this, proceed as follows:

1. Open transaction SU3 and choose the *Parameters* tab.
2. Add the parameter `FPM_CONFIG_EXPERT` and set the Parameter Value to `X`.

The `Change Configuration` link appears in the IDR header area when you start the `FPM Configuration Editor`, via the `Web Dynpro Explorer`, for your application configuration. This corresponds to a change of the explicit and implicit configuration in development mode.

- Starting your FPM application with URL parameter `sap-config-mode=X`.

  The link `Adapt Configuration` appears in the IDR header area when you start the `FPM Configuration Editor` via `Web Dynpro application customize_component`. This corresponds to a customizing of the explicit and implicit configuration in the administrator mode. In the administrator mode you may adapt all elements of the configuration that have not been marked previously as final elements in the development mode.

**Quick Help**

You can use this function in a floorplan to provide application users with a quick help that gives a helpful explanation of a subview, initial screen, main step, or substep. The quick help is only displayed if the user has activated it using the context menu.

**Features**

You can either enter the quick help text directly or give a reference to a documentation object. It is a good idea to use a reference to a documentation object when the content of the quick help is used in multiple views or applications. If you enter a text directly and enter a reference to a documentation object, then the content of the documentation object is displayed as quick help.

You can display the quick help using the application's context menu. You can create, change, or delete quick help texts.

The quick help text is stored in the hierarchical view of the configuration editor either in the `Text` or `Documentation Object` attribute of an `Explanation` hierarchy element.

When you create a quick help from the action area of the configuration editor of Floorplan Manager, the `Explanation` hierarchy element is created below the corresponding hierarchy element (for example, subview details). Additionally, a suggested text is created by the system in the `Text` attribute.

You can also delete a quick help completely by selecting the `Delete` function in the attribute view of an `Explanation` hierarchy element.

**Creating Quick Help**
Procedure

To create a quick help in the configuration editor, you can either enter the quick help text directly or enter a reference to a documentation object.

Creating Quick Help as Direct Text

1. Select a Web Dynpro application configuration in the Object Navigator of the ABAP Workbench.
2. In the Web Dynpro Explorer: Display Web Dynpro Configuration screen, choose Web Dynpro Configuration → Test → Execute in Administrator Mode.

The Web Dynpro application is launched in a separate browser window.

3. In this window, go to the application's identification region and choose the Adapt Configuration link.
4. In the Editor for Web Dynpro ABAP Components — Customizing screen, choose Change.
5. On the Component Customizing <application name> screen, make sure that the Component-Defined view is on.
6. Choose Expand All.
7. Select the subview, main step, substep, or initial screen and choose Add Explanation in the action area of the configuration editor.

The system automatically generates a suggested text for the explanation in the Text field. The suggested text is composed of the title of the subview, main step, substep, or initial screen and the prefix "explanation". For example, if a substep has the title Details, then the suggested text is Explanation Details.

8. In the Text field in the attribute view, overwrite the suggested text with the quick help text you would like.
9. Save the configuration.
10. Test the new configuration.

Creating Quick Help Linking to a Documentation Object

1. To create a documentation object, choose SAP Menu → Tools → ABAP Workbench → Utilities → SE61 – Documentation.
   1. Choose General Text as the document class.
   2. Enter a technical name for the documentation object.
   3. Choose Create.
   4. Then enter desired quick help text.
   5. Choose Save Active.

The documentation object is now created and can be assigned as a quick help.
2. Select a Web Dynpro application configuration in the Object Navigator of the ABAP Workbench.

3. In the Web Dynpro Explorer: Display Web Dynpro Configuration screen, choose Web Dynpro Configuration → Test → Execute in Administrator Mode.

The Web Dynpro application is launched in a separate browser window.

4. In this window, go to the application's identification region and choose the Adapt Configuration link.

5. In the Editor for Web Dynpro ABAP Components — Customizing screen, choose Change.

6. On the Component Customizing <application name> screen, make sure that the Component-Defined view is on.

7. Select the subview, main step, substep, or initial screen for which you would like to add a quick help and choose Add Explanation in the action area of the configuration editor.

The system automatically generates a suggested text for the explanation in the Text field.

8. In the Documentation Object field of the attribute view, enter the technical name of the documentation object.

9. Save the configuration.

10. Test the new configuration.

**Variants**

In some cases the final configuration of an OIF view switch or a GAF roadmap may only be decided at runtime. For example, assume that an initial screen asks you to select one of three options. The subsequent roadmap or view switch that appears is dependent on the option you selected in the initial screen. FPM makes this possible by allowing you to configure variants. Each variant is a complete set of configuration data for an OIF view switch or a GAF roadmap. You use the input from the initial screen (or from other startup information, such as application parameters) to select one of the variants.

**Configuring Variant Selection**

Variant selection is controlled programmatically with an application-specific configuration controller (AppCC).

To configure variant selection, proceed as follows:

1. Implement the interface IF_FPM_OIF_CONF_EXIT (or IF_FPM_GAF_CONF_EXIT) in one of the application components or in a new component. This interface has only one method OVERRIDE_EVENT_OIF (or OVERRIDE_EVENT_GAF) which passes a handler object of type IF_OIF (respectively IF_GAF). This handler object provides the API with information to manipulate the floorplan configuration at runtime.
2. To select the variant, use the \texttt{SET\_VARIANT} method of this object as follows:

OIF Instance

\begin{verbatim}
method OVERRIDE\_EVENT\_OIF .
  ... case io_oif->mo_event->MV\_EVENT\_ID.
    when if\_fpm\_constants=>gc\_event-leave\_initial\_screen.
      io_oif->set\_variant( ).
    ...
\end{verbatim}

GAF Instance

\begin{verbatim}
method OVERRIDE\_EVENT\_GAF .
  ... case io_gaf->mo_event->MV\_EVENT\_ID.
    when if\_fpm\_constants=>gc\_event-leave\_initial\_screen.
      io_gaf->set\_variant( ).
\end{verbatim}

In this sample code the variant selection takes place after the initial screen is exited. This is the latest point at which it is possible to select the variant. You can, however, select the variant at an earlier stage.

The last thing to do is to declare the AppCC to the FPM:

1. In the \textit{FPM Configuration Editor}, open the component configuration editor window. In the control region, choose \textit{Change} \rightarrow \textit{Global Settings}.
2. In the \textit{Global Settings} dialog box, under \textit{Application-Specific Parameters}, enter the Web Dynpro Component which you are using as an application-specific configuration controller.
3. Choose \textit{Save}.

\textbf{Initial Screen}

The \textit{Initial Screen} is an optional screen. It is composed of one or more UIBBs.

\textbf{Adding an Initial Screen}

1. Start the \textit{FPM Configuration Editor} and open the OIF or GAF component configuration of your component.
2. In the control region, choose Add → Initial Screen.
3. Choose Add UIBB → Add UIBB and enter the component (ID) and the relevant component window.

The FPM adds the Start Button automatically to the toolbar of the initial screen. It is non configurable. When you choose this button at run-time, FPM raises the event IF_FPM_CONSTANTS=>GC_EVENT-LEAVE_INITIAL.Screen, exits the initial screen, and displays the first roadmap step (in GAF instances) or View Switch (in OIF instances).

Occasionally, you need to omit the initial screen from your application. If this is the case, raise the LEAVE_INITIAL_SCREEN event within your application-specific code:

```java
data: lo_fpm type ref to if_fpm
lo_fpm = cl_fpm_factory=>get_instance( )
lo_fpm->raise_event_by_id(IF_FPM_CONSTANTS=>GC_EVENT-LEAVE_INITIAL_SCREEN).
```

If your application has no initial screen, FPM displays the view switch (OIF) or the first roadmap step (GAF) at start-up.

**Skipping the Initial Screen**

OIF and GAF applications may start with an initial screen, in which you select the object you intend to work with. If the object is already known by the application (e.g. you are calling the application with the parameters already set), the initial screen is unnecessary. To skip an initial screen at runtime, proceed as follows:

Launch the FPM event LEAVE_INITIAL_SCREEN. You can launch this one of two ways:

- in the OVERRIDE_EVENT_*-method of your AppC
- in the PROCESS_BEFORE_OUTPUT method of one of your initial screen UIBBs (if you are not using an AppCC):

```java
data: lo_fpm type ref to if_fpm,
   lv_object_id type string.
* Check event id
   if lv_event_id = if_fpm_constants=>gc_event_start.
* Determine if Parameter OBJECT_ID is provided
   lo_fpm = cl_fpm_factory=>get_instance( ).
   lo_fpm->mo_app_parameter->get_value(  
      exporting lv_key = 'OBJECT_ID'
      importing ev_value = lv_object_id ).
* In case OBJECT_ID is set, navigate directly to the main floorplan * area
   if not lv_object_id is initial.
      lo_fpm->raise_event_by_id(  
         if_fpm_constants=>gc_event-leave_initial_screen ).
   endif
```
**Confirmation Screen**

FPM does not provide a confirmation screen by default. However, a confirmation screen is available to both the OIF and GAF instances.

Confirmation screens may be variant dependent (that is, each variant of your application may require a different confirmation screen). FPM allows you to configure a confirmation screen for each variant.

**Confirmation Screen in OIF Instances**

In OIF applications, the confirmation screen appears only when the object currently being processed in the application is deleted. After an object has been deleted, the confirmation screen appears in place of the normal view switch. Note that if your application does not include a delete function, you do not need a confirmation screen.

**Confirmation Screen in GAF Instances**

Most GAF applications use a final confirmation step at the end of the roadmap. This confirmation step informs the user that the action he has just executed has been completed successfully. The configuration of the `FPM_GAF_COMPONENT` explicitly supports such a use case.

**Adding and Configuring the Confirmation Screen**

To add a *Confirmation Screen* to your application, perform the following steps:

1. Start the *FPM Configuration Editor* of your application component and open the *Component Configuration* screen.
2. In the control region, choose ➤ Add → Confirmation Screen. You can then select the radio buttons in the hierarchy to move between the Confirmation Screen and other screens in your application.

You display the confirmation screen in OIF applications by pressing the standard DELETE button during run-time. This raises the event `IF_FPM_CONSTANTS=>GC_EVENT-DELETE_CURRENT_OBJECT`. You can also raise this event within your application-specific code:

```plaintext
data: lo_fpm type ref to if_fpm
lo_fpm = cl_fpm_factory=>get_instance( )
lo_fpm->raise_event_by_id(IF_FPM_CONSTANTS=>GC_EVENT-
DELETE_CURRENT_OBJECT)
```
FPM Event Loop

In Web Dynpro ABAP programming, a user interaction is reflected by a Web Dynpro action. If you require a user interaction to affect not only a local component but other components in the application too, the Web Dynpro action must be transferred to an FPM event.

This FPM event then passes through an FPM phase model (Event Loop) which is integrated into the Web Dynpro phase model. Within the FPM event loop all involved components can participate in the processing of the event.

If the FPM event results in another screen assembly (for example, navigation to another step in a GAF application or the selection of another view or sub view in an OIF application), the FPM handles this itself; there is no need for the application to fire plugs or similar.

Raising Standard Events

In a floorplan-based application, most events are triggered when a user chooses Next or Previous (in a GAF instance) or when switching from one view to another (in an OIF instance). For these interactions, the FPM automatically initiates the FPM event loop. Furthermore, these standard events are handled generically by the FPM.

However, there are scenarios where a standard event needs to be triggered from within an application-specific UIBB, for example by-passing the initial screen if all necessary start-up parameters have been provided as URL parameters.

Each FPM event is represented at runtime by an instance of the class CL_FPM_EVENT. This class encapsulates all information (including the ID and additional, optional parameters) which is needed to execute the event.

Triggering the FPM Event Loop

To trigger an FPM event loop, you complete the following two steps:

1. Create an instance of CL_FPM_EVENT with the appropriate attributes. For all the standard event IDs, there are constants available in the IF_FPM_CONSTANTS interface.
2. Raise the event by calling the method IF_FPM~RAISE_EVENT and passing on the instance of CL_FPM_EVENT.

When an event requires no additional parameters, other than the event ID, the FPM offers an additional method RAISE_EVENT_BY_ID. This makes Step 1 above obsolete. In this case, raise the FPM event as detailed in the sample code below:

```plaintext
data lo_fpm type ref to if_fpm
lo_fpm = cl_fpm_factory=>get_instance()
```
Since it is unknown whether the event can be executed successfully or not at the point the event is raised, do not enter code after the call to RAISE_EVENT[BY_ID].

**Triggering Application-Specific Events**

To raise an application-specific event, follow the same rules as described in Triggering the Event Loop. The only difference is that the FPM, since it does not know the semantics of the event, does not perform specific actions for this event. However, the processing of the event is identical, in that all involved components participate in the event loop in the same way as with ‘standard events’ (see Reacting to Framework Events).

The following code provides an example of triggering an application-specific event (including event parameters):

```plaintext
data: lo_fpm   type ref to if_fpm,
     lo_event type ref to cl_fpm_event.
create object lo_event
  exporting
    iv_event_id = 'DELETE_AIRPORT'.
lo_event->mo_event_data->set_value(
  iv_key   = 'AIRPORT_ID',
  iv_value = lv_airport_id).
lo_fpm = cl_fpm_factory=>get_instance().
lo_fpm->raise_event( io_event = lo_event ).
```

**Reacting to Framework Events**

The FPM event loop is integrated into the Web Dynpro phase model, as the following figure shows:
Key Web Dynpro Methods

The following Web Dynpro methods are important in FPM applications:

<table>
<thead>
<tr>
<th>Method</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOINIT</td>
<td>A standard Web Dynpro method that is called only once in the lifetime of a Web Dynpro component by the Web Dynpro runtime. This method is used to initialize your component, for example initialize attributes and create helper classes.</td>
</tr>
<tr>
<td>DOBEFOREACTION</td>
<td>A standard Web Dynpro method that is called by the Web Dynpro runtime on all visible UIBBs when the user triggers a round trip. According to Web Dynpro programming guidelines, generic validations must be handled in this method; for example check that all mandatory fields are filled.</td>
</tr>
</tbody>
</table>

Action handler (ONACTION...)

The registered Web Dynpro action handler is called. You then have the following options:

- If the user interaction does not affect other UIBBs, and there is no need for FPM features such as data-loss dialog boxes, you can handle the action locally in your UIBB. Use standard Web Dynpro programming; for example selection of another radio-button leads to different enabled/disabled settings of other controls on the same view.
- However, for all actions which may affect other UIBBs, raise an FPM event.
Different Categories of Web Dynpro Interfaces

Regarding the behavior of instantiating the Web Dynpro components and their participation within the FPM event loop, the Web Dynpro interfaces provided by the FPM can be divided into two categories:

- Category 1

  More than one Web Dynpro component implements this Web Dynpro interface. Those which do may have more than one instance and the instances may only participate in a part of the FPM event loops during the application’s lifetime.

  The following Web Dynpro interfaces belong to this category:
  
  o IF_FPM_UI_BUILDING_BLOCK
  o IF_FPM_TRANSACTION, IF_FPM_WORK_PROTECTION
  o IF_FPM_RESOURCE_MANAGER

- Category 2

  Only one Web Dynpro component implements this Web Dynpro interface. The corresponding Web Dynpro component has only one instance which participates at all FPM event loops that happen during the application’s lifetime.

  The following Web Dynpro interfaces belong to this category:
  
  o IF_FPM_APP_CONTROLLER
  o IF_FPM_SHARED_DATA
  o IF_FPM_OIF_CONF_EXIT (or IF_FPM_GAF_CONF_EXIT)

Overview Page Floorplan (OVP)

The overview page floorplan (OVP) is based on WD component FPM_OVP_COMPONENT.

The OVP is a configuration of applications containing a set of pages and the navigation between the pages. You can use the OVP to model an application interface that initially displays an overview of the most important data to the users. From this overview, users can navigate to the editing pages to add, change or delete data and navigate back again.

Structure of an OVP

Content Areas
Each OVP floorplan consists of at least one content area. Content areas can be divided into numerous sections. Each section can contain numerous application-specific views (UIBBs) and generic user interface building blocks (GUIBBs). Each content area is assigned a content type. The following content area types are available with the OVP:

- Main view
- Edit page
- Initial screen
- Dialog box

Sections

You use sections to structure a content area. Various layouts for a section are possible; you can use the Section Layout attribute to determine the number of columns you want in a section.

You assign UIBBs and GUIBBs to the individual columns within a section.

The UIBBs and GUIBBs can also be rendered as panels (by choosing the UIBB/GUIBB Rendering Type attribute Tray (panel) in the configuration editor). If you render the UIBBs and GUIBBs inside a section as panels, the UIBB or GUIBB title and all their toolbar buttons and actions are displayed in the panel header.

The following diagram illustrates the structure of an OVP content area:

![Diagram of OVP content area structure]

Personalization

Personalization allows users to change the structure of an overview page at runtime to suit their own individual requirements. Personalization can be switched on and off in the configuration editor or by API method call for the following parts of an application:

- the whole application
- a given content area
Personalization Editor

To open the Personalization Editor at runtime, click the Personalize icon in the page toolbar.

The Personalization Editor allows you to perform the following activities:

- Add or remove UIBBs to sections (you can drag and drop UIBBs that are listed in the section Currently Not Displayed Assignment Blocks onto a section)

- Change the number of columns within a section using the Layout Type dropdown list

- Decide whether the UIBBs in a column should be stacked and the number of stacks per column.

  You can stack multiple UIBBs inside a panel on top of each other; the UIBBs are displayed as part of a tab strip in the panel header. To do this, select the column attribute Stacked Layout. Note that you can also do this inside the configuration editor by selecting the Column n Stacked attribute of the appropriate section.

- When the application is run in configuration (expert) mode (URL parameter: sap-config-mode=X), the personalization settings are valid for all end users who do not have their own individual settings. Choosing the Reset to Default button in end-user mode restores the administration personalization (if it exists); if there is no administration personalization, choosing Reset to Default restores the original configuration settings.

UIBB Toolbar Buttons

You can define the set of toolbar buttons that are displayed in the panel toolbar using the following methods:

- Define a list of actions in the configuration editor of the content area in which the UIBB is assigned. You can define actions and assign the corresponding FPM events and target content areas.
If the UIBB actions are exposed from the UIBB at runtime, for example when the UIBB is based on a generic UIBB and the list of actions is provided by the feeder class, you can select the Capture Actions attribute for the UIBB toolbar in the configuration editor. FPM then uses the UIBB-API to determine the available toolbar actions at runtime and the resulting buttons are added to the panel header toolbar alongside the configured buttons.

**OVP-Related FPM Events for Navigation**

You can use the following FPM events to enable navigation between the OVP content areas:

<table>
<thead>
<tr>
<th>Event ID (to be entered in configuration editor)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPM_CALL_FULL_SCREEN</td>
<td>Navigate to edit page.</td>
</tr>
<tr>
<td>FPM_CALL_DEFAULT_EDIT_PAGE</td>
<td>Navigate to default edit page (can be defined for a UIBB in configuration).</td>
</tr>
<tr>
<td>FPM_BACK_TO_MAIN</td>
<td>Leave edit page and navigate back to overview main page (without saving the data).</td>
</tr>
<tr>
<td>FPM_SAVE_AND_BACK_TO_MAIN</td>
<td>Transactional event. Leave edit page and navigate back to overview main page.</td>
</tr>
<tr>
<td>FPM_CHANGE_CONTENT_AREA</td>
<td>Replace current content area by another content area (for example, replace one overview page by another overview page).</td>
</tr>
<tr>
<td>FPM_OPEN_DIALOG</td>
<td>Open a dialog box. Remember to pass the ID of the dialog box content area using event parameter TARGET_CONTENT_AREA.</td>
</tr>
</tbody>
</table>

**Dynamic Changes at Runtime**

You can use the following methods to apply dynamic changes at runtime:

- **OVP CNR API** (similar to that of the OIF and GAF)
  Use this to make simple adjustments of the toolbar buttons.

- **ACC (application controller component)**
  Use this for complete access to the OVP API.

Further details for both these methods are described below.

**OVP CNR API**

The interface IF_FPM_CNR_OVP provides methods to change toolbar buttons in an OVP application at runtime. This interface includes the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>

(C) SAP AG Floorplan Manager ABAP - Developer's Guide Preliminary Version
GET_CONTENT_AREAS
Get a list of all content areas currently available.

GET_CURRENT_CONTENT_AREA
Get the current content area.

GET_UIBBS
Get a list of UIBBs for a content area or section.

GET_TOOLBAR_ELEMENTS
Get a list of existing toolbar elements.

GET_TOOLBAR_BUTTON
Get attributes of a toolbar button.

GET_TOOLBAR_TOGGLE_BUTTON
Get attributes of a toolbar toggle button.

GET_TOOLBAR_BUTTON_CHOICE
Get attributes of a toolbar button-choice.

ADD_TOOLBAR_BUTTON
Add a new toolbar button to an existing toolbar.

ADD_TOOLBAR_TOGGLE_BUTTON
Add a new toolbar toggle button to an existing toolbar.

ADD_TOOLBAR_BUTTON_CHOICE
Add a new toolbar button-choice to an existing toolbar.

CHANGE_TOOLBAR_BUTTON
Change an existing toolbar button.

CHANGE_TOOLBAR_TOGGLE_BUTTON
Change an existing toolbar toggle button.

CHANGE_TOOLBAR_BUTTON_CHOICE
Change an existing toolbar button-choice.

REMOVE_TOOLBAR_ELEMENT
Remove a toolbar element.

GET_APPLICATION_PARAMETERS
Get the application parameters.

CHANGE_APPLICATION_PARAMETERS
Change the application parameters.

GET_PAGE_SELECTOR
Get the page selector dropdown list box label and entries.

CHANGE_PAGE_SELECTOR
Change the page selector dropdown list box label and entries.

SET_GENERIC_BUTTON_ACTION_TYPE
Set the action type of the generic OVP buttons.

SET_SIDE_PANEL_LINK
Define the side panel link.

SET_TAG_VALUE
Set a value for a tag (side panel).

---

**Application Controller API**

An application WD component or ABAP class can implement the interface `IF_FPM_OVP_CONF_EXIT` that provides the method `OVERRIDE_EVENT_OVP`. This method offers access to the OVP interface of type `IF_FPM_OVP`. In addition to the methods of interface `IF_FPM_CNR_OVP`, which are described in the previous table, you can also use the following methods to change the OVP floorplan state dynamically:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_CONTENT_AREA</td>
<td>Add a new content area.</td>
</tr>
<tr>
<td>CHANGE_CONTENT_AREA</td>
<td>Change an existing content area.</td>
</tr>
<tr>
<td>REMOVE_CONTENT_AREA</td>
<td>Remove an existing content area.</td>
</tr>
<tr>
<td>GET_SECTIONS</td>
<td>Get a list of all sections of a content area.</td>
</tr>
<tr>
<td>ADD_SECTION</td>
<td>Add a new section to a content area.</td>
</tr>
<tr>
<td>CHANGE_SECTION</td>
<td>Change an existing section.</td>
</tr>
<tr>
<td>REMOVE_SECTION</td>
<td>Remove a section.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ADD_UIBB</td>
<td>Add a UIBB to an existing content area or section.</td>
</tr>
<tr>
<td>CHANGE_UIBB</td>
<td>Change an existing UIBB.</td>
</tr>
<tr>
<td>REMOVE_UIBB</td>
<td>Remove a UIBB.</td>
</tr>
<tr>
<td>GET_EVENT</td>
<td>Get the current FPM event.</td>
</tr>
<tr>
<td>IS_EVENT_CANCELLED</td>
<td>Returns the result indicating whether the current FPM event is cancelled or not.</td>
</tr>
<tr>
<td>CANCEL_EVENT</td>
<td>Cancel event processing.</td>
</tr>
<tr>
<td>IF_FPM_WIRE_MODEL~ADD_WIRE</td>
<td>Add a wire.</td>
</tr>
<tr>
<td>IF_FPM_WIRE_MODEL~REMOVE_WIRE</td>
<td>Remove a wire.</td>
</tr>
<tr>
<td>IF_FPM_WIRE_MODEL~GET_WIRES</td>
<td>Get all wires.</td>
</tr>
</tbody>
</table>

### Setting a Default ALV View for a Freestyle UIBB

When you embed freestyle UIBB into an FPM application, you can select the default ALV view which will be displayed when the freestyle UIBB is launched during runtime.

#### Design Time Settings in the FPM Configuration Editor

To select the default ALV views, complete the following steps:

1. Implement the marker WD interface IF_FPM_CFG_CONF_ALV_USAGE in the freestyle UIBB which has a usage on the WD component SALV_WD_TABLE.
2. Using the FPM Configuration Editor, add this freestyle UIBB to your floorplan.
3. In the Component Configuration view of the Configuration Editor, choose the Attributes button of the embedded UIBB. Attributes of the UIBB, including the table Configurable ALV Tables, is displayed.
4. In this table, use the field help to enter one ALV view for each usage on the WD component SALV_WD_TABLE.

Note that this feature is restricted to those ALV views that have been created previously on the configuration level. To do this, you must run the FPM application with the URL parameter sap-config-mode = config and create the views using the ALV Settings dialog box.

#### Rendering the ALV Views during Runtime

The selected ALV view is not automatically applied to the usage on the WD component SALV_WD_TABLE in your freestyle UIBB. Instead, you must instantiate the usage on the WD component SALV_WD_TABLE with the corresponding ALV configuration key.

To do this, apply the ALV configuration keys in the method WDDOINIT of your freestyle UIBB with the following code:

```plaintext
method wddoinit.
  data lo_cmp_usage type ref to if_wd_component_usage.
  data lo_fpm_type ref to cl_fpm.
  data ls_conf_comp_usage type fpm_t_uibb_conf_comp_usage.
  data ls_conf_comp_usage type fpm_s_uibb_conf_comp_usage.
```
For more information on ALV views, see *SAP List Viewer in Web Dynpro for ABAP* in the SAP NetWeaver Library.

## FPM Dialog Component

The FPM dialog component provides a framework for supporting dialog boxes. FPM dialog boxes are supported in all the floor plans (OIF, GAF, and OVP).

### Structure of an FPM Dialog Box

The content of FPM dialog boxes can be freestyle UIBBs or GUIBBs. The CNR and IDR regions are not available in the FPM dialog boxes.

### Example of an FPM Dialog Box

![Example of an FPM Dialog Box](image-url)
FPM supports up to three levels of dialog boxes (opening one dialog box from within another dialog box), Dialog 1 to Dialog 3 (see following screenshot):

- Dialog 1 and Dialog 2 display any of the following button sets: Ok and Cancel, Close or no button
- Dialog 3 displays only the Close button. It is not possible to open another FPM dialog box after displaying a Dialog 3.

Example of the 3 Levels of FPM Dialog Boxes

![Example of the 3 Levels of FPM Dialog Boxes](image)

You can add a title to an FPM dialog box and you can also add OK, CANCEL, CLOSE button sets, tool tips for the buttons and choose a layout for the UIBBs in the dialog box.

**FPM Dialog Box Features**

- FPM dialog boxes are provided with two buttons OK and Cancel with default action. The dialog box buttons are handled by FPM itself.
- FPM dialog boxes support all UIBB types. This means UIBBs can have application-specific buttons. Any buttons specific to the UIBB are defined and handled by the UIBB and are embedded within the UIBB (see following screenshot).
FPM dialog boxes have their own message area and message handling. FPM dialog boxes allow you to raise local and global messages. For more information see, Message Manager in Dialog Boxes.

FPM dialog boxes have one or more UIBBs and their layout can be configured. For more information see, Configuring FPM Dialog Boxes.

The FPM error page can be triggered in FPM dialog boxes. For more information see, FPM Error page in Dialog Boxes.

FPM dialog boxes support navigation to URLs and Web Dynpro applications. Suspend and Resume navigation is also supported. The navigation with dialog boxes is configured similar to that of the main screen.

FPM dialog boxes support usage of data-loss dialog boxes. Any UIBBs configured in an FPM dialog box can raise a data-loss dialog box. This data-loss dialog box is handled in the event loop.

Applications can raise FPM events like FPM_SAVE, FPM_NEW, FPM_REFRESH, and so on and application-specific events in FPM dialog boxes.

FPM dialog boxes can be opened from the application toolbar. The toolbar button is configured with the dialog event ID. The toolbar properties can be used to maintain the FPM event ID and the event parameters to open the dialog box. For more information see, Configuring FPM Dialog Boxes.

FPM dialog boxes support all GUIBBs and composite UIBBs.

FPM dialog boxes support data transfer across dialog boxes through event data.

Creating and Configuring an FPM Dialog Box

You configure FPM dialog boxes in the FPM configuration editor. Each dialog box is uniquely identifiable by the Dialog ID.

Proceed as follows to configure FPM dialog boxes:

1. To add an FPM dialog box to an application, choose Add in the configuration editor, above the Hierarchy. Choose Dialog Box from the dropdown list.
2. In the dialog box that appears, enter the Dialog Box ID and provide a name for the dialog box. The Dialog Box ID cannot contain special characters other than '_'.
3. The dialog box is created for the application. Enter the UIBB attributes for the dialog box. More UIBBs can be added to the dialog box by choosing the Add UIBB button in the configuration editor screen.
4. You can change the layout for the UIBBs. Choose the top node in the Hierarchy. The attributes for the dialog box appear. Select the appropriate layout type from the Layout Type dropdown list.
5. You can also change the button set for the dialog box. Choose Dialog Box in the Hierarchy. In the Attributes section, select the appropriate button set from the Dialog Buttons dropdown list.

6. You can create multiple dialog boxes in a single application. To access the individual dialog boxes, select the Dialog Box radio button (just above the Hierarchy). The dropdown list is activated and allows you to select the dialog box; the attributes of the selected dialog box are displayed.

7. It is possible to trigger dialog boxes by choosing a button in the toolbar. To do this, first create a button in the toolbar. Provide an event ID for the button. Choose the Toolbar element in the Hierarchy. A table is shown in the configuration editor where the button event can be associated with the event to open the dialog box OPEN_DIALOG_BOX. Pass the event parameter details in the Maintain Event Parameters table. It is mandatory to pass the value for the event parameter DIALOG_BOX_ID. In this manner FPM determines which dialog box is opened when a particular toolbar button is chosen.
Opening and Closing FPM Dialog Boxes

You open FPM dialog boxes using the following methods:

- **IF_FPM API**

  The floor plan interface IF_FPM contains the method OPEN_DIALOG_BOX. The *Dialog ID* must be passed to this API. The table below describes the method with its parameters:

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN_DIALOG_BOX</td>
<td>IV_DIALOG_BOX_ID</td>
<td>This method is used in the application to open the dialog box.</td>
</tr>
<tr>
<td></td>
<td>IS_DIALOG_BOX_PROPERTIES (optional)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IO_EVENT_DATA (optional)</td>
<td></td>
</tr>
</tbody>
</table>

- **FPM event**

  Raise the FPM event (FPM_OPEN_DIALOG) cl_fpm_event => gc_event_open_dialog_box.

  The closing of FPM dialog boxes is handled by FPM itself; FPM triggers an event (FPM_CLOSE_DIALOG) cl_fpm_event=>gc_event_close_dialog_box.

**Event Processing in Dialog Boxes**

All the UIBB types that can be used in the main screen can also be used in FPM dialog boxes. FPM events or any application-specific events can be handled in the FPM dialog boxes.

The MV_IS_DIALOG_MODE Attribute

The IF_FPM-MV_IS_DIALOG_MODE attribute provides information on the state of an application, whether it is in a dialog screen or in a main screen. The application can read the state (OPENED or CLOSED) using this attribute.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV_IS_DIALOG_MODE</td>
<td>FPM_DIALOG_STATE</td>
<td>Indicates whether application is in dialog box or in main screen.</td>
</tr>
</tbody>
</table>

(Read Only)

Sample Coding to Call A Dialog Box

Opening a Dialog Box using Direct API
DATA: lo_fpm TYPE REF TO if_fpm,
     lv_window_id TYPE fpm_dialog_window_id.

lv_window_id = 'CONTENT_AREA_1'.
lo_fpm = cl_fpm_factory=>get_instance( ).
lo_fpm->open_dialog_box( EXPORTING
     iv_dialog_box_id = lv_window_id ).

Opening a Dialog Box by Raising an FPM Event

DATA: lo_event_params TYPE REF TO if_fpm_parameter,
     lr_event TYPE REF TO cl_fpm_event,
     lv_window_id TYPE fpm_dialog_window_id,
     lo_fpm TYPE REF TO if_fpm.

lv_window_id = 'CONTENT_AREA_1'.
CREATE OBJECT lo_event_params type cl_fpm_parameter.
lo_event_params->set_value( EXPORTING
     iv_key = if_fpm_constants=>gc_dialog_box-id
     iv_value = lv_window_id
     ).

CREATE OBJECT lr_event
EXPORTING
     iv_event_id = cl_fpm_event=>GC_EVENT_OPEN_DIALOG_BOX
     io_event_data = lo_event_params.

lo_fpm = cl_fpm_factory=>get_instance( ).
lo_fpm->raise_event( lr_event ).

Message Manager for FPM Dialog Boxes

FPM manages the message handling (regarding the parent component and the FPM dialog boxes) in terms of Visibility and Lifetime of a message in FPM dialog boxes.

<table>
<thead>
<tr>
<th>Lifetime/Visibility</th>
<th>Message Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>FPM takes care of the visibility based on the UI guidelines. The <em>Automatic</em> messages in an FPM dialog box are cleared after every roundtrip or if a new message is raised on a dialog box.</td>
</tr>
<tr>
<td></td>
<td>On Dialog Close:</td>
</tr>
<tr>
<td></td>
<td>– <em>Automatic</em> messages are not carried forward to the parent, or the previous level dialog box.</td>
</tr>
<tr>
<td>View</td>
<td>The message is visible as long as the view to which the message is</td>
</tr>
</tbody>
</table>

(C) SAP AG   Floorplan Manager ABAP - Developer's Guide
Preliminary Version 69
bound is available,

On Dialog Close

-Messages with MANU_VIEW lifetime are not carried to the parent or the parent dialog box.

On Dialog Open:

Messages with MANU_VIEW lifetime reappear on the dialog box when it is reopened until it is cleared manually.

These messages are not carried forward from parent window or the parent dialog box on the dialog box.

The message is visible as long as the controller that has raised the message is available.

On Dialog Close:

Messages with MANU_CONT lifetime would be passed to the parent if it is raised through the same controller.

On Dialog Open:

-Messages with MANU_CONT lifetime would be passed to the dialog box from parent window or the parent dialog if it is raised through the same controller.

The message is permanently displayed, in dialog box and the parent window throughout the application, until it is manually cleared by the application developer.

Pop-up: the message is visible only in a message dialog box, until it is closed

Error Page of an FPM Dialog Box

Navigation to an error page from dialog boxes is also possible at all levels of dialog boxes. If the application encounters an error during dialog box processing and subsequently wants to navigate to an error page, it can be done by calling the API of the error page. The main screen is replaced by the error page and the dialog box is automatically closed. No further navigation is possible.

FAQs on FPM Dialog Boxes

1. How do I add a dialog box to the application?
   In the FPM configuration editor, add a dialog box to the application with the unique Dialog ID, configure the UIBBs and the dialog box properties. Call the dialog box with the Dialog ID from the Web Dynpro component.
2. How do I open the dialog box using API?
The FPM dialog box can be opened by two methods; one is using API. Instantiate FPM and call the method open_dialog_box exporting Dialog ID and the properties from the Web Dynpro component.

3. How do I open the dialog box by raising events?
Set the event parameter Dialog ID. Instantiate cl_fpm_event with the event parameters and the event name cl_fpm_event=>GC_EVENT_OPEN_DIALOG_BOX. Raise the event using raise_event method.

4. Is it possible to add multiple UIBBs and GUIBBs in a single dialog box?
Yes.

5. Is it possible to raise an event in the dialog box? If so how do I handle these events?
Yes. Any events can be handled in the event loop methods as it is handled in the main screen.

6. How do I transfer data across dialog boxes?
Transfer data using the IO_EVENT_DATA parameter. Set the value to be transferred and pass the value while opening another FPM dialog box.

7. How do I configure GUIBBs?
GUIBBs can be configured similar to free style UIBBs. The FORM GUIBB, LIST GUIBB and Tabbed GUIBB can be configured in FPM dialog boxes. While adding UIBBs to the dialog box, add the configuration details of the GUIBB.

8. How do I configure toolbar buttons to open a dialog box?
Add a button in the application toolbar and configure the properties with the FPM Event ID. In the toolbar properties, maintain the action IDs and the event parameters Dialog Box ID.

9. How does transient behavior works with FPM dialog boxes?
Transient behavior is enabled only when FPM dialog box is closing. The parent screen of the FPM dialog box is not released even if transient behavior is switched on. The dialog box component is released when it is closed. Switch ON and OFF of transient behavior, whilst opening and closing of FPM dialog boxes, is handled by FPM itself.

10. How can I know the Dialog ID of the dialog box being closed? During closing of dialog the dialog id of the dialog being closed is available in the close dialog event. You can read the close dialog event to find the dialog id.

**Generic User Interface Building Block (GUIBB)**

You can use Floorplan Manager to compile application-specific views (UIBBs) from one or more applications that were realized as Web Dynpro components into new Floorplan Manager applications. These views generally include the majority of actual applications. Since the views were previously created using the Web Dynpro ABAP foundation, there generally was a high level of variance in the display and navigational behavior of the views. These views cannot be configured in Floorplan Manager.

By introducing generic user interface building blocks, Floorplan Manager has made it possible to improve the uniformity of application-specific views. Generic user interface building blocks are design templates for which, at design time, the application defines the data to be displayed along with a configuration. The concrete display of the data on the user
interface is not determined and generated by the GUIBB until runtime. This is done automatically using the configuration provided.

Floorplan Manager provides the following generic user interface building blocks:

- **Form component** (Web Dynpro component: FPM_FORM_UIBB)
- **List component** (Web Dynpro component: FPM_LIST_UIBB)
- **Tabbed component** (Web Dynpro component: FPM_TABBED_UIBB)
- **Search component** (Web Dynpro component: FPM_SEARCH_UIBB)
- **Launchpad component** (Web Dynpro component: FPM_LAUNCHPAD_UIBB)

**Feeder Classes**

A class that implements the **IF_FPM_GUIBB_FORM** interface (for form components), the **IF_FPM_GUIBB_LIST** interface (for list components), or the **IF_FPM_GUIBB_SEARCH** interface (for search components) and provides all necessary application-specific information to the GUIBB.

**Structure**

Using the **GET_DEFINITION** method, the class defines the field catalog of the component and supplies the component at runtime with data from the application using the **GET_DATA** method.

**Features**

Feeder class implementations are based on a predefined interface definition providing all necessary methods and corresponding signatures in order to standardize the communication between the application and the GUIBB.

This communication embraces the following:

- Application definition (for example data definition, structure or table definitions and their technical aspects)
- Default layout information and corresponding field dependencies
- The (optional) action definition based on metadata
- The action/event handling and data forwarding to the underlying application model

**Form Component (GUIBB FORM)**

A generic design template for displaying data in a form that is implemented using the Web Dynpro component **FPM_FORM_UIBB**.

You use this design template in application-specific views (UIBB) where you want to display data using a form. You can determine the concrete display of the data in a form when configuring the Web Dynpro component **FPM_FORM_UIBB**.

**Structure**

A **FORM** is comprised of various sub objects:
- **ELEMENT**

  Elements are descriptor/field combinations that can be configured for the display type of the field or descriptors.

- **MELTINGGROUP**

  A melting group is a group of multiple fields.

- **TOOLBAR**

  Contains buttons that can have actions assigned to them and can be executed in the form.

- **GROUP**

  A group consists of elements, melting groups, and toolbars. You can enter a separate name and group type for each group. The following group types are possible:

  - Full screen width with one column
  - Full screen width with two columns
  - Half screen width with one column

  Only one element or melting group can be displayed per line in a column.

The information that can be displayed on a form is determined by the feeder class assigned to the configuration of the Web Dynpro component FPM_FORM_UIBB.

**Integration**

You can configure a form component using the Form Editor for Floorplan Manager.

**IF_FPM_GUIBB_FORM Interface**

The following tables describe the methods (and their attributes) of the IF_FPM_GUIBB_FORM interface.

If your application does not need a particular method, implement an empty method, otherwise the system will dump.

You must implement at least the following methods:

- GET_DEFINITION
- GET_DATA
Methods of IF_FPM_GUIBB_FORM Interface

**GET_DEFINITION:**

Allows the feeder to provide all necessary information for configuring a form: the list of available fields and their properties and the list of actions (FPM events).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EO_FIELD_CATALOG</td>
<td>Is of type CL_ABAP_STRUCTDESCR. The components of this object are the available fields. The simplest way to provide a field catalog is to create a flat DDIC structure containing all fields and then get the field catalog via eo_field_catalog := CL_ABAP_STRUCTDESCR=&gt;describe_by_name( &lt;name&gt; )</td>
</tr>
</tbody>
</table>

The form GUIBB supports only flat structures. When using deep structures, only the highest level fields are available.

| ET_FIELD_DESCRIPTION | Here you can provide the additional information needed to create the form, for example Label_by_DDIC, LABEL_REF |
| ET_ACTION_DEFINITION | A list of all actions (which will be transformed to FPM Events at runtime) that you can assign to form elements. |
| ET_SPECIAL_GROUPS | Here you have the same options that you have in the ABAP ALV (see function module REUSE_ALV_GRID_DISPLAY) to group the fields within your field catalogue. You must enter the special group for each field in the field description table in field SP_GROUP. At design-time the FPM Configuration Editor groups the fields. This is an easier way to find fields if your field catalogue contains many fields. |

**GET_PARAMETER_LIST:**

Called at design time and allows you to define a list of the parameters that the feeder class supports. This list is used by the FPM Configuration Editor to provide the input fields for these parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT_PARAMETER_DESCR</td>
<td>Is returned from this method. It describes which parameter is possible. In Field TYPE, the DDIC type needs to be entered.</td>
</tr>
</tbody>
</table>

**INITIALIZE:**

Called at runtime when the form is created. It is the first feeder method which is called from FPM.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_PARAMETER</td>
<td>Contains a list of the feeder parameters and</td>
</tr>
</tbody>
</table>
the values for them specified in the configuration.

**FLUSH:**

The first feeder method which is called during an event loop. Whenever an FPM event is triggered (this includes all round trips caused by the form itself) this method is called. Use it to forward changed form data to other components in the same application.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_CHANGE_LOG</td>
<td>Lists all changes made by the user.</td>
</tr>
<tr>
<td>IS_DATA</td>
<td>Is a structure containing the changed data</td>
</tr>
</tbody>
</table>

**PROCESS_EVENT:**

Called within the FPM event loop, it forwards the FPM `PROCESS_EVENT` to the feeder class. Here the event processing can take place and this is where the event can be canceled or deferred.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_EVENT</td>
<td>The FPM event which is to be processed</td>
</tr>
<tr>
<td>EV_RESULT</td>
<td>The result of the event processing. There are 3 possible values:</td>
</tr>
<tr>
<td></td>
<td>ev_result = if_fpm_constants=&gt;gc_event_result-OK</td>
</tr>
<tr>
<td></td>
<td>ev_result = if_fpm_constants=&gt;gc_event_result-FAILED.</td>
</tr>
<tr>
<td></td>
<td>ev_result = if_fpm_constants=&gt;gc_event_result-DEFER</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>A list of messages which shall be displayed in the message region.</td>
</tr>
</tbody>
</table>

**GET_DATA:**

Called within the FPM event loop and forwards the FPM `PROCESS_BEFORE_OUTPUT` event to the feeder class. Here you specify the form data after the event has been processed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_EVENT</td>
<td>The FPM event which is to be processed</td>
</tr>
<tr>
<td>IT_SELECTED_FIELDS</td>
<td>The list of fields necessary for the form rendering. Provide only the data for the fields listed in this table; all other fields are neither visible at runtime nor used as reference for visible fields.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>A list of messages which shall be displayed in the message area.</td>
</tr>
<tr>
<td>EV_DATA_CHANGED</td>
<td>For performance reasons, the GUIBB adjusts the data in the form only if the data has been changed. To indicate this, set this flag</td>
</tr>
</tbody>
</table>
whenever you change the data to be displayed within this feeder.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV_FIELD_USAGE_CHANGED</td>
<td>Indicates whether or not the field usage has been changed by this method. If you change the field usage without setting this flag to X, your changes are ignored.</td>
</tr>
<tr>
<td>EV_ACTION_USAGE_CHANGED</td>
<td>Indicates whether or not the action usage has been changed. Use an X to indicate whether you changed the action usage. If you do not, your changes are ignored.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CT_FIELD_USAGE</th>
<th>Field usage to change. The field usage consists of the field attributes which might change at runtime (for example, enabled, and visible). Note that if you change the fixed values of a field, set the flag FIXED_VALUES_CHANGED for this field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT_ACTION_USAGE</td>
<td>Action usage to change. The action usage consists of the attributes related to actions which might change at runtime. For example, visibility. If an action is rendered as a button, then the visibility setting of the button is defined here.</td>
</tr>
</tbody>
</table>

### GET_DEFAULT_CONFIG:

Call this if you want to have a default configuration. Use it to call pre-configured form configurations when a user starts the FPM Configuration Editor. This avoids the user, who uses a feeder class to create a form, having to create it from the beginning.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_LAYOUT_CONFIG</td>
<td>Of type IF_FPM_GUIBB_FORM_CONFIG: This object provides the API to create a default configuration</td>
</tr>
</tbody>
</table>

### CHECK_CONFIG:

Call this if you want to make your own application-specific checks on the configuration in the FPM Configuration Editor immediately before saving.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_LAYOUT_CONFIG</td>
<td>Of type IF_FPM_GUIBB_FORM_CONFIG: This object provides the API to read the configuration to be saved.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>A list of messages which shall be displayed in the message region.</td>
</tr>
</tbody>
</table>
Form Editor for Floorplan Manager

You use the form editor to adjust a form in an application to your specific business requirements. This is done by configuring form components.

Features

The form editor consists of the following work areas:

- **Preview**

  In the preview, all form elements from the current configuration are displayed so as to give you a picture of the layout of the form.

- **Hierarchy**

  All form elements (groups, melting groups, and elements) are displayed in the hierarchy.

- **Attribute view**

  Attributes of the currently selected form element that can be changed using the form editor are displayed in the attribute view.

- **Action area**

  The action area contains links to all the actions you can execute for the form component. The actions that can be selected depend on the concrete configuration of the form. This means that it can differ within a configuration.

The form editor provides you with the following actions:

- Add Group
- Add Melting Group
- Edit Feeder Class
- Edit Parameters
- Configure Toolbar
- Configure Group

The form editor provides you with the following functions for editing a group:

- **Change Group Attributes**

  The group name, group type, and index can be changed.

- Add New Group
- Add Melting Group
Add Element

You can select a field from the field catalog and determine the label text and display type.

Delete Group

The form editor provides you with the following functions for editing a melting group:

Add Group Element

You can select a field from the field catalog. Fields are configured in more detail by changing the group element attributes.

Change Group Element Attributes

The display type, visibility of the label, label text, and index can be changed. Any other group element attributes that can be changed depend on the display type.

Delete Group Element

The form editor provides you with the following functions for editing a toolbar:

Add Button

Change Button

Delete Button

The form editor is launched in a separate browser window. You can launch the form editor in change or display mode and save your changes at any time.

The component-defined processing view is pre-set. Make sure that this view is selected before configuring a form component.

The form editor launches from the configuration editor for Floorplan Manager automatically when you launch the configuration of an application-specific view (UIBB) that uses the FPM_FORM_UIBB Web Dynpro component.

Using the CHECKBOX_GROUP Display Type in a Form

As of release SAP NetWeaver 7.0 enhancement package 2, the checkbox group display type is available. In contrast to the other display types, the application has to ensure that everything works.

To use this field to full extent, consider the following:
The field type must be of type *Character* and the field length needs to be at least the number of checkboxes you expect.

The values of the checkboxes need to be set as fixed values for the field.

Default values for a checkbox can be set in the field.

For example, mark the second checkbox field value = X.

Within the `FLUSH` method you get the data and the change log back.

In the field for the checkbox you see at the index of the field whether it is checked (Checked = X) or not (Unchecked = Space).

**Remark on field length**

As we set an X for the marked entry on the place in the field it is necessary to have the length at least as long as much as you have values.

You have a field with length 10 and 10 fixed values. If you mark a checkbox this place will be X, checkbox 1 2 3 4 5 6 7 8 9 10. Let us presume all are marked. Within `FLUSH` method this field of the structure will be updated XXXXXXXXXX. Let us presume the ninth field is not marked. Within `FLUSH` method this field of the structure will be updated XXXXXXXX_X.

Let us presume the field would be of char10 and you have 12 fixed values. If you mark the twelfth checkbox and want to return this information to the feeder class it does not work as you can only fill ten Xs in the field.

That is the reason why the field length should be at least equal to the numbers of fixed values.

**Group Layout in a Form**

As of release SAP NetWeaver 7.0 enhancement package 2, there are two possibilities of the layout for groups:

- Previous group layout version

  A group can be of type *Full Width One Column*, of type *Full Width 2 Columns* or of type *Half Width One Column*. You can arrange the groups within the form depending on the group type you choose.

- New group layout version
A form can be of layout type 2 *Column Layout*. This layout can be assigned to a form on the level of the `FORM` node. If you choose this layout the groups are all of type *Half Width One Column*. The groups can be put in the first or in the second column of the form. That means each column is separately rendered and the group heights do not depend on the group next to it.

**Add Form**

At any time, you can add a new form as an additional, application-specific view to a Floorplan Manager application. Depending on the floorplan and the application-specific views already embedded, you can position a new form as a [form component](#) in one of the following ways:

- in a subview of an object instance
- in a main step of a guided activity
- in a substep of a guided activity
- in a tabbed component in a master UIBB
- in a tabbed component in a tab UIBB

**Prerequisites**

If you would like to add a new form, you must assign a [feeder class](#) to the form. This feeder class must exist in the system.

**Procedure**

Substep A: Calling up the configuration editor of Floorplan Manager

1. Select a Web Dynpro application configuration in the *Object Navigator* of the *ABAP Workbench*.
2. On the *Web Dynpro Explorer: Display Web Dynpro Configuration* screen, choose "Web Dynpro Configuration → Test → Execute".

The Web Dynpro application is launched in a separate browser window.

3. In this window, go to the application’s identification region and choose the *Adapt Configuration* link.
4. In the *Editor for Web Dynpro ABAP Components — Customizing* screen, choose *Change*.
5. On the *Component Customizing <application name>* screen, make sure that the *Component-Defined* view is on.

Substep B: Adding a Form

1. In the preview, select the place where you would like to add the new form.
2. If no UIBB has been defined for a subview or step, choose *Attributes*.
3. If a UIBB has already been assigned, choose *Add UIBB* or *Add Form UIBB* in the action area.
Now specify the UIBB as a form component by entering the following values in the attribute view:

1. In the *Component* field, enter `FPM_FORM_UIBB`.

   If you have chosen *Add Form UIBB*, check whether the field is correctly prefilled.

2. In the *View* field, enter `FORM_WINDOW`.

   If you have chosen *Add Form UIBB*, check whether the field is correctly prefilled.

3. In the *Configuration Name* field, enter a name for the form component configuration.

   Although the *Configuration Name* field is not marked as a required entry, it is necessary to enter a name here. Otherwise, the *Configure UIBB* action is not activated.

   The *Configuration Type*, *Configuration Variant*, and *Sequence Order* are optional.

4. Choose *Save*.

   The system has entered the form component as a new UIBB with the name `UIBB: FORM_WINDOW`.

**Substep C: Configuring the Form**

1. Choose *Configure UIBB*.

2. In the *Editor for Web Dynpro ABAP Components — Configuration* screen, choose *Create*.

3. In the *Create Configuration* dialog window, enter a description and choose a package.

4. Choose *OK*.

5. In the *Edit Feeder Class* dialog window, enter the feeder class that you would like to assign the new form to.

   You can search for a feeder class here.

   To do this, choose the input help in the *Edit Feeder Class* window. The system opens the *Select Feeder Class IF_FPM_GUIBB_FORM* dialog window. You can search for a feeder class here. Choose *Start Search*. The system lists all feeder classes for the *IF_FPM_GUIBB_FORM* Web Dynpro interface. Select a feeder class and choose *OK*. The system copies the feeder class into the *Feeder Class* field.
6. Choose Edit Parameters.
7. In the Edit Parameters window, you can choose a value for the feeder class parameters.
8. Choose OK.

Result

The system opens the form editor for the new form.

List Component (GUIBB LIST)

A generic design template for displaying data in a list that is implemented using the Web Dynpro component FPM_LIST_UIBB.

You use this design template in application-specific views (UIBB) where you want to display data using a list. You can determine the concrete display of the data in a list when configuring the Web Dynpro component FPM_LIST_UIBB.

Structure

A list consists of a number of columns. The component-defined view gives you the opportunity to specify:

- Which data is displayed in which columns.
- Which display type (such as display field or input field) is used in which column.
- Which order the columns are arranged in.
- The number of columns and rows that can be displayed in the view at one time.

The data of a list that can be displayed is determined by the feeder class that is assigned to the configuration of the Web Dynpro component FPM_LIST_UIBB.

Integration

You can configure a list component using the List Editor for Floorplan Manager.

IF_FPM_GUIBB_LIST Interface

The following tables describe the methods (and their attributes) of the IF_FPM_GUIBB_LIST interface.

If your application does not need a particular method, implement an empty method, otherwise the system will dump.

You must implement at least the following methods:
### GET_DEFINITION:

Allows the feeder to provide all necessary information for configuring a list: the list of available fields and their properties and the list of actions (FPM events).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EO_FIELD_CATALOG</td>
<td>Is of type <code>CL_ABAP_STRUCTDESCR</code>. The components of this object are the available fields. The simplest way to provide a field catalog is to create a flat DDIC structure containing all fields and then get the field catalog via <code>eo_field_catalog</code> = <code>CL_ABAP_STRUCTDESCR=&gt;describe_by_name('&lt;name&gt;')</code>. The list GUIBB supports only flat structures. When using deep structures, only the highest level fields are available.</td>
</tr>
</tbody>
</table>
| ET_FIELD_DESCRIPTION      | Optional only
Is used to add additional properties for the columns:

- **Attribute** `label_by_ddic`  
  Indicates whether column header text should be taken from DDIC or not
- **Attribute** `search_help`  
  DDIC search help and OVS search help
- **Attribute** `header_text_wrapping`  
  Determines whether the column header text shall be wrapped or not.
- Attributes for read-only, mandatory, enabled and visibility.
- Attributes for enabling filtering and sorting, for formatting amongst others.
- **CELL_DESIGN_REF**  
  Points to a technical column which contains data to change the background color of cells (data type `WDUI_TABLE_CELL_DESIGN`). Use this to change the background color of single cells. |
| ET_ACTION_DEFINITION      | A list of all actions (which will be transformed to FPM events at runtime) that you can assign to list elements. If an action is only active when a lead selection occurs, set the attribute `DISABLE_WHEN_NO_LEAD_SEL` to True.                                                                                               |
| ET_SPECIAL_GROUPS         | Here you have the same options that you have in the ABAP ALV (see function module `REUSE_ALV_GRID_DISPLAY`) to group the fields within your field catalogue. You must enter the special group for each field in the field description table in field `SP_GROUP`. At design-time the FPM Configuration Editor groups the fields. This is an easier way to find fields if your field... |
GET_DEFINITION:

Allows the feeder to provide all necessary information for configuring a list: the list of
available fields and their properties and the list of actions (FPM events).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalogue</td>
<td>contains many fields.</td>
</tr>
</tbody>
</table>

GET_PARAMETER_LIST:

Called at design time and allows you to define a list of the parameters that the feeder class
supports. This list is used by the FPM Configuration Editor to provide the input fields for
these parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT_PARAMETER_DESCR</td>
<td>Is returned from this method. It describes which parameter is possible. In Field TYPE, the DDIC type needs to be entered.</td>
</tr>
</tbody>
</table>

INITIALIZE:

Called at runtime when the list is created. It is the first feeder method which is called from
FPM.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_PARAMETER</td>
<td>Contains a list of the feeder parameters and the values for them specified in the configuration.</td>
</tr>
</tbody>
</table>

FLUSH:

The first feeder method which is called during an event loop. Whenever an FPM event is
triggered this method is called (this includes all round trips caused by the list itself). Use it to
forward changed list data to other components in the same application.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_CHANGE_LOG</td>
<td>Lists all changes made by the user.</td>
</tr>
<tr>
<td>IS_DATA</td>
<td>Is a structure containing the changed data.</td>
</tr>
</tbody>
</table>

PROCESS_EVENT:

Called within the FPM event loop and forwards the FPM PROCESS_EVENT to the feeder class.
Here the event processing can take place and this is where the event can be canceled or
defered.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_EVENT</td>
<td>The FPM event which is to be processed.</td>
</tr>
</tbody>
</table>
| EV_RESULT | The result of the event processing. There are 3 possible values:
- ev_result = if_fpm_constants=>gc_event_result-OK
- ev_result = if_fpm_constants=>gc_event_result-FAILED.
- ev_result = if_fpm_constants=>gc_event_result-DEFER |
| ET_MESSAGES | A list of messages which shall be displayed in the message region. |

GET_DATA:

Called within the FPM event loop, it forwards the FPM PROCESS_BEFORE_OUTPUT event to
the feeder class. Here you specify the list data after the event has been processed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_EVENT</td>
<td>The FPM event which is to be processed</td>
</tr>
</tbody>
</table>
GET_DATA:
Called within the FPM event loop, it forwards the FPM `PROCESS_BEFORE_OUTPUT` event to the feeder class. Here you specify the list data after the event has been processed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_SELECTED_FIELDS</td>
<td>The list of fields necessary for the list rendering. Provide only the data for the fields listed in this table; all other fields are neither visible at runtime nor used as reference for visible fields.</td>
</tr>
<tr>
<td>ETMESSAGES</td>
<td>A list of messages which shall be displayed in the message area.</td>
</tr>
<tr>
<td>EV_DATA_CHANGED</td>
<td>For performance reasons, the GUIBB adjusts the data in the list only if the data has been changed. To indicate this, set this flag whenever you change the data to be displayed within this feeder.</td>
</tr>
<tr>
<td>EV_FIELD_USAGE_CHANGED</td>
<td>Indicates whether or not the field usage has been changed by this method. If you change the field usage without setting this flag to 'X', your changes are ignored.</td>
</tr>
<tr>
<td>EV_ACTION_USAGE_CHANGED</td>
<td>Indicates whether or not the action usage has been changed. Use an 'X' to indicate whether you changed the action usage. If you do not, your changes are ignored.</td>
</tr>
<tr>
<td>CS_DATA</td>
<td>The list data to be changed.</td>
</tr>
<tr>
<td>CT_FIELD_USAGE</td>
<td>Field usage to change. The field usage consists of the field attributes which might change at runtime (for example enabled or disabled, visible or invisible, mandatory or optional, read-only or edit). Use it to control the properties of columns.</td>
</tr>
<tr>
<td>CT_ACTION_USAGE</td>
<td>Action usage to change. The action usage consists of the attributes related to actions which might change at runtime (for example enabled or disabled, visible or invisible, mandatory or optional, read-only or edit). Use it to control the properties of toolbars. If an action is rendered as a button, the visibility setting (for example) of the button is defined here.</td>
</tr>
<tr>
<td>CV_FIRST_VISIBLE_ROW</td>
<td>This parameter indicates the absolute table index of the first visible table row. Use it to move the current position of the table’s vertical scrollbar.</td>
</tr>
</tbody>
</table>

You can assign the attributes for `CT_FIELD_USAGE` and `CT_ACTION_USAGE` either to single cells or to whole columns.

If you want to set these attributes for the whole column, use the corresponding fields in the `field_usage` structure.

If you would like to set these attributes for single cells, proceed as follows:

1. Create a new column for the table (add a field to the field catalog in the `GET_DEFINITION` method).
2. Define the column as a technical column that is not visible at runtime, by setting the field `technical_field` to ‘X’. This column contains the properties of the cells.

3. In the `GET_DEFINITION` method, adjust the field description accordingly. For example, you have a column A and you want to set the property `Read_only` for single cells in that column. For this reason you created a technical column B. In the field description, set `read_only_ref` to B.

---

### GET_DEFAULT_CONFIG:

Call this if you want to have a default configuration. Use it to call pre-configured list configurations when a user starts the FPM Configuration Editor. This avoids the user, who uses a feeder class to create a list, having to create it again from the beginning.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_LAYOUT_CONFIG</td>
<td>Of type <code>IF_FPM_GUIBB_LIST_CONFIG</code>: This object provides the API to create a default configuration.</td>
</tr>
</tbody>
</table>

### CHECK_CONFIG:

Call this if you want to make your own application-specific checks on the configuration in the FPM Configuration Editor immediately before saving.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_LAYOUT_CONFIG</td>
<td>Of type <code>IF_FPM_GUIBB_LIST_CONFIG</code>: This object provides the API to read the configuration to be saved.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>A list of messages which shall be displayed in the message region.</td>
</tr>
</tbody>
</table>

---

### IF_FPM_GUIBB_LIST_PAGING Interface

The following tables describe the methods of the `IF_FPM_GUIBB_LIST_PAGING` interface. This is an optional interface and should only be implemented by the feeder class if the application wants to make use of the paging feature for the GUIBB list. Paging means that not all data of a list is loaded at once but only those portions that are needed at a certain time. You may consider implementing this interface if your list has more than 100 rows and thereby reducing memory and shortening response times.

You must implement at least the following methods:

- GET_DEFINITION
- PROCESS_EVENT
- GET_DATA
Methods

INITIALIZE:

This method is called once during start up. The feeder may use to switch paging on or off.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV_PAGING_ACTIVE</td>
<td>Tells the list UIBB whether the paging feature is active or not.</td>
</tr>
<tr>
<td>CV_BUFFERING_ACTIVE</td>
<td>Tells the list UIBB whether buffering of pages should be active or not. It is recommended to not buffer pages as in most cases the data is already buffered in the backend and thus it is avoided to have multiple buffers for the same data.</td>
</tr>
</tbody>
</table>

GET_ABSOLUTE_AMOUNT_OF_ROWS:

This method is called whenever the list data has been changed by the feeder.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV_AMOUNT_OF_ROWS</td>
<td>Tells the list UIBB how many rows the list has in total.</td>
</tr>
<tr>
<td>CV_FIRST_VISIBLE_ROW</td>
<td>Tells the list UIBB which should be the first visual row (all indexes are absolute).</td>
</tr>
</tbody>
</table>

FLUSH:

Is called at the beginning of each FPM event. This method replaces the method IF_FPM_GUIBB_LIST=>FLUSH.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_CHANGE_LOG</td>
<td>Contains the change log (all indexes are absolute).</td>
</tr>
<tr>
<td>IV_NEW_LEAD_SEL</td>
<td>Contains the new lead selection index (all indexes are absolute)</td>
</tr>
<tr>
<td>IV_OLD_LEAD_SEL</td>
<td>Contains the old lead selection index (all indexes are absolute)</td>
</tr>
</tbody>
</table>

GET_PAGE:

This method is called whenever the list UIBB needs data from the feeder, e.g. in case when the user scrolls within the table or the list UIBB ask for new data.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV_START_ABSOLUTE</td>
<td>Contains the start index</td>
</tr>
<tr>
<td>IV_AMOUNT_ROWS</td>
<td>Contains the required amount of rows.</td>
</tr>
<tr>
<td>IT_SELECTED_FIELDS</td>
<td>Tells what columns are configured</td>
</tr>
<tr>
<td>CT_DATA</td>
<td>This parameter is used for the table data.</td>
</tr>
</tbody>
</table>

PROCESS_EVENT:

This method is called within the FPM event loop. It forwards the FPM PROCESS_EVENT event to the feeder class. It replaces the IF_FPM_GUIBB_LIST=>PROCESS_EVENT method.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_EVENT</td>
<td>The FPM event which is to be processed.</td>
</tr>
<tr>
<td>IV_LEAD_INDEX</td>
<td>Contains the lead index (all indexes are absolute).</td>
</tr>
<tr>
<td>IT_SELECTED_LINES</td>
<td>Tells which rows are selected.</td>
</tr>
<tr>
<td>IV_RAISED_BY_OWN_UI</td>
<td>Tells whether the event was raised by the actual search UIBB or by some other UIBB.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>The feeder class may return messages via this parameter. As usual the messages will be displayed within the FPM message area.</td>
</tr>
</tbody>
</table>
GET_DATA:

Called within the FPM event loop and forwards the FPM `PROCESS_BEFORE_OUTPUT` event to the feeder class. The main purpose of this method is to transport data from the feeder class to the list UIBB. It replaces the `IF_FPM_GUIBB_LIST=>GET_DATA` method. For a detailed description of the parameter refer to the documentation of `IF_FPM_GUIBB_LIST=>GET_DATA` method.

List Editor for Floorplan Manager

You use the list editor to adjust a list within an application to your specific business requirements. This is done by configuring list components.

Features

The list editor consists of the following work areas:

- **Preview**

  In the preview, the list in the current configuration is displayed so as to give you a picture of the layout of the list.

- **Hierarchy**

  All list elements (columns, toolbar, and parameters) are displayed in the hierarchy as a tree structure.

- **Attribute view**

  Attributes of the currently selected list element that can be changed using the list editor are displayed in the attribute view.

- **Action area**

  The action area contains links to all the actions you can execute for the list component. Which actions can be selected depends on the concrete configuration of the list. This means that the selection of actions can differ within a configuration.

The list editor provides you with the following actions:

- Edit Feeder Class
- Edit Parameters
- Configure Column
- Configure Toolbar

The list editor provides you with the following functions for editing a column:

- Add Column
You can select a field from the field catalog and determine the column header and display type.

- Delete Column

Under **Settings**, the attribute **Width of List** allows you to determine the total width of the list.

The list editor is launched in a separate browser window. You can launch the list editor in change or display mode and save your changes at any time.

The component-defined processing view is pre-set. Make sure that this view is selected before configuring a list component.

The list editor launches from the configuration editor for Floorplan Manager automatically when you launch the configuration of an application-specific view (UIBB) that uses the `FPM_LIST_UIBB` Web Dynpro component.

**Add List**

You can add an additional, application-specific view to a Floorplan Manager application at any time in the form of a new list. Depending on the floorplan and the application-specific views already embedded, you can position a new list component in one of the following ways:

- in a subview of an object instance
- in a main step of a guided activity
- in a substep of a guided activity
- in a tabbed component in a master UIBB
- in a tabbed component in a tab UIBB

**Prerequisites**

If you would like to add a new form, you must assign a feeder class to the form. This feeder class must first be created and programmed.

**Procedure**

Substep A: Calling up the configuration editor of Floorplan Manager

1. Select a Web Dynpro application configuration in the **Object Navigator** of the ABAP Workbench.
2. On the **Web Dynpro Explorer: Display Web Dynpro Configuration** screen, choose ➔ **Web Dynpro Configuration ➔ Test ➔ Execute**.

The Web Dynpro application is launched in a separate browser window.
3. In this window, go to the application's identification region and choose the *Adapt Configuration* link.
4. In the *Editor for Web Dynpro ABAP Components — Customizing* screen, choose *Change*.
5. On the *Component Customizing <application name>* screen, make sure that the *Component-Defined view* is on.

Substep B: Adding a List

1. In the preview, select the place where you would like to add a new list.
2. If no UIBB has been defined for a subview or step, choose *Attributes*.
3. If a UIBB has already been assigned, choose *Add UIBB* or *Add List UIBB* in the action area.

Now specify the UIBB as a list component by entering the following values in the attribute view:

1. In the *Component* field, enter `FPM_LIST_UIBB`.
   - If you have chosen *Add List UIBB*, check whether the field is already correctly prefilled.
2. In the *View* field, enter `LIST_WINDOW`.
   - If you have chosen *Add List UIBB*, check whether the field is already correctly prefilled.
3. In the *Configuration Name* field, enter a name for the list component configuration.
   - Although the *Configuration Name* field is not marked as a required entry, it is necessary to enter a name here. Otherwise, the *Configure UIBB* action is not activated.
   - The *Configuration Type*, *Configuration Variant*, and *Sequence Order* are optional.
4. Choose *Save*.

   The system has entered the list component as a new UIBB with the name `UIBB: LIST_WINDOW`.

Substep C: Configuring the List

1. Choose *Configure UIBB*.
2. In the *Editor for Web Dynpro ABAP Components — Configuration* screen, choose *Create*.
3. In the *Create Configuration* dialog window, enter a description and choose a package.
4. Choose OK.
5. In the Edit Feeder Class dialog window, enter the feeder class that you would like to assign the new list to.

You can search for a feeder class here.

To do this, choose the input help in the Edit Feeder Class window. The system opens the Select Feeder Class IF_FPM_GUIBB_LIST dialog window. You can search for a feeder class here. Choose Start Search. The system lists all feeder classes for the IF_FPM_GUIBB_LIST Web Dynpro interface. Select a feeder class and choose OK. The system copies the feeder class into the ObjectTypeName field.

6. Choose Edit Parameters.
7. In the Edit Parameters window, you can choose a value for the feeder class parameters.
8. Choose OK.

Result

The system opens the list editor for the new list.

Additional Information on the List Component

The following information is useful when configuring a List Component.

Attributes

In the hierarchy of the Component Configuration of your application, the following attribute is available for the List Component:

- Lead Selection Action Assignment: You can assign an FPM event ID to the lead selection here. If a lead selection occurs during runtime, the assigned FPM event is raised. If you assign no event ID, the generic event ID IF_FPM_GUIBB_LIST=>GC_FPM_EVENT_ON_LEAD_SEL is assigned.

In the hierarchy of the Component Configuration of your List Component, choose Settings to display the following attributes:

- Column count: Determines the amount of columns that are displayed at runtime
- Row count: Determines the amount of rows that are displayed at runtime
- Selection Event: Like a Web Dynpro table, the List Component offers two kinds of selection at runtime:
  - Lead selection (the user uses the left mouse button to select one single row)
  - Normal selection (the user uses the right mouse button to select one or more rows)
Using this dropdown list box, you can determine what kind of selection raises an FPM event. The default is a Lead Selection.

- **Selection Mode**: Determines whether it is possible to select multiple rows
- **Selection behavior**: Determines whether currently selected rows are de-selected when the user makes a new selection

**FPM Events and the List Component**

As the List Component is itself an FPM UIBB, it takes part, when it is visible, in each FPM event loop. The List Component may also raise FPM events itself. These events are raised from the following three sources:

- **Cell events**
  
The columns may contain fields that have a display type that are capable of raising an event (for example, a button display type). All cell-based events have the FPM event ID `IF_FPM_GUIBB_LIST=>GC_GUIBB_LIST_ON_CELL_ACTION`. The corresponding row and column values are added as event parameters to this FPM event:
  
  - `IF_FPM_GUIBB_LIST=>GC_EVENT_PAR_ROW`
  
  - `IF_FPM_GUIBB_LIST=>GC_EVENT_PAR_COLUMN_NAME`

- **Toolbar events**
  
  Almost each toolbar element may raise an FPM event. In this case, the event ID is the action ID (which was defined by the feeder class in method `get_definition`). Some toolbar elements may contain specific values of interest (for example user inputs), such as the toggle button, the input field and the dropdown list box. To get these values, you may read the following FPM event parameters:
  
  - `IF_FPM_GUIBB_LIST=>GC_EVENT_PAR_TOGGLE_STATE` (for the toggle button),
  
  - `IF_FPM_GUIBB_LIST=>GC_EVENT_PAR_INPUT_VALUE` (for the input field) or
  
  - `IF_FPM_GUIBB_LIST=>GC_EVENT_PAR_DROP_DOWN_KEY` (for the dropdown list box).

- **Selection events**
  
  A row selection may also raise an FPM event. It is possible to choose whether only a lead selection raises an FPM event or also a normal selection (see configuration settings for details).

**Rendering G UIBB List as ALV**

From FPM 702e onwards, there is an option to render the data in a List Component using the SAP List Viewer (ALV). This allows the end user to personalize the table and, amongst other things, to export data from the table to a spreadsheet, to use the print feature, and to sort and filter data.

For more information on the SAP List Viewer (ALV), see the SAP NetWeaver Library under SAP NetWeaver Developer’s Guide, Using ABAP, and Web Dynpro for ABAP, Web Dynpro ABAP: Development in Detail, and Integration.
Design Time Settings in the Configuration Editor

In the Component Configuration screen of the List Component Configuration Editor, choose Settings in the Hierarchy. In the Attributes section, the Rendering Style dropdown list displays the following options:

- **Standard Rendering**
  
  The data is displayed in a simple table format.

- **Render as Row-Repeater**
  
  The data is displayed in a single column format. Column headings which would appear once in a table format are repeated here to form a group; one group is displayed on top of another group in a single column.

- **Render as ALV**
  
  The data is displayed in an ALV table format with sorting, filtering, printing and personalization capabilities as well as the feature to print to Microsoft Office Excel. No other settings are required for ALV output; it is not necessary to make changes to the feeder class, nor to the configuration of the List Component.

Runtime Activities

If ALV rendering has been selected, the end user can, at runtime, choose the Settings icon in the ALV table and make changes to the table layout and save these changes. The changes appear as a new view option in the View dropdown list the next time the application is run.

Important Points to Note

- Rendering a list with ALV consumes considerably more memory than rendering with the List Component. Therefore, it is recommended to render with ALV only when necessary.
- The following drawback should be taken into consideration. If ALV rendering is selected, it should never be deselected in a later delivery. If the end user personalizes an ALV table, the changes are lost when ALV is switched off in a later delivery.
- Note that it is possible to print table data to a spreadsheet without rendering the table in ALV format. In the Configuration Editor of the List Component, choose Settings in the Hierarchy. In the Attributes section, select the Export to Excel checkbox. This provides an Export List button above the table at runtime.
- Data handling and formatting in ALV may not be exactly the same as with the List Component, due to the technical differences between ALV tables and WD tables (for example, the event names are not the same for the two tables).
- It is recommended (but not mandatory) to use separate configurations for the List Component and ALV to avoid possible discrepancies. Due to the point above, applications might code in the feeder class specific to an event ID and parameters. Switching between the two modes for the same configuration might lead to discrepancies. The List Component and ALV are also different in terms of the features...
they offer. It is therefore best if they are separate configurations to avoid features being available in one and not available in the other for the same table.

**Hierarchical List Component (GUIBB TREE)**

A generic design template for displaying data in a hierarchical list or tree that is implemented using the Web Dynpro component FPM_TREE_UIBB. You use this design template in application-specific views (UIBB) where you want to display data in a hierarchical list or tree. You can determine the concrete display of the data in a tree when configuring the Web Dynpro component FPM_TREE_UIBB.

**Structure**

A hierarchical list is structured as follows:

- **Master Column**
  The master column displays all the items in a list. When the system first displays a table, each top-level, parent item in the master column is preceded by an Expand or Collapse icon, which allows you to see the child (sub) items contained within it. A top-level item in the master column without child items has no Expand or Collapse icon preceding it.
  
The master column is always visible; you cannot hide this column.
  
The master column is always displayed as the first column in a table; you cannot move its position in a table.
  
The hierarchical list component permits incremental loading of data, meaning that data relating to child-lists can be loaded into the application when the node is expanded.

- **Non-master columns**
  These columns display the details of each list item.

- **Rows**
  Each item in a list is displayed in a separate row.

- **Toolbar**
  A toolbar displays the Collapse All and Expand All buttons (if selected) and other buttons that you have created.

The component-defined view gives you the opportunity to specify the following:

- The hierarchy pattern for the master column of the tree
- The data is displayed in each column
- The display type (such as display field or input field) used for each column (except the master column)
- The order the columns are arranged in.
- The number of columns and rows that can be displayed in the view at one time

The data of a tree that can be displayed is determined by the feeder class that is assigned to the configuration of the Web Dynpro component FPM_TREE_UIBB.
Integration
You can configure a tree component using the Hierarchical List Editor for Floorplan Manager.

IF_FPM_GUIBB_TREE Interface

The following tables describe the methods (and their attributes) of the IF_FPM_GUIBB_TREE interface. If your application does not need a particular method, implement an empty method, otherwise the system will dump.

Note
You must implement at least the following methods:

- GET_DEFINITION
- GET_DATA

Methods of the IF_FPM_GUIBB_TREE Interface

GET_DEFINITION:
Allows the feeder to provide all necessary information for configuring a tree: the list of available fields and their properties and the list of actions (FPM events).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EO_FIELD_CATALOG</td>
<td>Is of type CL_ABAP_TABLEDESCR. The components of this object are the available fields. The simplest way to provide a field catalog is to create a flat DDIC structure containing all the fields and then get the field catalog via eo_field_catalog ?= CL_ABAP_TABLEDESCR=&gt;describe_by_name(&lt;name&gt; )</td>
</tr>
<tr>
<td></td>
<td>Note The tree GUIBB supports only flat structures. When using deep structures, only the highest level fields are available.</td>
</tr>
<tr>
<td>ET_FIELD_DESCRIPTION</td>
<td>This mandatory parameter is used to inform FPM which fields from the field catalog are to be used for which purpose in the tree. This is</td>
</tr>
</tbody>
</table>
achieved with the help of the field COLUMN_TYPE (see note and table below).

The ET_FIELD_DESCRIPTION parameter can also be used to provide additional properties for various columns:

**LABEL_BY_DDIC**: Indicates whether a column header text should be taken from DDIC or not.

Search helps: DDIC search help and OVS search help.

**HEADER_TEXT_WRAPPING**: Determines whether the column header text shall be wrapped or not.

Attributes for read-only, mandatory, enabled and visibility.

Attributes for enabling filtering and sorting, for formatting amongst others.

<table>
<thead>
<tr>
<th><strong>ET_ACTION_DEFINITION</strong></th>
<th>A list of all actions (which will be transformed to FPM events at runtime) that you can assign to the tree elements.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ET_SPECIAL_GROUPS</strong></td>
<td>This provides you with the same options that you have in the ABAP ALV (see function module REUSE_ALV_GRID_DISPLAY) to group the fields within your field catalogue. You must enter the special group for each field in the field description table in field SP_GROUP. At design-time, the FPM Configuration Editor groups the fields together, providing you with an easier method for finding fields.</td>
</tr>
</tbody>
</table>

**COLUMN_TYPE**: To create a hierarchical list (tree), you must establish the hierarchical relationship between records using the application data. To do this, the fields in the following table are required. This information is passed in the GET_DEFINITION method of the feeder class. From the field catalog provided in this method, you must select the fields which you want to use to determine the hierarchy. The field COLUMN_TYPE in the ET_FIELD_DESCRIPTION is used to provide this information. The following table explains the various column types.
### Field/Column Type

<table>
<thead>
<tr>
<th>Field/Column Type</th>
<th>Explanation</th>
<th>Optional</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Key</td>
<td>This column of a table contains the parent element in the hierarchy at any level.</td>
<td>No</td>
<td>Any</td>
</tr>
<tr>
<td>Row Key</td>
<td>This column of a table contains the row/child element in the hierarchy at any level.</td>
<td>No</td>
<td>Any</td>
</tr>
<tr>
<td>Expanded</td>
<td>This column of the table determines whether the parent node is expanded or not.</td>
<td>No</td>
<td>Boolean</td>
</tr>
<tr>
<td>Is Leaf</td>
<td>This column of the table determines whether the element is the last node in the hierarchy.</td>
<td>Yes</td>
<td>Boolean</td>
</tr>
<tr>
<td>Children Loaded</td>
<td>This column helps in stopping a backend call every time the same node is opened.</td>
<td>Yes</td>
<td>Boolean</td>
</tr>
<tr>
<td>Text</td>
<td>This column determines the text which needs to be rendered on the UI for the tree column.</td>
<td>Yes</td>
<td>String</td>
</tr>
<tr>
<td>Image</td>
<td>This column contains the string for an icon if you want to display one in the master column.</td>
<td>Yes</td>
<td>String</td>
</tr>
</tbody>
</table>

### GET_PARAMETER_LIST:

This method is called at design time and allows you to define a list of the parameters that the feeder class supports. This list is used by the FPM Configuration Editor to provide the input fields for these parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT_PARAMETER_DESCR</td>
<td>Is returned from this method. It describes which parameter is possible. In Field TYPE, the DDIC type needs to be entered.</td>
</tr>
</tbody>
</table>

### INITIALIZE:

Called at runtime when the tree is created. It is the first feeder method which is called from FPM.

| IT_PARAMETER                 | Contains a list of the feeder parameters and the values for them specified in the configuration. |

### FLUSH:

The first feeder method which is called during an event loop. Whenever an FPM event is triggered, this method is called (this includes all round trips caused by the list itself). You can use it to forward changed tree data to other components in the same application.

<table>
<thead>
<tr>
<th>IT_CHANGE_LOG</th>
<th>Lists all changes made by the user.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_DATA</td>
<td>A structure containing the changed data.</td>
</tr>
</tbody>
</table>
### PROCESS_EVENT:

Called within the FPM event loop, it forwards the FPM PROCESS_EVENT to the feeder class. Here the event processing takes place and this is where the event can be canceled or deferred.

<table>
<thead>
<tr>
<th>IO_EVENT</th>
<th>The FPM event which is to be processed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV_RESULT</td>
<td>The result of the event processing. There are 3 possible values:</td>
</tr>
<tr>
<td></td>
<td>- ev_result = if_fpm_constants=&gt;gc_event_result-OK</td>
</tr>
<tr>
<td></td>
<td>- ev_result = if_fpm_constants=&gt;gc_event_result-FAILED</td>
</tr>
<tr>
<td></td>
<td>- ev_result = if_fpm_constants=&gt;gc_event_result-DEFER</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>A list of messages which shall be displayed in the message region.</td>
</tr>
</tbody>
</table>

### GET_DATA:

Called within the FPM event loop, it forwards the FPM PROCESS_BEFORE_OUTPUT event to the feeder class. Here you specify the tree data after the event has been processed.

<table>
<thead>
<tr>
<th>IO_EVENT</th>
<th>The FPM event which is to be processed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_SELECTED_FIELDS</td>
<td>The list of fields necessary for the tree rendering. Provide only the data for the fields listed in this table; all other fields are neither visible at runtime nor used as reference for visible fields. The master column, however, is a combination of multiple fields of the field catalog and therefore the master column cannot be explicitly found in this parameter.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>A list of messages which shall be displayed in the message area.</td>
</tr>
<tr>
<td>EV_DATA_CHANGED</td>
<td>For performance reasons, the GUIBB adjusts the data in the tree only if the data has been changed. To indicate this, set this flag whenever you change the data to be displayed within this feeder.</td>
</tr>
<tr>
<td>EV_FIELD_USAGE_CHANGED</td>
<td>Indicates whether or not the field usage has been changed by this method. If you change the field usage without setting this flag to ‘X’, your changes are ignored.</td>
</tr>
<tr>
<td>EV_ACTION_USAGE_CHANGED</td>
<td>Indicates whether or not the action usage has been changed. Use an ‘X’ to indicate whether you changed the action usage. If you do not, your changes are ignored.</td>
</tr>
<tr>
<td>CT_DATA</td>
<td>The tree data to be changed. This is the actual</td>
</tr>
</tbody>
</table>
data that gets rendered on the screen.

CT_FIELD_USAGE

Field usage to change. The field usage consists of the field attributes which might change at runtime (for example enabled or disabled, visible or invisible, mandatory or optional, read-only or edit). Use it to control the properties of columns.

If you change the fixed values of a field, set the flag FIXED_VALUES_CHANGED for this field.

See also the note below on assigning attributes to single cells or whole columns.

CT_ACTION_USAGE

Action usage to change. The action usage consists of the attributes related to actions which might change at runtime (for example enabled or disabled, visible or invisible, mandatory or optional, read-only or edit). Use it to control the properties of toolbars. If an action is rendered as a button, the visibility setting (for example) of the button is defined here.

See also the note below on assigning attributes to single cells or whole columns.

CS_TREE_ATTRIBUTES

Use this parameter to specify the header and the tooltip for the master column in the tree.

Regarding columns, note that it is possible to assign the attributes for CT_FIELD_USAGE and CT_ACTION_USAGE either to single cells or to whole columns. If you want to set these attributes for the whole column, use the corresponding fields in the FIELD_USAGE structure. If you would like to set these attributes for single cells, proceed as follows:

1. Create a new column for the table (add a field to the field catalog in the GET_DEFINITION method).
2. Define the column as a technical column that is not visible at runtime, by setting the field TECHNICAL_FIELD to ‘X’. This column contains the properties of the cells.
3. In the GET_DEFINITION method, adjust the field description accordingly. For example, you want to set the property READ_ONLY for single cells in column A. For this reason, you create a technical column B. In the field description, you set READ_ONLY_REF to B.

GET_DEFAULT_CONFIG:

Call this if you want to have a default configuration. Use it to call pre-configured tree configurations when a user starts the FPM Configuration Editor. This avoids the user, who uses a feeder class to create a tree, having to create it again from the beginning.
**Parameter** | **Description**
--- | ---
IO_LAYOUT_CONFIG | It is of type `IF_FPM_GUIBB_TREE_CONFIG` which provides the API to create a default configuration.

### CHECK_CONFIG:

Call this if you want to make your own application-specific checks on the configuration in the FPM Configuration Editor immediately before saving.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_LAYOUT_CONFIG</td>
<td>It is of type <code>IF_FPM_GUIBB_TREE_CONFIG</code> which provides the API to read the configuration to be saved.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>A list of messages which shall be displayed in the message region.</td>
</tr>
</tbody>
</table>

---

**Hierarchical List Editor for Floorplan Manager**

You use the hierarchical list component editor to configure hierarchical list components (see [Hierarchical List Component](#)). Using the hierarchical list component editor, you can adjust a hierarchical list within an application to your specific business requirements.

Most attributes for a hierarchical list are defined in the feeder class assigned to the configuration of a hierarchical list component. However, you may change some of these attributes in the hierarchical list component editor. Changes that you make in the component editor take precedence over attributes defined in the feeder class.

**Features**

The hierarchical list component editor consists of the following work areas:

- **Preview**
  The preview displays the hierarchical list component in the current configuration, providing you with a picture of the layout of the hierarchical list.

- **Hierarchy**
  All hierarchical list component elements (for example, columns or toolbars) are displayed in the Hierarchy as a tree structure.

- **Attribute view**
  Attributes of the currently selected list element in the Hierarchy, which can be changed using the hierarchical list component editor, are displayed in the attribute view.

The hierarchical list component editor is similar to the list component editor and contains several element attributes or buttons that are not discussed here. For more information about these element attributes and buttons, see [List Component Editor for Floorplan Manager](#).

The hierarchical list component editor provides you with the following actions:

- **Edit Feeder Class**
• Edit Parameters
• Configure Toolbar
• Configure Columns

This button provides you with the following functions:
• Add a Column
  You can select a column from a list of available columns and determine the column header and the column display type (input field, checkbox for example). The list of available columns is defined in the field catalog of the feeder class assigned to the configuration of the hierarchical list component.
• Remove a Column
• Change the Column Sequence

Configure Columns, together with the attributes of the individual column elements in the Hierarchy, provides you with ample opportunity for customizing the non-master columns.

The hierarchical list component editor also provides you with a toolbar, allowing you to add the following toolbar elements:

• Expand All Nodes and Collapse All Nodes buttons
  Select the Default Button Set is Included checkbox in the attributes section of the Tree element in the Hierarchy to add these buttons. You might, on occasion, want to hide these buttons; a very large list could cause performance problems when these two buttons are chosen by the user at runtime.

• Other toolbar elements
  Choose Configure Toolbar to add various other toolbar elements (for example, separators, toggle buttons, and dropdown lists) to the toolbar.

Additional element attributes from the hierarchical list component editor, which you can edit, include:

• Column Header and Tooltip for the MasterColumn element
  You can include your own master column header and tooltip and override text defined by the feeder class.
• Default Button Set is Included
  For information on this, see above.
• Tree Width
  Under Settings, you can set the width of the entire list.

In addition, the Component-Defined processing view is pre-set. Make sure that this view is selected before configuring a tree component.

The Hierarchical List Editor launches from the configuration editor for Floorplan Manager automatically when you launch the configuration of an application-specific view (UIBB) that uses the FPM_TREE_UIBB Web Dynpro component.
Add Tree

You can add an additional, application-specific view to a Floorplan Manager application at any time in the form of a new tree. Depending on the floorplan and the application-specific views already embedded, you can position a new tree component in one of the following ways:

- In a sub-view of an object instance floorplan
- In a main step of a guided activity floorplan
- In a sub-step of a guided activity floorplan
- In a content area in the overview page floorplan
- In a tabbed component in a MASTER UIBB
- In a tabbed component in a TAB UIBB

Prerequisites: If you want to add a new tree, you must assign a feeder class to the tree. The feeder class must first be created and programmed.

Procedure

Substep A: Calling up the configuration editor of Floorplan Manager

1. Select a Web Dynpro application configuration in the Object Navigator of the ABAP Workbench.
2. On the Web Dynpro Explorer: Display Web Dynpro Configuration screen, choose Web Dynpro Configuration Test Execute. The Web Dynpro application is launched in a separate browser window.
3. In this window, go to the application's identification region and choose the Adapt Configuration link.
4. In the Editor for Web Dynpro ABAP Components — Customizing screen, choose Change.
5. On the Component customizing <application name> screen, make sure that the Component-Defined view is on.

Substep B: Adding a Tree

1. In the preview, select the place where you would like to add a new tree.
2. If no UIBB has been defined for a subview or step, choose Attributes.
3. If a UIBB has already been assigned, choose Add UIBB or Add Tree Component in the action area.
4. Specify the UIBB as a tree component by entering the following values in the attribute view:
   a. In the Component field, enter FPM_TREE_UIBB. If you have chosen Add Tree Component, check whether the field is already correctly prefilled.
   b. In the View field, enter TREE_WINDOW. If you have chosen Add Tree Component, check whether the field is already correctly prefilled.
   c. In the Configuration Name field, enter a name for the tree component configuration.
Although the Configuration Name field is not marked as a required entry, it is necessary to enter a name here. Otherwise, the Configure UIBB action is not activated. The Configuration Type, Configuration Variant, and Sequence Order are optional.

5. Choose Save. The system has entered the tree component as a new UIBB with the name UIBB: TREE_WINDOW.

Substep C: Configuring the Tree

1. Choose Configure UIBB
2. In the Editor for Web Dynpro ABAP Components — Configuration screen, choose Create.
3. In the Create Configuration dialog window, enter a description and choose a package.
4. Choose OK.
5. In the Edit Feeder Class dialog window, enter the feeder class that you would like to assign the new tree to.

You can search for a feeder class here.
To do this, choose the input help in the Edit Feeder Class window. The system opens the Select Feeder Class IF_FPM_GUIBB_TREE dialog window. You can search for a feeder class here. Choose Start Search. The system lists all feeder classes for the IF_FPM_GUIBB_TREE Web Dynpro interface. Select a feeder class and choose OK. The system copies the feeder class into the ObjectTypeName field.

6. Choose Edit Parameters.
7. In the Edit Parameters window, you can choose a value for the feeder class parameters.
8. Choose OK.
   The system opens the Hierarchical List Editor for the new tree.

Additional Information on the Hierarchical List Component

The following information is useful when configuring a Hierarchical List (Tree) Component.

Attributes

In the hierarchy of the Component Configuration of your application, the following attribute is available for the Hierarchical List Component:

- **Lead Selection Action Assignment**: You can assign an FPM event ID to the lead selection here. If a lead selection occurs during runtime, the assigned FPM event is raised. If you assign no event ID, the generic event ID IF_FPM_GUIBB_TREE=>GC_FPM_EVENT_ON_LEAD_SEL is assigned.
In the hierarchy of the Component Configuration of your Hierarchical List Component, choose *Settings* to display the following attributes:

- **Column count**: Determines the amount of columns that are displayed at runtime
- **Row count**: Determines the amount of rows that are displayed at runtime
- **Selection Event**: Like a Web Dynpro table, the Hierarchical List Component offers two kinds of selection at runtime:
  
  i) **Lead selection** (the user uses the left mouse button to select one single row)
  
  ii) **Normal selection** (the user uses the right mouse button to select one or more rows)
  
  Using this dropdown list box, you can determine what kind of selection raises an FPM event. The default is a Lead Selection.

- **Selection Mode**: Determines whether it is possible to select multiple rows
- **Selection behavior**: Determines whether currently selected rows are deselected when the user makes a new selection

**FPM Events and the Hierarchical List Component**

As the Hierarchical List Component is itself an FPM UIBB, it takes part, when it is visible, in each FPM event loop. The Hierarchical List component may also raise FPM events itself. These events are raised from the following three sources:

- **Cell events**
  The columns may contain fields that have a display type that are capable of raising an event (for example, a button display type). All cell-based events have the FPM event ID `IF_FPM.GUIBB_TREE=>GC_GUIBB_TREE_ON_CELL_ACTION`. The corresponding row and column values are added as event parameters to this FPM event
  
  `IF_FPM.GUIBB_TREE=>GC_EVENT_PAR_ROW` and
  
  `IF_FPM.GUIBB_TREE=>GC_EVENT_PAR_COLUMN_NAME`.

- **Toolbar events**
  Almost each toolbar element may raise an FPM event. In this case, the event ID is the action ID (which was defined by the feeder class in method `GET_DEFINITION`). Some toolbar elements may contain specific values of interest (for example user inputs), such as the toggle button, the input field and the dropdown list box. To get these values, you may read the following FPM event parameters
  
  `IF_FPM.GUIBB_TREE=>GC_EVENT_PAR_TOGGLE_STATE` (for the toggle button),
  
  `IF_FPM.GUIBB_TREE=>GC_EVENT_PAR_INPUT_VALUE` (for the input field) or
  
  `IF_FPM.GUIBB_TREE=>GC_EVENT_PAR_DROP_DOWN_KEY` (for the dropdown list box).

  There is a default set of buttons in the toolbar for applications. This button set contains two buttons which are the *Expand All* and the *Collapse All* buttons which control the
display content of the tree nodes. Once the default button set is configured, the applications themselves must handle the event IDs of these buttons in the feeder class.

- **Selection events:** A row selection may also raise an FPM event. It is possible to choose whether only a lead selection raises an FPM event or also a normal selection (see configuration settings for details).

**Search Component (GUIBB SEARCH)**

A generic design template for displaying a search query which is implemented using the Web Dynpro component 

You can determine the search query by configuring the Web Dynpro component 

**Structure**

A search component is comprised of the following objects:

- **Search Attributes**

  This are the attributes that a user can use to build up a search query, e.g. cost center, personnel number or area code, etc. Each search attribute has a certain ABAP data type. From this ABAP data type a certain meta type is derived. The existing predefined Meta types are: Text, alpha numeric, numeric, date, enumeration, and Boolean.

- **Search Operators**

  This are operators such as is, is greater than, or is between that the user can combine with search attributes in order to build up the query. Each search attribute gets a default set of search operators assigned. The content of that set depends on the meta-type of the search attribute. The table below shows the search operators that are available for the different search attribute meta-types:

<table>
<thead>
<tr>
<th>Search Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Is, is not, is empty, starts with, contains</td>
</tr>
<tr>
<td>Alpha numeric</td>
<td>Equals, does not equal, is greater than, is less than, is between, is empty, contains</td>
</tr>
<tr>
<td>Numeric</td>
<td>Equals, does not equal, is greater than, is less than, is between, is empty, contains</td>
</tr>
<tr>
<td>Date</td>
<td>Is, is not, is between, is earlier than, is later than</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Is, is not, is empty (in cases where the dropdown box contains an empty entry)</td>
</tr>
</tbody>
</table>
- **Search Values**

  These are the values that are used for searching objects. There one input field where the user can enter this value. However if the user chooses the search operator “is between” two input fields are displayed in order to build up a range.

- **Search Row**

  A search row is a combination of one search attribute plus one search operator plus one search value (or two in case of a range).

- **Search Query**

  A search query consists of one or several search rows.

**Integration**

You can configure a search component using the [Search Component Editor](#) for Floorplan Manager.

### IF_FPM_GUIBB_SEARCH Interface

The following tables describe the methods of the interface `IF_FPM_GUIBB_SEARCH`.

If your application does not need a particular method, implement an empty method, otherwise the system will dump.

![Icon](#)

You must implement at least the following methods:

- `GET_DEFINITION`
- `PROCESS_EVENT`
- `GET_DATA`

**Methods**

**GET_DEFINITION:**

Allows the feeder to provide all necessary information for configuring a search: A list of available search attributes and optionally a list of columns for the result table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EO_FIELD_CATALOG_ATTR</td>
<td>Is of type <code>CL_ABAP_STRUCTDESCR</code>. The components of this object are the available search attributes. The simplest way to provide a field catalog is to create a flat</td>
</tr>
</tbody>
</table>
### GET_DEFINITION:

Allows the feeder to provide all necessary information for configuring a search: A list of available search attributes and optionally a list of columns for the result table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDIC structure containing all search attributes and then get the field catalog via <code>eo_field_catalog_attr ?= CL_ABAP_STRUCTDESCR=&gt;describe_by_name(&lt;name&gt;)</code></td>
<td></td>
</tr>
<tr>
<td>The search GUIBB supports only flat structures. When using deep structures, only the highest level fields are available.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional only: Here you can provide additional information for search attributes, for example F4 helps or input field data format. You can also change the set of search operators that are assigned to a search attribute.</td>
<td></td>
</tr>
<tr>
<td>Contains the columns that are possible to choose from during configuration. This is optional only.</td>
<td></td>
</tr>
<tr>
<td>Optional only: Here you can provide additional information for the columns of the result table, e.g. the column text.</td>
<td></td>
</tr>
<tr>
<td>This parameter determines the selection mode of the result table, such single line selection or multiple line selection.</td>
<td></td>
</tr>
</tbody>
</table>

### GET_PARAMETER_LIST:

Called at design time and allows you to define a list of the parameters that the feeder class supports. This list is used by the FPM Configuration Editor to provide the input fields for these parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is returned from this method. It describes which parameters are possible. In Field TYPE, the DDIC type needs to be entered.</td>
<td></td>
</tr>
</tbody>
</table>

### INITIALIZE:

Called at runtime when the search UIBB is created. It is the first feeder method which is called from FPM.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains a list of the feeder parameters and the values for them specified in the configuration.</td>
<td></td>
</tr>
</tbody>
</table>
The first feeder method which is called during an event loop. Whenever an FPM event is triggered (this includes all round trips caused by the search UIBB itself) this method is called. It tells the relevant user input data of the search UIBB.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_FPM_SEARCH_CRITERIA</td>
<td>Contains the actual search criteria</td>
</tr>
<tr>
<td>IV_MAX_NUM_RESULTS</td>
<td>Contains what the amount of maximum number of result objects.</td>
</tr>
<tr>
<td>IT_SELECTED_LINES_OF_RESULT</td>
<td>Contains what lines within the result table are currently selected.</td>
</tr>
</tbody>
</table>

 Called within the FPM event loop, it forwards the FPM PROCESS_EVENT event to the feeder class. This method should be used for actually conducting the search. But before doing so you should check for the ID IF_FPM_GUIBB_SEARCH=>FPM_EXECUTE_SEARCH event. This FPM event is raised as soon as the user presses the search button.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_EVENT</td>
<td>The FPM event which is to be processed.</td>
</tr>
<tr>
<td>EV_RESULT</td>
<td>The result of the event processing. There are 3 possible values:</td>
</tr>
<tr>
<td></td>
<td>ev_result =</td>
</tr>
<tr>
<td></td>
<td>if_fpm_constants=&gt;gc_event_result-OK</td>
</tr>
<tr>
<td></td>
<td>ev_result =</td>
</tr>
<tr>
<td></td>
<td>if_fpm_constants=&gt;gc_event_result-FAILED.</td>
</tr>
<tr>
<td></td>
<td>ev_result =</td>
</tr>
<tr>
<td></td>
<td>if_fpm_constants=&gt;gc_event_result-DEFER</td>
</tr>
<tr>
<td>IT_FPM_SEARCH_CRITERIA</td>
<td>The actual search criteria.</td>
</tr>
<tr>
<td>IV_RAISED_BY_OWN_UI</td>
<td>Tells whether the event was raised by the actual search UIBB or by some other UIBB.</td>
</tr>
<tr>
<td>IV_MAX_NUM_RESULTS</td>
<td>Tells the maximum number of found objects to be displayed in the result table.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>The feeder class may return messages via this parameter. As usual the messages will be displayed within the FPM message area.</td>
</tr>
</tbody>
</table>

 Called within the FPM event loop and forwards the FPM PROCESS_BEFORE_OUTPUT event to the feeder class. The main purpose of this method is to transport data from the feeder class to the search UIBB.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_EVENT</td>
<td>The FPM event which is to be processed.</td>
</tr>
<tr>
<td>IV_RAISED_BY_OWN_UI</td>
<td>Tells whether the event was raised by the actual search UIBB or by some other UIBB.</td>
</tr>
<tr>
<td>IT_VISIBLE_ATTRIBUTES</td>
<td>Tells what search attributes are currently visible on the UI.</td>
</tr>
<tr>
<td>IT_SELECTED_COLUMNS_OF_RESULT</td>
<td>Tells the columns that have been configured for the</td>
</tr>
</tbody>
</table>
PROCESS_EVENT:

Called within the FPM event loop, it forwards the FPM PROCESS_EVENT event to the feeder class. This method should be used for actually conducting the search. But before doing so you should check for the ID IF_FPM_GUIBB_SEARCH=>FPM_EXECUTE_SEARCH event. This FPM event is raised as soon as the user presses the search button.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_SELECTED_SEARCH_ATTRIBUTES</td>
<td>Tells the search attributes that have been configured.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>The feeder class may return messages via this parameter. As usual the messages will be displayed within the FPM message area.</td>
</tr>
<tr>
<td>EV_SEARCH_CRITERIA_CHANGED</td>
<td>Contains the current search criteria.</td>
</tr>
<tr>
<td>ET_RESULT_LIST</td>
<td>If the application (feeder class) chooses to let the search UIBB render the result table then it needs to inform the search UIBB about content of the result via this parameter (but only if the event ID is IF_FPM_GUIBB_SEARCH=&gt;FPM_EXECUTE_SEARCH).</td>
</tr>
<tr>
<td>CT_FPM_SEARCH_CRITERIA</td>
<td>Tells the search UIBB whether the search criteria have been changed. If so then the search criteria will be updated accordingly.</td>
</tr>
</tbody>
</table>

GET_DEFAULT_CONFIG:

Call this if you want to have a default configuration. Use it to call pre-configured form configurations when a user starts the FPM configuration editor. This avoids the user, who uses a feeder class to create a search having to create it from scratch.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_LAYOUT_CONFIG</td>
<td>Of type IF_FPM_GUIBB_SEARCH_CONFIG: This object provides the API to create a default configuration.</td>
</tr>
</tbody>
</table>

CHECK_CONFIG:

Call this if you want to make your own application-specific checks on the configuration in the FPM Configuration Editor immediately before saving.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_LAYOUT_CONFIG</td>
<td>Of type IF_FPM_GUIBB_SEARCH_CONFIG: This object provides the API to read the configuration to be saved.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>A list of messages which shall be displayed in the message region.</td>
</tr>
</tbody>
</table>

Search Component Editor for Floorplan Manager

This information is provided soon.

Add Search

You can add an additional, application-specific view to a Floorplan Manager application at any time in the form of a new search. Depending on the floorplan and the application-specific views already embedded, you can position a new search component in one of the following ways:
• in a subview of an object instance
• in a main step of a guided activity
• in a substep of a guided activity
• in a tabbed component in a master UIBB
• in a tabbed component in a tab UIBB

Prerequisites

If you would like to add a new search form, you must assign a feeder class to the search. This feeder class must first be created and programmed.

Procedure

Substep A: Calling up the configuration editor of Floorplan Manager

1. Select a Web Dynpro application configuration in the Object Navigator of the ABAP Workbench.
3. The Web Dynpro application is launched in a separate browser window.
4. In this window, go to the application's identification region and choose the Adapt Configuration link.
5. In the Editor for Web Dynpro ABAP Components — Customizing screen, choose Change.
6. On the Component Customizing <application name> screen, make sure that the Component-Defined view is on.

Substep B: Adding a Search

1. In the preview, select the place where you would like to add a new search.
2. If no UIBB has been defined for a subview or step, choose Attributes.
3. If a UIBB has already been assigned, choose Add UIBB or the new possibility Add Search UIBB in the action area.

Now specify the UIBB as a search component by entering the following values in the attribute view:

1. In the Component field, enter FPM_SEARCH_UIBB.
2. In the View field, enter SEARCH_WINDOW.
3. In the Configuration Name field, enter a name for the list component configuration.

Although the Configuration Name field is not marked as a required entry, it is necessary to enter a name here. Otherwise, the Configure UIBB action is not activated.
The Configuration Type, Configuration Variant, and Sequence Order are optional.

4. Choose Save.

The system has entered the list component as a new UIBB with the name UIBB:
SEARCH_WINDOW.

Substep C: Configuring the Search

1. Choose Configure UIBB.
2. In the Editor for Web Dynpro ABAP Components — Configuration screen, choose Create.
3. In the Create Configuration dialog window, enter a description and choose a package.
4. Choose OK.
5. In the Edit Feeder Class dialog window, enter the feeder class that you would like to assign to the new search.

You can search for a feeder class here.

To do this, choose the input help in the Edit Feeder Class window. The system opens the Select Feeder Class IF_FPM_GUIBB_SEARCH dialog window. You can search for a feeder class here. Choose Start Search. The system lists all feeder classes for the IF_FPM_GUIBB_SEARCH Web Dynpro interface. Select a feeder class and choose OK. The system copies the feeder class into the ObjectTypeName field.

6. Choose Edit Parameters.
7. In the Edit Parameters window, you can choose a value for the feeder class parameters.
8. Choose OK.

Result

The system opens the search component editor for the new search.

Launchpad Component (GUIBB LAUNCHPAD)

A generic design template for displaying an overview or navigation block that is implemented using the Web Dynpro component FPM_LAUNCHPAD_UIBB.

Structure

A navigation block is comprised of the following components:

- Main view
The main view on the left side can be shown as a Map, Directory, or Index.

The Map view shows a list of applications as displayed in the following example. There is a link with a large icon at the beginning of each group. This link has an optional description. If you select this link, you navigate to the appropriate site. Below this link are optional quick links. Quick links are part of the corresponding area page and configured as visible in the Map view. If you choose one of the quick links, you navigate to the application that was customized for this quick link.

Example of a Map view

![Map view example](image)

The Directory view displays all entries of the launchpad, also grouped in the same way as in the Map view, but subgroups are grouped by a LinkChoice UI element.

Example of a Directory view:
The *Index* view is an alphabetical list of all applications of a launchpad. There is the possibility to search for a specific string or application.

**Example of an Index view**

- **Most Frequently Used**

  On the right side is a view for the most frequently used applications. The list of applications depends on the applications that a user selects from the left side. If an application is selected very often by a user, it has a better ranking than an application that is chosen less often. The list can be prefilled by choosing the *Prefill Most Frequently Used* button in the launchpad customizing. In the *Available Applications* dialog box, an administrator can select the application to pre-fill the list.
• Related Links

This view contains applications that are useful for your daily work. It is a user independent static list of applications and is defined by an administrator.

**IF_FPM_GUIBB_LAUNCHPAD Interface**

The following tables describe the methods of the **IF_FPM_GUIBB_LAUNCHPAD** interface.

If your application does not need a particular method, implement an empty method, otherwise the system will dump.

If you do not need any dynamic changes, you do not need to implement this interface.

**Methods of IF_FPM_GUIBB_LAUNCHPAD Interface**

<table>
<thead>
<tr>
<th>Generated Code</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF_FPM_GUIBB_LAUNCHPAD</td>
<td>OVERWRITE_LAUNCHPAD_KEY</td>
<td>Called at runtime and before the Launchpad customizing, defined in the configuration, is read. Allows the feeder to specify another launchpad customizing. Depending on the parameter EV_ALSO_CALLED_IN_PBO, this method is also processed in Process Before Output.</td>
</tr>
<tr>
<td></td>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>IV_ROLE</td>
<td>Role of a launchpad.</td>
<td></td>
</tr>
<tr>
<td>IV_INSTANCE</td>
<td>Instance of a launchpad.</td>
<td></td>
</tr>
<tr>
<td>IR_EVENT</td>
<td>FPM event</td>
<td></td>
</tr>
<tr>
<td>EV_ALSO_CALLED_IN_PBO</td>
<td>Flag if this method is processed also in Process Before Output</td>
<td></td>
</tr>
<tr>
<td>EV_ANYTHING_CHANGED_IN_PBO</td>
<td>Must be set to abap_true if anything was changed in Process Before Output</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generated Code</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF_FPM_GUIBB_LAUNCHPAD</td>
<td>MODIFY</td>
<td>Called at run time before the content of the launchpad is displayed. It allows you to change some parts of the launchpad customizing. You can, for example, change the link text or set an application defined by a user parameter. Depending on the parameter EV_ALSO_CALLED_IN_PBO, this method is also processed in Process Before Output.</td>
</tr>
<tr>
<td></td>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>IR_NAVIGATION</td>
<td>A reference to the <strong>IF_FPM_NAVIGATION</strong> interface.</td>
<td></td>
</tr>
<tr>
<td>IR_EVENT</td>
<td>FPM event</td>
<td></td>
</tr>
<tr>
<td>EV_ALSO_CALLED_IN_PBO</td>
<td>Flag if this method is processed also in Process Before Output</td>
<td></td>
</tr>
<tr>
<td>EV_ANYTHING_CHANGED_IN_PBO</td>
<td>Must be set to abap_true if anything was changed in Process Before Output</td>
<td></td>
</tr>
</tbody>
</table>
OVERWRITE_REL_LINKS_LPDM_KEY:

Called at runtime, before the Launchpad customizing for Related Links, defined in the configuration, is read. Allows the feeder to specify another launchpad customizing for Related Links. Depending on the parameter EV_ALSO_CALLED_IN_PBO, this method is also processed in Process Before Output.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV_ROLE</td>
<td>Role of a launchpad.</td>
</tr>
<tr>
<td>IV_INSTANCE</td>
<td>Instance of a launchpad.</td>
</tr>
<tr>
<td>IR_EVENT</td>
<td>FPM event</td>
</tr>
<tr>
<td>EV_ALSO_CALLED_IN_PBO</td>
<td>Flag if this method is processed also in Process Before Output</td>
</tr>
<tr>
<td>EV_ANYTHING_CHANGED_IN_PBO</td>
<td>Must be set to abap_true if anything was changed in Process Before Output</td>
</tr>
</tbody>
</table>

MODIFYRELATED_LINKS:

Called at run time before the content of the launchpad for Related Links is displayed. It allows you to change some parts of the launchpad customizing for Related Links. You can, for example, change the link text or set an application as defined by a user parameter. Depending on the parameter EV_ALSO_CALLED_IN_PBO, this method is also processed in Process Before Output.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR_NAVIGATION</td>
<td>Reference to the IF_FPM_NAVIGATION interface. For further information about this interface, see the FPM Cookbook.</td>
</tr>
<tr>
<td>IR_EVENT</td>
<td>FPM event</td>
</tr>
<tr>
<td>EV_ALSO_CALLED_IN_PBO</td>
<td>Flag if this method is processed also in Process Before Output</td>
</tr>
<tr>
<td>EV_ANYTHING_CHANGED_IN_PBO</td>
<td>Must be set to abap_true if anything was changed in Process Before Output</td>
</tr>
</tbody>
</table>

NAVIGATE:

Called within the FPM event loop, this method forwards the FPM PROCESS_EVENT to the feeder class. This method can be used to cancel the navigation or to change or add an application or business parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_EVENT</td>
<td>The FPM navigation event which is to be processed.</td>
</tr>
<tr>
<td>EV_RESULT</td>
<td>The result of the event processing. There are 3 possible values:</td>
</tr>
<tr>
<td></td>
<td>ev_result = if_fpm_constants=&gt;gc_event_result-OK</td>
</tr>
<tr>
<td></td>
<td>ev_result = if_fpm_constants=&gt;gc_event_result-FAILED</td>
</tr>
<tr>
<td></td>
<td>ev_result = if_fpm_constants=&gt;gc_event_result-DEFER</td>
</tr>
</tbody>
</table>
Tabbed Component (GUIBB TABBED COMPONENT)

A generic design template for organizing additional application-specific views (UIBB) as tabs that is implemented using the Web Dynpro component FPM_TABBED_UIBB.

You use this design template for an application-specific view (UIBB). For example, you could use the template where you want to simultaneously display a selection list of business objects and the additional details of those business objects in tabs without changing the view. You can determine the concrete arrangement of the selection list, detail views, and data when configuring the Web Dynpro component FPM_TABBED_UIBB.

Structure

A tabbed component consists of two areas: the master area and the tab area, which can be arranged next to or on top of one another. If you arrange the areas horizontally, the master area is placed to the left of the tab area. If you arrange the areas vertically, the master area is placed above the tab area.

The content of the master area and the content of the tabs are determined by separate Web Dynpro components, which you set when configuring the Web Dynpro component FPM_TABBED_UIBB.

If you do not set the Web Dynpro component for the master area, this area is not displayed in the application. Instead, only the tabs appear with their application-specific views.

Tabbed Component Editor for Floorplan Manager

You use this editor to adjust a tabbed component within an application to your specific business requirements. This is done by configuring the component.

Features

The editor consists of the following work areas:

- **Preview**

  In the preview, all application-specific views (UIBBs) from the current configuration are displayed so as to give you a picture of the layout of the tabbed component.

- **Layout**

  In this area, you determine whether the tabbed component should be arranged horizontally or vertically. It is also possible to choose whether a tab strip (default) or a drop down list box shall be used for choosing the tabs in the detail area. Furthermore it
is possible to adjust the behavior in that way that if only one tab in the detail area is displayed the tab strip respectively the drop down list shall disappear or stay (default).

- **Hierarchy**

All application-specific views (UIBBs) are displayed in the hierarchy as a tree structure.

- **Attribute view**

Attributes of the currently selected application-specific view (UIBB) that can be changed using the editor are displayed in the attribute view.

- **Action area**

The action area contains links to all the actions you can execute for the tabbed component.

The editor for a tabbed component provides you with the following actions provided as button choice feature:

- Add Master Component (technical name: MASTER UIBB))
- Add Tab (technical name: TAB)
- Add Application-Specific View to Tab (technical name: TAB UIBB)

The editor for a tabbed component is launched in a separate browser window. You can launch the editor in change or display mode and save your changes at any time.

The component-defined processing view is pre-set. Make sure that this view is selected before configuring a tabbed component.

The editor for a tabbed component launches from the configuration editor for Floorplan Manager automatically when you launch the configuration of an application-specific view (UIBB) that has the **FPM_TABBED_UIBB** Web Dynpro component.

**Add Tabbed Component**

You can add an additional, application-specific view to a Floorplan Manager application at any time in the form of a new tabbed component. Depending on the floorplan and the application-specific views already embedded, you can position a new Tabbed Component in one of the following ways:

- in a subview of an object instance
- in a main step of a guided activity
- in a substep of a guided activity
- in a tabbed component in a master UIBB
- in a tabbed component in a tab UIBB
Procedure

Substep A: Calling up the configuration editor of Floorplan Manager

1. Select a Web Dynpro application configuration in the Object Navigator of the ABAP Workbench.
2. In the Web Dynpro Explorer: Display Web Dynpro Configuration screen, choose Web Dynpro Configuration →Test →Execute in Administrator Mode.

   The Web Dynpro application is launched in a separate browser window.

3. In this window, go to the application's identification region and choose the Adapt Configuration link.
4. In the Editor for Web Dynpro ABAP Components — Customizing screen, choose Change.
5. On the Component Customizing <application name> screen, make sure that the Component-Defined view is on.

Substep B: Adding a Tabbed Component

1. In the preview, select the place where you would like to add a new tabbed component.
2. If no UIBB has been defined for a subview or step, choose Attributes.
3. If a UIBB has already been assigned, choose Add UIBB or Add Tabbed UIBB in the action area.

   Now specify the UIBB as a tabbed component by entering the following values in the attribute view:

   1. In the Component field, enter FPM_TABBED_UIBB.

      If you have chosen Add Tabbed UIBB, check whether the field is already correctly prefilled.

   2. In the View field, enter TABBED_WINDOW.

      If you have chosen Add Tabbed UIBB, check whether the field is already correctly prefilled.

   3. In the Configuration Name field, enter a name for the tabbed component configuration.

      Although the Configuration Name field is not marked as a required entry, it is necessary to enter a name here. Otherwise, the Configure UIBB action is not activated.

      The Configuration Type, Configuration Variant, and Sequence Order are optional.
4. Choose Save.

The system has entered the tabbed component as a new UIBB with the name UIBB: TABBED_WINDOW.

Substep C: Configuring the Tabbed Component

1. Choose Configure UIBB.
2. In the Editor for Web Dynpro ABAP Components — Configuration screen, choose Create.
3. In the Create Configuration dialog window, enter a description and choose a package.
4. Choose OK.

Result

The system opens the editor for tabbed components. In the preview, a tab is displayed that the system has generated.

Changing the Tabbed Component Dynamically at Runtime

You may rename, add and remove tabs or child-UIBBs (or embedded UIBBs) from your tabbed component during runtime.

To do so, proceed as follows:

1. Choose an application-specific Web Dynpro component and add the Web Dynpro interface IF_FPM_TABBED_CONF_EXIT to the Implemented Interfaces tab of your Web Dynpro component. This is one of the Web Dynpro components that provide you with a child UIBB.
2. Save and activate the newly added interface. For example, somewhere in your code you want to rename a tab. To do this, you must raise your own FPM event (e.g. CHANGE_TAB_NAME) as the sample code below shows:

```plaintext
DATA: lo_fpm TYPE REF TO if_fpm,
     lo_event TYPE REF TO cl_fpm_event.
lo_fpm = cl_fpm=>get_instance( ).
lo_event = cl_fpm_event=>create_by_id( 'CHANGE_TAB_NAME' ).
lo_event->mo_event_data->set_value( iv_key   = 'ID'
                                lo_event->mo_event_data->set_value( iv_key   = 'NAME'
                                iv_value = lv_tab_name ).
lo_fpm->raise_event( io_event = lo_event ).
```

3. In the component controller, implement the method OVERRIDE_CONFIG_TABBED. To continue with the above example of renaming a tab, implement the following sample code:

```plaintext
```
Drag-and-Drop

Drag-and-Drop is a feature that allows users to move (cut or copy) data at runtime within and between individual user interface building blocks (UIBBs) and generic user interface building blocks (GUIBBs).

Drag-and-Drop allows you to carry out the following activities:

- Move data between a freestyle UIBB, List, and Hierarchical List *
- Move data from a freestyle UIBB, List, or Hierarchical List to a Form
- Move nodes within a Hierarchical List

The Form GUIBB behaves differently from other GUIBBs. The Form GUIBB acts only as a drop target; that is, you can drop data on to a Form GUIBB but it is not possible to drag (move) data from a Form GUIBB.

* The following will be available with the next release of FPM: Movement of data between Hierarchical Lists and movement of data from a freestyle list UIBB to a List GUIBB.

Enabling Drag-and-Drop

Drag-and-Drop can be enabled or disabled for individual UIBBs and GUIBBs separately in an application. It is enabled in the following areas:

- In the feeder class method GET_DEFINITION of the corresponding GUIBB
- In the configuration editor of the corresponding GUIBB
If no Drag-and-Drop attributes are defined in the feeder class, attributes for Drag and Drop do not appear in the configuration editor.

Configuration editor attributes take precedence over feeder class attributes.

The Drag-and-Drop feature has the following attributes:

- **Tags**

  In a Drag-and-Drop operation, the user can only drop data onto a target if both the source and target of the data have at least one common description (a tag). Multiple tags are possible; assign a space between each tag.

- **Scope**

  Scope forms part of the drag source (or drop target) data, indicating whether data can be dragged (or dropped) from one UIBB or GUIBB to another UIBB or GUIBB (Global) or only within the same UIBB or GUIBB (Local).

The drop event is triggered using the FPM event FPM_DROP_COMPLETED. It is also handled in UIBBs by raising the FPM event FPM_DROP_COMPLETED.

Failure in a drop event is indicated by the parameter ET_MESSAGES in the GET_DATA method of the feeder class.

**Configuring Drag-and-Drop**

Drag-and-Drop attributes can also be defined in the configuration editor of the corresponding UIBB or GUIBB.

To configure Drag-and-Drop in the configuration editor, proceed as follows:

1. Ensure that Drag-and-Drop is defined in the corresponding feeder class of the GUIBB.
2. To view the Drag-and-Drop attributes, choose the top node in the Hierarchy in the configuration editor. For a Form GUIBB, the Drag-and-Drop attributes are found by clicking the Group nodes. By default, tags and scope attributes defined in the feeder class are displayed.
3. Select the checkbox Enable Drag or Enable Drop to enable the drag or drop feature.
4. Enter new tags or or add further tags. You can also use wild card options by using *.
5. Enter Local or Global in the dropdown list using the field help.
Events and Event Parameters

During a drop action, the event FPM_DROP_COMPLETED is raised with an event parameter. A structure FPMGB_S_DRAG_AND_DROP is set as the event parameter in the event loop of the event FPM_DROP_COMPLETED.

The structure is comprised of the following parameters:

- **DRAG_SOURCE_DATA**
  A pointer representing the whole data set from where the drag is initiated.

- **DRAG_SOURCE_INDICES**
  The row selected index from where the drag is occurred.

- **DROP_POSITION**
  This is the index where the drop occurs. This is only applicable for LIST and Hierarchical List GUIBBs.

- **DRAG_UIBB_CONFIG_KEY**
  This is the key indicating the drag source UIBB details. The key includes GUIBB component + View + Configuration Name. See example below:

  FPM_LIST_UIBB V_LIST FPM_TEST_DRAG_LIST_UIBB00

Class, Methods and Parameters of Drag-and-Drop

The Drag-and-Drop feature uses the methods, parameters and classes described in the following tables:

<table>
<thead>
<tr>
<th>GET_DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method of the feeder class. Allows the feeder class to provide all necessary information for configuring Drag-and-Drop via the parameter et_dnd_definition.</td>
</tr>
</tbody>
</table>
ET_DND_DEFINITION

This attribute is of type structure FPMGB_S_DND_DEFINITION which defines the drag and drop attributes. The drag and drop has to be defined separately using the attribute TYPE value DRAG or DROP.

CL_FPM_GUIBB_DRAG_AND_DROP

A public class containing methods to raise the drop event and to set the Drag-and-Drop data. It contains the following methods and parameters:

RAISE_DROP_EVENT

FPM GUIBBs use this method internally to raise the FPM_DROP_COMPLETED event during the drop action.

Note that applications can also use this method to raise the FPM_DROP_COMPLETED event during drop on UIBBS. This can be called in on drop action method of the UIBB View.

<table>
<thead>
<tr>
<th>IS_DROP_INFO</th>
<th>This attribute sets the drop information of the GUIBB from ON_DROP_ACTION method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_COMP</td>
<td>This attribute sets the source UIBB configuration details (Config ID + Config type + Config variant).</td>
</tr>
<tr>
<td>IV_COMP_NAME</td>
<td>This attribute sets the source UIBB Component name.</td>
</tr>
</tbody>
</table>

SET_FPM_DROP_DATA

This method is called from the FLUSH method of the UIBB to set the drag and drop data.

<table>
<thead>
<tr>
<th>IO_DATA</th>
<th>This attribute has a reference to the data from where the drag occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_EVENT</td>
<td>This attribute sets the event FPM_DROP_COMPLETED to which the dragged data has to be set.</td>
</tr>
<tr>
<td>IT_INDICES</td>
<td>This attribute sets the selected (dragged) data.</td>
</tr>
<tr>
<td>IV_DROP_POSITION (OPTIONAL)</td>
<td>This attribute sets the drop position in the drop target.</td>
</tr>
<tr>
<td>IV_CONFIG_KEY (OPTIONAL)</td>
<td>This attribute sets the drag source config key to identify the source.</td>
</tr>
</tbody>
</table>

Event Processing during Drag-and-Drop

The FPM_DROP_COMPLETED event is raised by the GUIBBs during a drop action. In the event loop, during a FLUSH call, FPM compares the current (drop) GUIBB configuration name and the drag source configuration name which is set as drag data. If they are the same, FPM will set the drag GUIBB feeder class CT_DATA reference as the FPM event parameter DRAG_SOURCE_DATA which can be used again in the drop feeder class. Similarly, selected lines
will be set as FPM event parameter `DRAG_SOURCE_INDICES` which can be used in the drop feeder class.

In the event loop, during a `GET_DATA` call in the drop feeder class, the application checks the `EVENT_ID` and processes the drop action; it appends or replaces the data. This data is again mapped to the GUIBB via the `CT_DATA` attribute. Similarly, during a `GET_DATA` call in the drag feeder class, the application checks the `EVENT_ID` and removes the data from the drag source or table. The applications map the drop data as required in the `GET_DATA` method.

**Handling Drop in UIBBs**

For UIBBs, the *Drag-and-Drop* attributes *Tags* and *Scope* can be set in the UI elements’ properties tab by creating the drag source information and drop target information. For more details about enabling drag and drop for a UI element, see the SAP NetWeaver library.

However, you can use the FPM event `FPM_DROP_COMPLETED` to handle the drop action in UIBBs. During the `ON_DROP_ACTION` method, raise this event with the required parameters using the method `RAISE_DROP_EVENT`. You can also set the *Drag-and-Drop* information using the `SET_FPM_DROP_DATA` method.

**Navigation with Launchpads**

To navigate to a specific application outside of your FPM application, you use the following FPM toolbar menus:

- *You Can Also*
- *Related Links*

These FPM toolbar menus utilize launchpads.

**Navigation APIs**

The FPM also provides you with the following two navigation interfaces, allowing you to control the launchpads:

- **IF_FPM_NAVIGATION**: Use this to navigate to an application using a given launchpad.
- **IF_FPM_NAVIGATE_TO**: Use this to navigate to an application without using a launchpad.

**Suspend and Resume**

The *Suspend and Resume* feature is available for FPM applications. This can be described briefly as a feature in which the Web Dynpro application, built within the FPM framework, can be placed in a suspended state whilst the user navigates to another URL. The user can
work on the URL and then navigate back to the suspended FPM application, which is resumed from exactly the same state before navigation occurred.

The *Suspend and Resume* feature is also available with Web Dynpro ABAP and Web Dynpro Java application categories.

Note that the usage of a report launchpad is mandatory to enable suspend and resume for FPM applications.

For a detailed explanation of this feature, see [Suspend and Resume](#).

**Launchpads**

A collection of navigation destinations that is stored as a separate technical object in the system.

You use a launchpad to allow users to navigate to specific goals outside of the current Floorplan Manager application. For example, this could mean navigating to other Web Dynpro ABAP applications, external Web pages, transactions, reports, or other business objects. Within a floorplan, two elements, *YouCanAlso* and *RelatedLinks*, are available to you in the toolbar. You can assign different launchpads to these elements.

The launchpad is displayed as a dropdown list. Descriptions that you have written for launchpad applications cannot be displayed in Floorplan Manager.

*YouCanAlso* and *RelatedLinks* are default toolbar elements of the SAP user interface design for floorplans. They are displayed as links. In the concrete Floorplan Manager application, both elements can have different names on the user interface. This name can also be changed using the configuration editor.

Any number of applications can be assigned to a launchpad. Each application is described by its application type as well as additional attributes that are dependent on the application type. In a Floorplan Manager application, not every application type can be called up in every runtime environment. The following table describes which application types can be called up in which runtime environment.

### Application Types in the Runtime Environment

<table>
<thead>
<tr>
<th>Application Type</th>
<th>SAP Enterprise Portal</th>
<th>SAP NetWeaver Business Client (Portal Roles)</th>
<th>SAP NetWeaver Business Client 1.0 (PFCG Roles)</th>
<th>SAP NetWeaver Business Client 3.0 (PFCG Roles)</th>
<th>Stand-Alone Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portal Pages</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>Transactions</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>BAdI implementation is necessary</td>
</tr>
</tbody>
</table>

(C) SAP AG  Floorplan Manager ABAP - Developer's Guide Preliminary Version
<table>
<thead>
<tr>
<th>Feature</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>BAdI implementation is necessary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Writer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>SAP BI Report (Query)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>SAP BI Report (Web Template)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>BI Report</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>BEx Analyzer</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>Manager's Desktop</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Web Dynpro Java</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>Web Dynpro ABAP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>KM Document</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>Visual Composer xApps</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>Info Set Query</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>Object-Based Navigation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>Crystal Reports</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td></td>
<td>BAdI implementation is necessary</td>
</tr>
<tr>
<td>Xcelsius Dashboards</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td></td>
<td>BAdI implementation is necessary</td>
</tr>
</tbody>
</table>

Multiple folders can be created for every launchpad. You can use these folders to group applications.
As opposed to the conventional use of launchpads in portals, in Floorplan Manager only those navigation destinations that were created as applications in the first visible folder after the top node of the launchpad are shown. The applications in the other folders of the launchpad are hidden in the display.

Creating a Launchpad with Applications

Procedure

To create a launchpad, complete the following steps:

1. On the SAP Easy Access screen, enter /nlpd_cust in the command field.
2. Choose Continue.

   The system calls up the start screen for Launchpad Customizing.

3. On the Overview of Launchpads screen, choose New Launchpad. In the dialog box, enter the following information:
   - role name
   - instance name
   - a description
   - the namespace (if required)

4. Launchpad Type is empty by default; for the Launchpad GUIBB you need the launchpad type FPM_UIBB. When you use an embedded launchpad, for example You Can Also, in the Launchpad GUIBB you must use the launchpad type Embedded. Version 1 is the default version.
5. If the launchpad should be switched, enter a switch id.
6. Choose Continue.
8. Enter the data for LinkText.

You can write a description for the application. However, Floorplan Manager cannot display this description in the dropdown list.

9. Choose an application type.
10. Enter the information for the selected application type.
11. Choose Save.

To add multiple applications, repeat steps 7 through 10.
For launchpads to which you would like to assign an application, we recommend that you do not use a folder. This is because Floorplan Manager currently displays only one folder. If the launchpad consists of multiple folders, only the topmost folder is displayed.

**Including a Launchpad in the User Interface**

**Procedure**

To assign a launchpad to the YouCanAlso or RelatedLink elements on the user interface, complete the following steps.

1. Select a Web Dynpro application and application component.
2. Select a Web Dynpro application configuration in the Object Navigator of the ABAP Workbench.

The Web Dynpro application is launched in a separate browser window.

4. In this window, go to the application's identification region and choose the Adapt Configuration link.
5. On the Editor for Web Dynpro ABAP Components — Customizing screen, choose Change.
6. On the Component Customizing <application name> screen, make sure that the Component-Defined view is on.
7. In the navigation area of the configuration editor, choose the Toolbar element.
8. Choose Expand Node.
9. To display the attributes of the YouCanAlso element, click the YouCanAlso pushbutton.

If you would like to assign the launchpad to the RelatedLinks element, choose the RelatedLinks element.

10. In the Role field, enter the name of the launchpad role.
11. In the Instance field, enter the name of the launchpad instance.
12. To change the name of the button element, enter a different name in the Name field.
13. Save the configuration.
14. Test the new configuration.

**Working in the Navigation Customizing**

**General Settings**

You can make changes to a launchpad that are valid for all destinations and applications in a launchpad. To do this, proceed as follows:

1. Choose transaction LPD_CUST.
2. Locate and open your launchpad.
3. On the menu bar, choose Extras →General Settings.
4. In the Change Launchpad Role dialog box you can set the following options:
   - **OBN Navigation Mode**
     - **User Set of Roles**: An Object-based navigation (OBN) target can be assigned to any roles that are assigned to a user.
     - **Source Role**: The navigation target must be assigned to the same role as the application that uses the launchpad.
   - **Check Application Alias is Unique**
     Use this to check that a destination application alias being used by an FPM application is unique
   - **Set Icons as Selectable**
     - **Marked**: Folder icons in the UIBB Launchpad are selectable for navigation.
     - **Unmarked**: Folder icons in the UIBB Launchpad are not selectable for navigation.

**Source Parameters and Parameter Mapping**

To support parameter mapping, you can define a set of parameters that are known by the FPM application that uses the launchpad. To do this, proceed as follows:

To do this, proceed as follows:

1. Choose transaction **LPD_CUST**.
2. Locate and open your launchpad.
3. On the menu bar, choose **Extras → Source Parameters**. The Default Parameters dialog box displays the parameters which are known by your FPM application.
4. Some application categories (for example Web Dynpro ABAP) have a button **Parameter Mapping** in the **Parameter** input field. To map your parameters, choose this button to extend the Default Parameter dialog box and enter the following data:
   - **Parameter**: Enter the parameter name that the FPM application sends to the destination application.
   - **Replaced by**: Enter the parameter name that the destination application expects to receive. When you launch the destination application, the launchpad automatically replaces the parameter that was sent by the FPM application by the parameter in the Replaced by column.
5. For application categories such as BI Query, Template, Enterprise Report and Xcelsius Dashboards, you can also enter the following data:
   - **Value Type**
     Defines the type of the parameter. Possible values are:
     - **Single Value**
     - **Multiple Values**
     - **Interval**
     - **Multiple Intervals**
     - **Groups**
     - **External Format**
Defines the kind of parameter. Possible values are:

- **Variable**
- **Filter**
- **Dynamic Placeholder**

A dynamic placeholder is a variable in the form `<TEXT>` that you enter as a placeholder in the field `Application Parameter`. This variable is replaced at runtime either by the source parameter or the fixed value specified in the `Fixed Value` parameter.

- **Fixed Value**

This value is forwarded to the target application if the `Field Source` parameter is empty or if the flag `Overwrite Source Parameter` is set.

- **Overwrite Source Parameter**

If you set this flag, the fixed value is forwarded to the target application, independent of whether the field `Source Parameter` contains a value or not.

6. For *Crystal Report* application categories, you can enter the following data:

- **Value Type**

  Defines the type of the parameter. Possible values are:

  - **Single Value**
  - **Multiple Values**
  - **Interval**

- **Type**

  Defines the kind of parameter. Possible values are:

  - **Empty**
  - **Dynamic Placeholder**

  A dynamic placeholder is a variable in the form `<TEXT>` that you enter as a placeholder in the `Application Parameter` field. This variable is replaced at runtime either by the source parameter or the fixed value specified in the `Fixed Value` parameter.

- **Fixed Value**

  This value is forwarded to the target application if the `Field Source` parameter is empty or if the flag `Overwrite Source Parameter` is set.

- **Overwrite Source Parameter**
If you set this flag, the fixed value is forwarded to the target application, independent of whether the field Source Parameter contains a value or not.

**Copying an entire Launchpad**

To copy an entire launchpad, proceed as follows:

1. Choose transaction LPD_CUST.
2. On the menu bar of the Overview of Launchpads screen, choose ➔ Launchpad ➔Read from other system by RFC ➔.
3. In the Action Launchpad dialog box, enter an RFC destination. This displays a list of launchpads available in the system and client you have entered as the RFC destination. Note that if you leave RFC-Destination empty, you are provided with a list of all launchpads in your current client.
4. Choose Continue.
5. Choose the launchpad that you want to copy and choose Continue.
6. In the dialog box that appears, enter another role and/or instance. Choose Continue.

**Copying Applications from one Launchpad to Another**

For convenience, you can copy one or more applications inside a launchpad to another launchpad. To do this, proceed as follows:

1. Choose transaction LPD_CUST.
2. Choose New Launchpad.
3. In the dialog box, enter the Role, Instance and Description. Choose Continue.
4. Choose Copy from other Launchpad. A dialog box appears, listing all the launchpads in the current system and client.
5. Choose a Launchpad and choose Continue.
6. In the navigation area, your chosen launchpad appears next to your new launchpad. Click and drag the launchpad destination applications from your chosen launchpad to your new launchpad. Note that you can also drag an entire folder to a new launchpad.
7. Save your entry.

**Performing Searches in Launchpads of a Client**

1. You can perform searches covering all launchpads in a client. To do this, proceed as follows:
2. To initialize a search, run the report APB_LPDP_UPDATE_SEARCH_TABLE. This report analyzes all launchpads in a system and client, and displays the information in a search table.
3. In the Launchpad Customizing, Change Launchpad Role screen, choose Search.
4. You are provided with a list containing all applications of all launchpads in the current client. You can now perform searches throughout the whole list and select the appropriate applications or launchpads.
5. Choose OK to display a selected application in the navigation area.
Re-displaying a SAP-Delivered Launchpad

You can make changes to a particular launchpad and save the changes. However, it is still possible to display the SAP-delivered launchpad. To do this, proceed as follows:

1. Choose transaction `LPD_CUST`.
2. Open the launchpad which you have made changes to.
3. Choose `Extras → Show SAP Version`. In the navigation area, the SAP-delivered launchpad (without the changes) appears alongside the changed version of the launchpad.

Replacement of Switch BC-Set

Previously, a launchpad had to be added to Switch BC-Sets to ensure its visibility in business functions. This has been replaced by the `Direct Switched Launchpad`.

Switching on a Whole Launchpad

To switch on a whole Launchpad, enter the switch ID when you create a new Launchpad. You can also enter a switch ID by opening an existing launchpad and selecting the menu option `Launchpad → Add Switch ID`.

Switching Applications On and Off inside a Launchpad

To switch an application on or off inside a Launchpad, enter a value in the field `Switch Id`. In a system with customer status, this application will be shown only if the switch is active.

Replacing Existing Applications inside a Launchpad

It is possible to replace an existing application inside a launchpad. For example, you can replace a transaction that was delivered in a previous release with a new WD application in the current release. To do this, enter a replacement application in the `Replace Application` field in the `Switch Support` section under `Advanced Parameters`. In a customer system, the former application is replaced by the current application only if the switch is active.

Mapping of Business Objects and Operations

To support object-based navigation in the Portal and the SAP NetWeaver Business Client, you can provide an alternative back-end business object and operation name. These names are used in the SAP NetWeaver Business Client connected to a PFCG role. To do this, enter `Object Based Navigation` in the `Application Category` text box. The `Application Parameters` section appears, allowing you to enter the business object and operation name.

Replacing Object-Based Navigation

In a ‘browser only’ environment, object-based navigation does not work. To support this environment, you can enter an alternative target such as WD ABAP in the OBN application category. To do this, enter an existing application in the `Alternative Application for Browser`
Only Environment field. This target will be used automatically in the ‘browser only’ environment for navigation.

**Transporting a Launchpad**

To transport a launchpad, proceed as follows:

1. Choose transaction LPD_CUST.
2. Open the launchpad you want to transport in change mode.
3. Choose Launchpad → Transport.
4. In the dialog box, enter the package to which you want to assign the texts that you created in the launchpad. As a result, the texts are also forwarded to translation. Choose Continue.
5. In the dialog box, enter a Customizing request and choose Continue. This request includes the relevant table entries for the following tables:
   - APB_LAUNCHPADT
   - APB_LAUNCHPAD_V
   - APB_LPDP_CONTROLL
   - APB_LPDP_OTR_KEYS
   - APB_LPDP_VERSIONS
6. In the dialog box, enter a Workbench request. This request includes the texts from the launchpad. These are objects of the type R3TR DOCT.
7. Release both requests.
8. For further information, see SAP Note 1165160.

**IF_FPM_NAVIGATION API**

(Runtime class CL_FPM_NAVIGATION)

This navigation interface provides you with a list, MT_TARGETS, with all customized applications of a given launchpad.

To access this navigation API, use the interface IF_FPM. This provides the GET_NAVIGATION method, which returns an instance of the navigation API, IF_FPM_NAVIGATION.

**Tables, Attributes and Domains**

<table>
<thead>
<tr>
<th>Table: MT_TARGETS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td>entry_type</td>
</tr>
<tr>
<td>parent</td>
</tr>
<tr>
<td>key</td>
</tr>
<tr>
<td>alias</td>
</tr>
<tr>
<td>text</td>
</tr>
</tbody>
</table>

(C) SAP AG Floorplan Manager ABAP - Developer's Guide Preliminary Version
Table: MT_TARGETS

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Kind</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Type</td>
<td>STRING</td>
<td>Description.</td>
</tr>
<tr>
<td>icon_path</td>
<td>Type</td>
<td>STRING</td>
<td>Path to an icon.</td>
</tr>
<tr>
<td>enable</td>
<td>Type</td>
<td>BOOLE_D</td>
<td>Determines if an application is active/enabled or inactive/disabled.</td>
</tr>
<tr>
<td>visible</td>
<td>Type</td>
<td>BOOLE_D</td>
<td>Determines the visibility of an application</td>
</tr>
</tbody>
</table>

Attributes:

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OPENING_LAUNCHPAD_FAILED</td>
<td>True indicates that the launchpad could not be opened successfully</td>
</tr>
</tbody>
</table>

Domain: FPM_NAVIGATION_TARGET_TYPE

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP</td>
<td>Line contains an application.</td>
</tr>
<tr>
<td>FOL</td>
<td>Line contains a folder.</td>
</tr>
<tr>
<td>SEP</td>
<td>Line contains a separator.</td>
</tr>
</tbody>
</table>

Methods

This navigation interface provides the methods described in the tables below:

**NAVIGATE:**

Starts the navigation of an application.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Direction</th>
<th>Type kind</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV_TARGET_KEY</td>
<td>importing</td>
<td>Type</td>
<td>STRING</td>
<td>GUID of Application.</td>
</tr>
</tbody>
</table>

**MODIFY:**

Changes attributes of an application. For example, you can change the visibility of an application, enable or disable an application and change its description and text.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Direction</th>
<th>Type kind</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV_VISIBLE</td>
<td>importing</td>
<td>Type</td>
<td>BOOLE_D</td>
<td>Set an application to visible/invisible.</td>
</tr>
<tr>
<td>IV_ENABLE</td>
<td>importing</td>
<td>Type</td>
<td>BOOLE_D</td>
<td>Enable/disable an application.</td>
</tr>
<tr>
<td>IV_TEXT</td>
<td>importing</td>
<td>Type</td>
<td>STRING</td>
<td>An alternative text for the application.</td>
</tr>
<tr>
<td>IV_DESCRIPTION</td>
<td>importing</td>
<td>Type</td>
<td>STRING</td>
<td>An alternative description for the application.</td>
</tr>
<tr>
<td>IV_TARGET_KEY</td>
<td>importing</td>
<td>Type</td>
<td>STRING</td>
<td>GUID of Application.</td>
</tr>
<tr>
<td>IV_NOTIFY</td>
<td>importing</td>
<td>Type</td>
<td>BOOLE_D</td>
<td>Invokes notification on all registered nodes / objects.</td>
</tr>
</tbody>
</table>

**SET_FILTER:**

Allows you to display the content of another folder.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Direction</th>
<th>Type kind</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT_Filter</td>
<td>importing</td>
<td>Type</td>
<td>T_FILTER</td>
<td>GUIDs of folder which content should be displayed.</td>
</tr>
</tbody>
</table>

**MODIFY_PARAMETERS:**

Changes the values of existing parameters or adds a parameter if none exists.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Direction</th>
<th>Type kind</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_TARGET_KEY</td>
<td>importing</td>
<td>Type</td>
<td>STRING</td>
<td>GUID of Application.</td>
</tr>
<tr>
<td>IT_APPLICATION_PARAMETER</td>
<td>importing</td>
<td>Type</td>
<td>APB_LP D_T_PA RAMS</td>
<td>Contains application parameters that will be added or changed.</td>
</tr>
<tr>
<td>IT_BUSINESS_PARAMETER</td>
<td>importing</td>
<td>Type</td>
<td>APB_LP D_T_PA RAMS</td>
<td>Contains business parameters that will be added or changed.</td>
</tr>
</tbody>
</table>

**ADD_BEX_ANALYZER:**

Adds an application of type BEx Analyzer to a given launchpad.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Direction</th>
<th>Type kind</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV_PARENT_FOLDER_ID</td>
<td>importing</td>
<td>Type</td>
<td>FPM_APPLICATION_ID</td>
<td>GUID of parent folder. If the parameter is empty, the application will be added at top level.</td>
</tr>
<tr>
<td>IS_BEX_ANALYZER_FIELDS</td>
<td>importing</td>
<td>Type</td>
<td>FPM_S_BEX_ANALYZER</td>
<td>Structure that contains the fields to add with BEx Analyzer application type.</td>
</tr>
<tr>
<td>EV_APPLICATION_ID</td>
<td>exporting</td>
<td>Type</td>
<td>FPM_APPLICATION_ID</td>
<td>GUID of Application.</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>exporting</td>
<td>Type</td>
<td>FPM_T_T100_MESSAGES</td>
<td>Error messages.</td>
</tr>
<tr>
<td>EV_ERROR</td>
<td>exporting</td>
<td>Type</td>
<td>BOOLE_D</td>
<td>Status = false - the application was added; Status = true - an error occurred.</td>
</tr>
</tbody>
</table>

The following are other methods with a similar interface to **ADD_BEX_ANALYZER**, which allow you to add a specified application, at runtime, to a launchpad:

- ADD_URL
- ADD_TRANSACTION
- ADD_REPORT_WRITER
- ADD_OBN
- ADD_INFOSET_QUERY
- ADD_FOLDER
- ADD_BI.ENTERPRISE_REPORT
- ADD_BI_QUERY
- ADD_BI_TEMPLATE
- ADD_KM_DOCUMENT
- ADD_PORTAL_PAGE
- ADD_VISUAL_COMPOSER
- ADD_WEBDYNPRO_ABAP
- ADD_WEBDYNPRO_JAVA
- ADDCRYSTAL_REPORT
- ADD_XCELSIUS_DASHBOARD

**REMOVE:**

Removes an application from a launchpad.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Direction</th>
<th>Type kind</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_APPLICATION_ID</td>
<td>importing</td>
<td>Type</td>
<td>STRING</td>
<td>GUID of Application</td>
</tr>
</tbody>
</table>

**Integration: Navigation in the Event Loop**

If you call the `IF_FPM_NAVIGATION` method `NAVIGATE`, a new event object of type `cl_fpm_navigation_event` is created. This event object contains all the application parameters. The interface `IF_FPM_UI_BUILDING_BLOCK` contains the `PROCESS_EVENT` method, which allows you to call the navigation event and change these parameters.

To do this, implement the following code in the `PROCESS_EVENT` method:

```
"First check if the event is a navigation event"
check io_event->MV_EVENT_ID = io_event->gc_event_navigate.
"Make a cast from the event object to the cl_fpm_navigation_event object"
DATA lr_event type ref to cl_fpm_navigation_event.
lr_event ?= io_event.
"Get the business parameter"
lr_bus_parameter ?= lr_event->mo_event_data.
"Get the launcher parameter"
lr_launcher_parameter ?= lr_event->mo_launcher_data.
```

Note the use of the following `lr_parameter` methods:

- `to_lpparam`: provides you with an internal table with the parameters
- `get_value`, `set_value` or `delete_value`: allow you to change a parameter

If the event processing requires further user interaction (for example, requesting further data via a dialog box), the event processing can be deferred by returning `EV_RETURN = IF_FPM_CONSTANTS~GC_EVENT_RESULT-DEFER`. 
If the result of the event processing is ok, you can return \texttt{EV\_RETURN = IF\_FPM\_CONSTANTS\_GC\_EVENT\_RESULT\_OK}; if the result of the event processing is not ok, you can return \texttt{EV\_RETURN = IF\_FPM\_CONSTANTS\_GC\_EVENT\_RESULT\_FAILED}.

To prevent a loss of data, you can implement the \texttt{NEEDS\_CONFIRMATION} method. This method is located in the interface \texttt{IF\_FPM\_UI\_BUILDING\_BLOCK}. This method contains the navigation event and you can decide whether to raise a data-loss dialog box. To do this, you must return the following value: \texttt{eo\_confirmation\_request = cl\_fpm\_confirmation\_request=>go\_data\_loss}.

**\texttt{IF\_FPM\_NAVIGATE\_TO} API**

This interface provides you with a set of methods to launch an application without using a launchpad.

To access this Navigation API, use the interface \texttt{IF\_FPM}. This provides the method \texttt{GET\_NAVIGATE\_TO( )} which returns an instance of the Navigation API \texttt{IF\_FPM\_NAVIGATE\_TO}.

This interface contains the methods described in the table below and the following list.

**Methods of \texttt{IF\_FPM\_NAVIGATE\_TO} API**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{LAUNCH_BEX_ANALYZER}</td>
<td>Launches an application of type BEx Analyzer.</td>
</tr>
<tr>
<td>\texttt{IS_BEX_ANALYZER_FIELDS}</td>
<td>Structure that contains the fields to add with BEx Analyzer application type.</td>
</tr>
<tr>
<td>\texttt{ET_MESSAGES}</td>
<td>Error messages</td>
</tr>
<tr>
<td>\texttt{EV_ERROR}</td>
<td>Status: false - the application was added; true - an error occurred</td>
</tr>
</tbody>
</table>

The following are other methods with a similar interface to \texttt{LAUNCH\_BEX\_ANALYZER}, which allow you to launch a specified application:

- \texttt{LAUNCH\_URL}
- \texttt{LAUNCH\_TRANSACTION}
- \texttt{LAUNCH\_REPORT\_WRITER}
- \texttt{LAUNCH\_OBN}
- \texttt{LAUNCH\_INFOSET\_QUERY}
- \texttt{LAUNCH\_FOLDER}
- \texttt{LAUNCH\_BI\_ENTERPRISE\_REPORT}
- \texttt{LAUNCH\_BI\_QUERY}
- \texttt{LAUNCH\_BI\_TEMPLATE}
Restarting a WD ABAP Application

The IF_FPM_NAVIGATE_TO interface contains a RESTART method. This method will restart the currently running WD application.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_HEADER_TEXT</td>
<td>Text that will be displayed as title in the window.</td>
</tr>
<tr>
<td>ID_WD_CONFIGURATION</td>
<td>An optional WD ABAP application configuration</td>
</tr>
<tr>
<td>IT_PARAMETERS</td>
<td>An optional set of parameters that will be forwarded to the application</td>
</tr>
<tr>
<td>ET_MESSAGES</td>
<td>Messages</td>
</tr>
<tr>
<td>EV_ERROR</td>
<td>Is set to true if an error occurred</td>
</tr>
</tbody>
</table>

Note that there are some restrictions for the restart feature:

- The iView that is used to start the Web Dynpro application must contain the application parameter system_alias=<system>.
- The iView exists on a page on its own which means only this application is executed in the main window. If several applications are displayed in the main window, the application that is used to execute the restart function is the only one that is displayed after a restart.
- If the restart occurs from an external window, you must navigate to this window with navigation mode 3.
- The window header can be transferred to the restart method. If no header is transferred, the system displays the text Launchpad Start WD ABAP.
- In NetWeaver Business Client, a navigation bar cannot be displayed on the left-hand side.
- If navigation takes place using object-based navigation (OBN) in the Portal or in the NetWeaver Business Client that is connected to a Portal, the OBN Navigation Mode must be set to User Set of Roles. If it is set to Source Role, the system cannot find the targets following a restart because the restart changes the role context.

Extracting Launchpad Content and Launch Service

To extract the content of one or more launchpads, you can use the function module READ_LAUNCHPADS. This function module contains the following import parameters:

- ID_ROLE
- ID_INSTANCE
- ID_LANGU
All three parameters are optional. If you call the function module without any parameters you will get all launchpads that exist in the client in all existing languages.

To launch a single entry of the extracted launchpad, call the Web Dynpro application APB_LPD_LAUNCH_SERVICE. This application expects the following URL parameters:

- role
- instance
- application_id

It is also possible to add business parameters to the URL. These parameters are forwarded to the target application.

**Suspend and Resume**

The Suspend and Resume feature enables an FPM application to remain in a suspended state when a user navigates to a URL or any other Web Dynpro ABAP or Web Dynpro Java application. When the user navigates back to the FPM application, the Suspend and Resume feature allows the application to be resumed in the exact state it was before navigation occurred.

The basic settings to utilize this feature include the time out of suspended applications. Session Management and the Suspend and Resume feature are provided by technology layers like Web Dynpro ABAP Foundation, Portal, and ABAP Server etc and are not provided or influenced by FPM. Suspend and Resume is supported in the following client environments:

- Stand-alone
- NWBC
- Portal

Suspend and Resume is currently limited to URL and Web Dynpro ABAP or Web Dynpro Java application navigation. In the Report Launchpad Customizing, Suspend and Resume is only available for the URL and Web Dynpro ABAP or Web Dynpro Java application category of Report Launchpads. The same is also applicable to the API, in that only dynamic navigation to URLs via APIs can utilize the Suspend and Resume methods.

There is a uniform method to enable both Suspend and Resume across all the clients. But the method in which the external URLs get the information to navigate back to the Web Dynpro application varies. Only the FPM’s methods to suspend and resume are detailed here.

With the Suspend and Resume feature, it is possible to pass parameters back and forth to the URL from the FPM application.
Suspending via Static Launchpad Customizing for URL Application Category

1. Open the Launchpad Customizing (transaction LPD_CUST).
2. On the Overview of Launchpads screen, choose New Launchpad.
3. Enter the Role, Instance and Description. Choose Continue.
5. Enter the following details:
   - Linktext – for example FPM_TEST
   - Application Category – choose URL
   - Application Parameters - enter the URL of the application to be opened on suspension of the FPM application.

   Note that you can also enter a description and application alias. The application alias is recommended if you use APIs of the launchpad.

6. Check the Activate Suspend and Resume Functionality checkbox.

When the user uses this launchpad application to navigate away from the FPM application, the FPM application is suspended.

Suspending via Static Launchpad Customizing for Web Dynpro ABAP or Web Dynpro Java Application

1. Customizing in LPD_CUST is similar as above but has to select the WDA or WDJ in Application category field.
2. Check the Activate Suspend and Resume Functionality checkbox after providing all other information.

When the user uses this launchpad application to navigate away from the FPM application, the FPM application is suspended.

Suspending via Launchpad API

It is possible from EhP1 of NW onwards to also use navigation dynamically, that is without creating a launchpad Customizing. It is possible to enable Suspend and Resume for such navigation too.

For information on how to get a handle to IF_FPM_NAVIGATE_TO, see Navigation.

Once a handle is obtained to the IF_FPM_NAVIGATE_TO object, you can call the method LAUNCH_URL to open external applications. This method takes in an input parameter IS_URL_FIELDS of type FPM_S_LAUNCH_URL. In the structure FPM_S_LAUNCH_URL, the field USE_SUSPEND_RESUME must be set to abap_true or ‘X’. When the application is launched (refer to Dynamic APIs of the launchpad), the FPM application is suspended.
Resuming a Suspended Application

When the user wants to navigate from the external URL back to the suspended FPM application, the FPM event loop is triggered. This is the entry point back into the application.

The application reacts to the FPM event `FPM_RESUME`, which is accessed via the constant `CL_FPM_EVENT=> GC_EVENT_RESUME`. The event data will contain the URL parameters that are passed from the external URL back into the FPM application.

The key to access this is via the following key parameter:
`CL_FPM_SUSPEND_RESUME_UTILITY=>CO_RESUME_URL_PARAMETERS`. The value obtained is an internal table of the type `TIHTTPNVP`, containing the URL key-values pair passed by the external application. Note that this data is available only during the lifetime of the event object and is not stored by FPM. The application maintains a copy if the user needs to access this information later.

Sample code to resume an application is shown below (in the Component Controller's `PROCESS_EVENT` method):

```
METHOD PROCESS_EVENT .
  "We will need to check the Navigation mode and set it to the launch pad accordingly.
  DATA lr_event TYPE REF TO cl_fpm_navigation_event.
  "Check if this is the resume event.
  CASE io_event->mv_event_id.
    WHEN cl_fpm_event=>gc_eventResume.
      get_resume_parameters( io_event ).
  ENDCASE.
  Method GET_RESUME_PARAMETERS
  DATA: lr_fpm_event_data TYPE REF TO if_fpm_parameter.
  DATA: it_url_parameters TYPE tihttpnvp.
  lr_fpm_event_data = io_event->mo_event_data.
  CALL METHOD lr_fpm_event_data->get_value EXPORTING
    iv_key = cl_fpm_suspend_resume_utility=>co_resume_url_parameters
    IMPORTING
    ev_value = it_url_parameters.
```

At the end of this code, the internal table `it_url_parameters` contains the URL parameters passed back from the external application. The above mentioned code, along with other information, can be found in the test application `FPM_TEST_SUSPEND_RESUME` in the `APB_FPM_TEST` package.

Handling Dialog Boxes

Depending on the action required, you can manage dialog boxes in the following ways:
Using the `NEEDS_CONFIRMATION` method during the FPM Event Loop

Using the `PROCESS_EVENT` method for the handling of application-specific dialog boxes

Using the work-protect mode offered by the Portal and the NWBC (using the `IF_FPM_WORK_PROTECTION` interface)

**Triggering a Data-Loss Dialog Box in the FPM Event Loop**

Each UIBB can request a data-loss dialog box during the FPM event loop.

To do this, return the pre-defined instance of the class `CL_FPM_CONFIRMATION_REQUEST` as detailed below:

```plaintext
METHOD needs_confirmation
  IF ...
    eo_confirmation_request = cl_fpm_confirmation_request=>go_data_loss
  ENDIF
ENDMETHOD
```

To display other confirmation dialog boxes, create your own instance of the class `CL_FPM_CONFIRMATION_REQUEST` and add your own application-specific text.

**Handling Application-Specific Dialog Boxes**

To process an event in method `IF_FPM_UI_BUILDING_BLOCK~PROCESS_EVENT` (see chapter FPM Events), it may be necessary to gather additional information from the user by means of a dialog box. Dialog boxes may contain simple text and buttons, but they may also be more complex and include input fields, checkboxes, etc.

The processing of dialog boxes in Web Dynpro programming can be cumbersome, since Web Dynpro dialog boxes cannot be processed in a synchronous way (that is trigger the dialog box, wait for it to be closed and continue processing). This means that the UIBB would need to return the result of the event processing (OK or FAILED) before the dialog box could be processed.

To achieve synchronous dialog box handling, the FPM allows you to defer the processing of the event loop and resume it after the dialog box has been processed. This procedure is described below:

**Procedure**

**Deferring Current Event Processing**

You defer the processing of the current event in the method `PROCESS_EVENT`. Sample code for this is shown below:
ev_result = if_fpm_constants=>gc_event_result-defer.

Registering a Dialog Box

This procedure is purely Web Dynpro ABAP and not a feature of the FPM. Therefore, we recommend that you read the Web Dynpro ABAP documentation regarding Web Dynpro ABAP dialog boxes in general. Nevertheless, a short description of how to register a dialog box is detailed below:

Firstly, the registration of the dialog box with Web Dynpro needs to be triggered in the method `PROCESS_EVENT`, as this is the last method until program control returns to the FPM.

However, for an application-specific dialog box you need your own Web Dynpro ABAP View and the registration of the dialog box is only possible from within this View. For this reason, in the method `PROCESS_EVENT` you need to call a method of the View that is used for the application-specific dialog box. However, as View methods cannot be accessed from within methods of the component controller, you need to use the Web Dynpro ABAP event mechanism: raise an event in the method `PROCESS_EVENT` and register an event handler on the corresponding View.

The process for this is described below:

1. Create a new Web Dynpro ABAP View and name it `DIALOG BOX_CARRIER`.
2. In the `Component Controller`, create a new Web Dynpro ABAP Event and name it `REGISTER_DIALOG BOX_EVENT`.
3. In the method `PROCESS_EVENT` raise the Web Dynpro ABAP Event `REGISTER_DIALOG BOX_EVENT`.
4. In the View `DIALOG BOX_CARRIER`, create a new method and name it `REGISTER_DIALOG BOX` of method type event handler for the event `REGISTER_DIALOG BOX_EVENT`.
5. In the method `REGISTER_DIALOG BOX`, use the ABAP Window API to create a dialog box, register action handler methods to the buttons of the dialog box and register the dialog box for opening.
6. Create Web Dynpro ABAP actions and handler methods for the actions that arise from the dialog box; in this case, from the `Yes` and `No` buttons. In the example above, the names are `ONRESUME_EVT_OK` and `ONRESUME_EVT_FAILED`.

The sample code below shows how this might look (the code uses a standard dialog box with buttons `Yes` and `No`):

```abh
DATA: lo_api              TYPE REF TO if_Web Dynpro_component,
     lo_window_manager   TYPE REF TO if_Web Dynpro_window_manager,
     lo_view_api         TYPE REF TO if_Web Dynpro_view_controller,
     lo_dialog box       TYPE REF TO if_Web Dynpro_window,
     lo_api            = Web Dynpro_comp_controller->Web Dynpro_get_api( ).
```

```abh
lo_api->Web Dynpro_view_controller->Web Dynpro_window_manager->Web Dynpro_window_manager->Web Dynpro_get_api( ).
```
lo_window_manager = lo_api->get_window_manager( ).
lo_view_api = Web Dynpro_this->Web Dynpro_get_api( ).
lo_dialog_box = lo_window_manager->create_dialog_box_to_confirm(
  text = 'some dialog box text...'
  button_kind = if_Web Dynpro_window=>co_buttons_yesno
  message_type = if_Web Dynpro_window=>co_msg_type_question
  window_title = 'some dialog box title...'
  window_position = if_Web Dynpro_window=>co_center).
CALL METHOD lo_dialog_box->subscribe_to_button_event
  EXPORTING
    button = if_Web Dynpro_window=>co_button_yes
    action_name = 'ONRESUME_EVT_OK'
    action_view = lo_view_api.
1. CALL METHOD lo_dialog_box->subscribe_to_button_event
  EXPORTING
    button = if_Web Dynpro_window=>co_button_no
    action_name = 'ONRESUME_EVT_FAILED'
    action_view = lo_view_api.
lo_dialog_box->open( ).

Resuming the Event

Once the required user input has been obtained, the frozen FPM event is continued (either receiving the result OK or FAILED). To do this, call the FPM method RESUME_EVENT_PROCESSING within the action handler methods for the buttons of the dialog box. The sample code below shows how this might look:

\[
\begin{align*}
\text{DATA lo_fpm TYPE REF TO if_fpm.} \\
\text{lo_fpm = cl_fpm_factory->get_instance( ).} \\
\text{lo_fpm->resume_event_processing( if_fpm_constants=>gc_event_result-ok ).}
\end{align*}
\]

After the event is resumed, the remaining UIBBs are processed (if there is more than one UIBB).

The figure below summarizes the behavior described above:
The FPM allows the application to make use of the “work-protect mode” offered by the Portal and the NWBC (that is, to display a data-loss dialog box when the user closes the application without first saving the data).

To achieve this, the application must ‘tell’ the FPM whether it contains unsaved (“dirty”) data. For this, the FPM provides the Web Dynpro Interface IF_FPM_WORK_PROTECTION. It contains only one method, which is described in the table below:

**METHODS**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_DIRTY</td>
<td>This interface can be implemented by any Web Dynpro component in your application which is known to the FPM (for example any UIBB or a shared-data component). At runtime, the FPM will detect all components implementing this interface. If any of these components signals unsaved data, then the application is marked as ‘dirty’. This application ‘dirty-state’ is then passed on by the FPM to the shell (that is, the Portal or the NWBC).</td>
</tr>
</tbody>
</table>
The shell-API requires this information as soon as the application state changes. Therefore, the `IF_FPM_WORK_PROTECTION~IS_DIRTY` method is called by the FPM runtime during each roundtrip. Therefore, it needs to perform this very quickly. Note that the FPM does not necessarily call the method `IS_DIRTY` on all UIBBs that are currently visible. As soon as one UIBB informs the FPM that it has unsaved data, the FPM does not need to call the method on the remaining visible UIBBs. For this reason, do not assume that the `IS_DIRTY` method is called by the FPM on all visible UIBBs.

Your application can use the sample code shown below:

```abap
METHOD is_dirty.
  if * component contains unsaved data
    ev_dirty = abap_true.
  else.
    ev_dirty = abap_false.
  endif.
ENDMETHOD.
```

**FPM Message Management**

FPM message management is an integral part of FPM and is available to all applications that use the standard floorplans. It guarantees consistent and guideline-compliant message handling.

**Features**

FPM message management consists of two parts:

- **IF_FPM_MESSAGE_MANAGER Interface (Message Manager)**

  This interface provides you with methods to perform the following tasks:

  - Clear messages
  - Raise Exceptions
  - Report messages

- **Message Region**

  All messages to be reported are displayed in the *Message Region*. This UI element is included in all FPM applications.

  You can make the following changes to the Message Region in the *Global Settings* dialog box:

  - Set the maximum message size
When the application displays your messages, the message area expands to accommodate the number of messages that you enter in the Maximum Message Size field. Once the number of messages exceeds the maximum limit, a scroll bar appears in the message area. Thus you can view messages other than those immediately visible in the message area.

- Turn on the message log

You can produce a log of the messages for your application. When the message log is turned on, all the previously reported messages can be seen. When a message is to be reported, the Display Message Log link appears in the Message Region. Note that this link appears only when there is at least one message in the log.

Note: You can also turn on the message log by using the URL parameter FPM_SHOW_MESSAGE_LOG=X. However, if you turn on the message log in the Global Settings dialog box, you cannot turn it off using the URL parameter.

Using the FPM Message Manager

Procedure

1. In the Component Controller of your Web Dynpro Component, choose the Attributes tab.
2. Declare an attribute of the component globally (for example MR_MESSAGE_MANAGER) and declare the Associated Type as type IF_FPM_MESSAGE_MANAGER.
3. Choose the Attributes tab of your Component Controller. In the Web Dynpro DOINIT method, create a handle to the FPM Message Manager (which is a read-only attribute in the IF_FPM interface), as detailed in the code below:

```
Method Web Dynpro DOINIT
  "Get the handle to the IF_FPM interface
  Web Dynpro_this->MR_FPM = CL_FPM_FACTORY->GET_INSTANCE( )
  Web Dynpro_this->MR_MESSAGE_MANAGER = Web Dynpro_this->MR_FPM->MO_MESSAGE_MANAGER
Endmethod
```

T100 based message. This example is taken from the demo applications and can be found in the Web Dynpro component FPM_HELLOSFLIGHT_OIF_DEMO in the APB_FPM_DEMO package.
CALL METHOD Web_Dynpro_THIS->MR_MESSAGE_MANAGER->REPORT_T100_MESSAGE
EXPORTING
  IV_MSGID = 'APB_FPM_DEMO'
  IV_MSGNO = 009
  IO_COMPONENT = Web_Dynpro_this
  IV_SEVERITY =
    if_fpm_message_manager=>GC_SEVERITY_ERROR
  IV_LIFETIME =
    if_fpm_message_manager=>GC_LIFE_VISIBILITY_AUTOMATIC
  IV_PARAMETER_1 = lv_carrid_string
  IO_ELEMENT = lo_el_sflight_selection
  IV_ATTRIBUTE_NAME = `CARRID`.

When the message appears in the Message Region, the parameter &1 is replaced by the actual flight name.

**IF_FPM_MESSAGE_MANAGER Interface**

This programming interface provides you with methods for controlling message management in your FPM application in a logical manner.

It provides you with methods to perform the following tasks:

- Reporting messages

  There are three methods available to report messages (including T100 and Bapiret2 messages).

- Raising exceptions

  There are four methods available to raise exception messages (including T100 and Bapiret2 messages).

- Clearing messages

  There is one method available to clear all messages.

**Methods for Reporting Messages**

The methods for reporting messages are provided by the **IF_FPM_MESSAGE_MANAGER** interface. This interface provides the following methods for reporting messages:

- REPORT_MESSAGE
- REPORT_BAPIRET2_MESSAGE
- REPORT_T100_MESSAGE
- REPORT_OBJECT_MESSAGE

Note the following information relating to all reporting methods:

- By default, the message is not mapped to a context element.
• If there are minor inconsistencies while reporting the message, FPM automatically takes alternative action (unless an exception is raised). The following is the alternative action that FPM takes: If the message is reported to be bound to a context element and if the element or the attribute is missing, FPM reports the message without the binding.

• FPM raises an exception in the following cases:
  o If the message lifetime is marked to be bound to a controller, but the controller is NULL or not reachable.
  o If the component for the message is missing.
  o If the Message Lifetime is set to Manual and View, but the element or attribute is missing.

Attributes

The attributes of the three methods for reporting messages are described in the table below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Relevant Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_COMPONENT</td>
<td>All</td>
<td>The severity of the message to be reported. There are three possible values, as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Passes an object reference to the message manager. This object reference is used to store the message. Preferably, the Web Dynpro component, which raises the message, must be passed here. You can pass another object reference only in the event of exceptions where the object raising the message does not have a handle to the Web Dynpro component (for example an ABAP OO class). This is important for those messages whose lifetime is maintained manually by the application. (see IV_LIFETIME). When you create a message whose lifetime is manual, the application creating such a message must then delete the message once it is no longer needed. In this case, you must pass the component whose messages need to be cleared. This helps to prevent messages from a different component being cleared by a component that has not raised them. This could happen when you re-use components from different areas.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Relevant Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error (E)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Warning (W)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Success (I)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is Error. These messages affect the navigation in different ways for each floorplan. Thus, navigation relating to an error message in a GAF application may be different to navigation relating to an error message in an OIF application. The following three values can be passed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GC_SEVERITY_ERROR for Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GC_SEVERITY_WARNING for Warning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GC_SEVERITY_SUCCESS for Success</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is an optional parameter. The default is Error.</td>
</tr>
<tr>
<td>IV_LIFETIME</td>
<td>All</td>
<td>Determines when, where and how long a message appears for. This is a very important parameter and must be given special attention. This parameter is a combination of the following two elements:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lifetime: Determines how long the message exists in the message area; that is, the creation and deletion of the message. The available lifetimes are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• automatic: FPM handles the destruction of the message as defined by the UI guidelines for the floorplan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Manual: the application developer handles the deletion of the message from the message area.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Visibility: Determines when the message appears in the message area. The following values apply:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Automatic: FPM takes care of the</td>
</tr>
<tr>
<td>Parameter</td>
<td>Relevant Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>visibility based on the UI guidelines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>View</strong>: the message is visible as long as the view to which the message is bound is available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Controller</strong>: the message is visible as long as the controller that has raised the message is available (see the parameter controller IO_Controller for details)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Application</strong>: the message is permanently displayed (until it is deleted manually by the application developer) and is visible whilst the application is running.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Pop-up</strong>: the message is visible only in a dialog box.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default values for both <strong>Lifetime</strong> and <strong>Visibility</strong> are <strong>Automatic</strong>. Not all combinations of lifetime and visibility are possible. Some combinations, for example <strong>Lifetime</strong> = <strong>Manual</strong> + <strong>Visibility</strong> = <strong>Pop-up</strong> are not available. The permitted combinations are as follows (showing the constant to be used - <strong>Lifetime</strong> + <strong>Visibility</strong>):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>GC_LIFE_VISIBILITY_AUTOMATIC</code>: <strong>Automatic</strong> + <strong>Automatic</strong> (Fully handled by FPM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>GC_LIFE_VISIBILITY_AUT_Dialog Box</code>: <strong>Automatic</strong> + <strong>Pop-up</strong> (Creation and destruction handled by FPM; visible as long as the dialog box is visible)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>GC_LIFE_VISIBILITY_MANUAL_VIEW</code>: <strong>Manual</strong> + <strong>View</strong> (Should be deleted by the application; visible until the view that created it is visible)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>GC_LIFE_VISIBILITY_MANUAL_CONTROLLER</code>: <strong>Manual</strong> + <strong>Controller</strong> (Should be deleted by the application; visible as long as the controller that created it is visible)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Relevant Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IV_PARAMETERS</td>
<td>All</td>
<td>A group of parameters of the type Web Dynpro R_NAME_VALUE_LIST that can be stored along with the message. This will be passed to the Web Dynpro message manager as is and will have no visualize changes to the message. Refer to the Web Dynpro message manager documentation for further details.</td>
</tr>
<tr>
<td>IR_MESSAGE_USER_DATA</td>
<td>All</td>
<td>Additional data that can be stored along with the message. This does not influence the message visually. This parameter can be used by the application developers to provide error resolution mechanism. See the Web Dynpro help for further details.</td>
</tr>
<tr>
<td>IV_MESSAGE_INDEX</td>
<td>All</td>
<td>Numerical value indicating the order in which the message is to be displayed. If no value is passed (this is an optional parameter), the message appears in the order in which the Web Dynpro runtime chooses to display it. Messages are sorted for display, according to the following attributes: Error severity Message index (parameter MSG_INDEX) Context element (if it exists) Context attribute (if it exists)</td>
</tr>
<tr>
<td>IO_ELEMENT</td>
<td>All</td>
<td>A reference to a context element to which the message is bound. The message is then clickable and the focus shifts to a UI element bound to this context element.</td>
</tr>
<tr>
<td>IV_ATTRIBUTE_NAME</td>
<td>All</td>
<td>The element attribute to which the message must be mapped. This</td>
</tr>
<tr>
<td>Parameter</td>
<td>Relevant Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IV_IS_VALIDATION_INDEPENDENT</td>
<td>All</td>
<td>Defines whether a message, referring to a context attribute or a context element, influences the execution of a standard action. If the parameter’s value is ABAP_FALSE (default value), the standard action is no longer executed after this message is created. However, if the parameter’s value is ABAP_TRUE, the standard action is executed.</td>
</tr>
<tr>
<td>IO_CONTROLLER</td>
<td>All</td>
<td>Pass the reference to the controller whose lifetime will dictate the lifetime of the messages which have the lifetime set to the context.</td>
</tr>
<tr>
<td>IS_NAVIGATION_ALLOWED</td>
<td>All</td>
<td>Use this flag if you need to allow navigation, even on an ‘E’ message in the GAF. Relevant for GAF applications only.</td>
</tr>
<tr>
<td>IV_VIEW</td>
<td>All</td>
<td>The name of the view of the dialog box. The error is then restricted only to the dialog box. Otherwise there is a side effect in that the error message (if a non-automatic type) is also reported on the main screen when the dialog box is closed. Relevant only if the message manager is used in application-specific dialog boxes.</td>
</tr>
<tr>
<td>IV_MESSAGE_TEXT REPORT_MESSAGE</td>
<td>REPORT_MESSAGE</td>
<td>Any free text that must be reported in the message area. When used with the UI element and attribute parameters, it becomes a clickable free text message.</td>
</tr>
<tr>
<td>IS_BAPIRET2 REPORT_BAPIRET2_MESSAGE</td>
<td>REPORT_BAPIRET2_MESSAGE</td>
<td>The BAPIRET2 structure directly in the message. The severity of the message is automatically selected from the BAPIRET2 structure. The T100 message that is embedded in the BAPIRET2 structure is used to display the message text. Additionally, the lifetime, visibility and context mapping can be set along with the</td>
</tr>
</tbody>
</table>
Parameter | Relevant Method | Description
--- | --- | ---
IR_MESSAGE_OBJECT | REPORT_OBJECT_MESSAGE | Exception Object providing access to message long text.
IT_ATTRIBUTES | All | Table of attributes to which the message is mapped. Note that this parameter overrides the IV_ATTRIBUTE_NAME if both parameters are set.
IT_CUSTOMIZING_PARAMETERS | ALL | 1. Optional Parameter of type String table. If the message needs to be mapped to a context in ALV, then include constant GC_REPORT_IN_ALV (present in interface IF_FPM_MESSAGE_MANAGER) to this importing parameter.
| | | 2. Optional Parameter of type String table. When a permanent message needs to be reported only in the view and not in any dialog boxes, then include constant GC_NAVIGATE_ERROR (present in interface IF_FPM_MESSAGE_MANAGER) to this importing parameter.
IV_MSGID | REPORT_T100_MESSAGE | Used when reporting a T100 based message. Supply the parameter with the message class.
IV_MSGNO | REPORT_T100_MESSAGE | The message number of the message class specified by the IV_MSGID.
IV_PARAMETER_1, IV_PARAMETER_2, IV_PARAMETER_3, IV_PARAMETER_4 | REPORT_T100_MESSAGE | Optional parameters for the message.

**Mandatory Parameters**

The table below shows which parameters are mandatory for each method:

<table>
<thead>
<tr>
<th>Method</th>
<th>Mandatory Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT_MESSAGE</td>
<td>Message text</td>
</tr>
<tr>
<td>REPORT_T100_MESSAGE</td>
<td>Message class and message number</td>
</tr>
</tbody>
</table>
Methods for Raising Exception Messages

The RAISE_EXCEPTION methods are provided by the IF_FPM_MESSAGE_MANAGER interface. This interface provides the following methods for raising exceptions:

- RAISE_EXCEPTION
- RAISE_T100_EXCEPTION
- RAISE_CX_ROOT_EXCEPTION
- RAISE_BAPIRET2_EXCEPTION

All exceptions are logged into the system with the following details:

- the method that was used to raise the exception
- the text of the exception
- additional text (if used)

From SP13 onwards, there is no recovery mechanism from the exceptions.

Attributes

The following table describes the attributes of the four RAISE_EXCEPTION methods.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Relevant Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV_TEXT</td>
<td>RAISE_EXCEPTION</td>
<td>Optional text that can be passed while raising a simple exception. This text is logged and can later be used for analysis</td>
</tr>
<tr>
<td>IV_MSGID</td>
<td>RAISE_T100_EXCEPTION</td>
<td>Message class ID for the T100 message. Use this parameter to raise an exception whose text is based on the T100 message mechanism.</td>
</tr>
<tr>
<td>IV_MSGNO</td>
<td>RAISE_T100_EXCEPTION</td>
<td>Message number of the T100 message class.</td>
</tr>
<tr>
<td>IV_PARAMETER_1</td>
<td>RAISE_T100_EXCEPTION</td>
<td>Optional message parameters.</td>
</tr>
<tr>
<td>IV_PARAMETER_2</td>
<td>RAISE_T100_EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>IV_PARAMETER_3</td>
<td>RAISE_T100_EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>IV_PARAMETER_4</td>
<td>RAISE_T100_EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>IO_EXCEPTION</td>
<td>RAISE_CX_ROOT_EXCEPTION</td>
<td>The exception class inheriting from CX_ROOT. This parameter is a mandatory parameter.</td>
</tr>
<tr>
<td>IV_ADDITIONAL_TEXT</td>
<td>RAISE_CX_ROOT_EXCEPTION</td>
<td>Additional text to be added while reporting an exception based on</td>
</tr>
</tbody>
</table>
Parameter Relevant Method Description
---
IS_BAPIRET2 RAISE_BAPIRET2_EXCEPTION The BAPIRET2 structure for raising an exception.

### Method for Clearing Messages

This method is provided by the `IF_FPM_MESSAGE_MANAGER` interface. Note the following information relating to this method:

- The method clears messages from the Message Region and acts upon all those methods that have Lifetime set to Manual.
- This is the only method to selectively clear those messages with a Lifetime set to Manual from the Message Region.
- This method ensures that messages from a different component are not cleared accidentally.
- The defaults for the parameters contain a negative semantic with respect to the method name; if the method is called with defaults, all the messages are deleted.

### Attributes

The following table describes the attributes for the `CLEAR_MESSAGES` method.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Relevant Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_COMPONENT</td>
<td>CLEAR_MESSAGES</td>
<td>The component in which messages were previously reported. Only those messages that were reported from this component will be cleared. If this contains object references other than components, then those object references will be used. This is a mandatory parameter.</td>
</tr>
<tr>
<td>IV_EXCLUDE_ERROR</td>
<td>CLEAR_MESSAGES</td>
<td>Pass true if error messages belonging to the component are not to be deleted. This is an optional parameter and the default is false. This means that all the error messages belonging to this component will be deleted unless this parameter contains a true value. Looking at the parameter’s name, the parameter indicates that the default value (false) has to be overridden only if error messages are to be saved from being cleared and this parameter contains negative semantic with respect to the method name.</td>
</tr>
<tr>
<td>IV_EXCLUDE_WARNING</td>
<td>CLEAR_MESSAGES</td>
<td>Default is false. Override it with true, if warnings raised for the component are to be saved.</td>
</tr>
<tr>
<td>IV_EXCLUDE_SUCCESS</td>
<td>CLEAR_MESSAGES</td>
<td>Default is false. Override it with true, if success messages raised for the component are to be saved.</td>
</tr>
</tbody>
</table>
Handling of FPM Message Manager in Non-FPM Dialog Boxes

The handling of FPM Message Manager is also possible in Non-FPM dialog boxes (that is in application specific dialog boxes). FPM handles the messages with respect to the parent component and the dialog box in terms of visibility and lifetime of a message.

<table>
<thead>
<tr>
<th>Lifetime/Visibility</th>
<th>Message Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>FPM takes care of the visibility based on the UI guidelines. The automatic messages in the dialog box get cleared after every roundtrip or if a new message is raised on a dialog box.</td>
</tr>
<tr>
<td></td>
<td>This behavior is similar to the automatic messages in the parent window.</td>
</tr>
<tr>
<td>View</td>
<td>The message is visible as long as the view to which the message is bound is available; this message is not transferred to the parent window if the dialog box is closed.</td>
</tr>
<tr>
<td>Controller</td>
<td>The message is visible as long as the controller that has raised the message is available. This message would be passed to the parent if it is raised through the same controller.</td>
</tr>
<tr>
<td>Application</td>
<td>The message is permanently displayed, in the dialog box and the parent window throughout the application, until it is manually cleared by the application developer.</td>
</tr>
<tr>
<td>Pop-up</td>
<td>Pop-up: The message is visible only in a dialog box as long as the Non-FPM dialog box is open.</td>
</tr>
</tbody>
</table>

FPM Message Manager FAQ

1. Can I use the Web Dynpro message manager along with the FPM message manager?
   Yes. However, you create and maintain your own reference of the Web Dynpro message manager. Messages that are reported directly into the Web Dynpro message manager will not be maintained by FPM after they are reported and the application must handle the message independent of the FPM lifetime and visibility functions. Exceptions logged directly into the Web Dynpro message manager are not logged under the FPM_RUNTIME_MESSAGES checkpoint group.

2. I want to use the FPM floorplan but I do not want to use the FPM message area. Can I do this?
   Yes. Use the Web Dynpro message area. However, FPM message manager functions such as automatic lifetime handling, consolidated dialog box display etc is not then available.

3. Should I create a message area to use the FPM message manager?
   No. If you are using a standard floorplan (for example OIF or GAF), the message area is a standard part of an FPM application’s UI.

4. Can I change the position of the message area?
   No. If you create an additional message area, the messages are repeated in both message area views.

5. I reported a message mapped to a context. I see only the text and the message is not navigable. What is happening?
The element and the attribute do not contain valid references. In such a case, FPM still displays the message but it is not navigable.

6. When I raise an exception, the screen dumps. When I examine the stack I see that the IF_FPM_MESSAGE_MANAGER is the point where the dump occurs. Why?

As of SP13 of NW 7.00 and SP03 of NW7.10, there is limited support for exception handling for FPM applications. Features such as recovery mechanisms from exceptions, special exception screens, etc are not available. All RAISE_XX_EXCEPTION methods in FPM will log any exception raised from the method and then force a dump. In this manner, the applications are terminated.

Message Mapper

Messages that are displayed by the current system can contain technical terms which might not be understood by everyone who work with the system. Message Mapper is a feature that is used to map messages (error messages, warning messages, and information/success messages) that are currently displayed by the system to a more understandable, user-friendly form.

With Message Mapper you can perform the following activities:

- Map messages to an alternative message to be displayed by the system
- Group messages into specific categories and have the system display an alternative message for the category
- Hide messages
- Log messages in an application log

Message mapping can be performed by both SAP applications and customer applications.

Enabling Message Mapper


Message Mapping Fields

Message Context

Message Mapper can be called from different contexts of applications, for example, from ESS, HRAS, PLM, and SRM and so on. Applications can specify their own contexts. The message context is a mandatory parameter.
Applications can also use the context to describe a role. Roles can also be described as part of the context or category. The application decides the context and category based on its own requirements.

A context can be, for example SRM, CRM, or ERP depending on the application that is using the Message Mapper. A context could also be a role such as Employee or Manager with a specific category to differentiate the roles further.

**Message Categories**

You can group messages into categories. Message categories can be created by SAP applications and customers’ applications. Customers cannot delete categories created by SAP applications, but they can add new entries based on their own requirements.

You can use categories to specify, for example, the following items: roles such as Administrator, Manager, and Employee; technical groups such as No Authorization group or a Wrong Customizing group); functional groups such as an SRM Shopping Cart Customizing group.

Examples:

1. In the context of SRM, CRM, ERP and so on, you can create a category called No Authorization. You can use it to display the alternate message “You are not authorized to perform the changes” for all messages belonging to the No Authorization category.

2. In the context of SRM, CRM, ERP and so on, you can create a category called Employee. You can use it to hide all warning messages from employees.

**Message Namespace**

There is a separate namespace for SAP and customer message mappings. Customers can override the message mapping made by SAP applications if such mappings are not indicated as final (Final is a field in the message mapping table (a table containing all mapping entries)). Customer mappings have priority over SAP mappings. The namespace is not part of the message mapping API, but is a part of the message mapping table. The customer namespace begins with “Z” or “Y”.

**Message Source**

You can map only T100 and BAPIRET2 messages; that is, the original messages passed to the Message Mapper should be either T100 or BAPIRET 2 Messages. The T100 or BAPIRET2 messages can be mapped to T100 or OTR or free text messages.

**Generalization**
Generalization refers to the process used by the Message Mapper to match system messages to alternative messages. The way messages are mapped to system messages varies; messages can be mapped specifically by specifying all fields of a message, or they can be mapped more broadly by specifying fewer fields. When the application later reports a message, the Message Mapper checks which fields are present in the reported message and whether there is a specific mapping for this collection of fields. The Message Mapper starts the search for a mapping using the lowest level of generalization, G0, (it checks to see if there is a mapping which includes all the fields) and it continues with a higher level of generalization until an alternate message mapping is found. If all generalization levels are searched, and no mapping is found, the original message is reported.

By default, the Generalization Type is Type 1.

Alternate messages are either T100 or OTR or free text messages. Applications can specify alternate messages for a Message ID (within a particular context) without the message number.

You can provide alternate messages for the field combinations outlined in the following table:

<table>
<thead>
<tr>
<th>Generalization Level</th>
<th>Message Fields</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G8</td>
<td>Context</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G7</td>
<td>Context</td>
<td></td>
<td>Category</td>
<td></td>
</tr>
<tr>
<td>G6</td>
<td>Context</td>
<td></td>
<td>Category</td>
<td>Message Type</td>
</tr>
<tr>
<td>G5</td>
<td>Context</td>
<td></td>
<td>Category</td>
<td>Message Type</td>
</tr>
<tr>
<td>G4</td>
<td>Context</td>
<td>Message Id</td>
<td></td>
<td>Message Type</td>
</tr>
<tr>
<td>G3</td>
<td>Context</td>
<td>Message Id</td>
<td>Message No</td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td>Context</td>
<td>Message Id</td>
<td>Message No</td>
<td>Message Type</td>
</tr>
<tr>
<td>G1</td>
<td>Context</td>
<td>Message Id</td>
<td>Message No</td>
<td>Category</td>
</tr>
<tr>
<td>G0</td>
<td>Context</td>
<td>Message Id</td>
<td>Message No</td>
<td>Category</td>
</tr>
</tbody>
</table>
MESSAGE MAPPER FLOW

START

Mapping found for the specified criteria

YES

Hide message?

NO

Is Generalization allowed?

YES

User has specified option for Generalization

YES

Get the path for the option specified by the user

NO

Get the default option and the corresponding path

NO

Mapping found while generalizing?

YES

Hide message?

NO

Stop Generalization

NO

Log the message (if logging is enabled)

Return original

A

YES

Mapping found for the specified criteria

YES

Log the message (if logging is enabled)

A

NO

Log the message (if logging is enabled)

Do not report message

A

NO

Map original message parameters to alternate message

Log the message (if logging is enabled)

Report the alternate message

STOP
Changing Message Types

You can change the message type of the original message to a different message type in the alternate message. If no alternate message type is described in the message mapping table, the message is reported with its original message type.

Hiding Messages

You can choose to hide specific messages by selecting the Hide option when you map the alternate message in the message mapping table.

Hiding Messages and Generalization

If you want to hide a message and use the generalization option, then the Hide on Generalization option must be selected. To do this, choose Display -> Message Mapper Settings in the component configuration.

During generalization, if the Hide option is encountered, the system checks whether the Hide on Generalization option is also selected in the configuration editor. If it is not selected, no further generalization occurs and the original message is returned.

If you specify an alternate message and select the Hide option for a particular mapping, hiding will take precedence and the alternate message is not reported.

Logging Messages

Mapped messages are logged under a separate group name. Only the original message is logged unless generalization has occurred. If generalization has occurred, both the original and the alternate message are logged.

Message logging is inactive by default, but you can control logging using the following methods:

- **Message Mapper Parameters**

  The following logging options are found in the Message Mapper Settings dialog box, in the Message Mapper Parameters section:

  - **Always Enable Logging**
    All messages that are mapped are logged.

  - **Log on Generalization**
    Messages are logged only if mapping is not present for a specified criteria and a subsequent level of generalization is considered.

  - **Log on Hide**
    Messages are logged only if they are hidden.

- **URL Parameters**
You can also use the following URL parameters to log messages:

- **FPM_ALWAYS_LOG**
  This URL parameter controls message logging. If this parameter is set to X, then logging is always enabled.

- **FPM_LOG_ON_GENERALIZATION and FPM_LOG_ON_HIDE**
  These parameters are available for Log on Generalization and Log on Hide respectively.

In addition to the above, the BAdI **FPM_BADI_LOG_MAPPED_MSG** is provided in which you can write specific logic for message logging (for example, specific message logging based on roles). If the BAdI is implemented, then the logging details written in the BADI take precedence over the logging details specified as default, irrespective of the options selected in the configuration editor.

Mapped messages are logged under a separate group name in transaction SLG1. Use the following entries to analyze the log:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>FPM_MSG_MAPPER</td>
</tr>
<tr>
<td>Sub-Object</td>
<td>RUNTIME</td>
</tr>
</tbody>
</table>

**Generalization**

You can set the Generalization Type in the Message Mapper Settings dialog box, under the Message Mapper Parameters section. The following Generalization Types exist:

- Type 0 (no Generalization)
- Type 1 (Default Generalization)

**Type 1: Default**

<table>
<thead>
<tr>
<th></th>
<th>Context</th>
<th>Message Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>G8</td>
<td>Context</td>
<td></td>
</tr>
<tr>
<td>G7</td>
<td>Context</td>
<td>Message Type</td>
</tr>
<tr>
<td>G6</td>
<td>Context</td>
<td>Category</td>
</tr>
<tr>
<td>G5</td>
<td>Context</td>
<td>Message Type</td>
</tr>
<tr>
<td>G4</td>
<td>Context</td>
<td>Category</td>
</tr>
<tr>
<td>G3</td>
<td>Context</td>
<td>Message Type</td>
</tr>
<tr>
<td>G2</td>
<td>Context</td>
<td>Message Type</td>
</tr>
<tr>
<td>G1</td>
<td>Context</td>
<td>Category</td>
</tr>
<tr>
<td>G0</td>
<td>Context</td>
<td>Category</td>
</tr>
</tbody>
</table>

The options mentioned above can be specified by a domain **FPM_MSG_GENERALIZE**, which has the following value range:

- FPM00 - No Generalization
If you do not require generalization, then option FPM00 needs to be passed; the alternate message (if present) is returned for the specified criteria, otherwise the original message is returned.

- **FPM01 - Type 1 – Default Generalization**
  If you do not pass any option, then the default option is FPM01 generalization.

**Example**

A message is reported from an application with parameters corresponding to G0 level of Generalization:

Context: **SRM**  
Message ID: **EBP**  
Message No: **123**  
Category: **No Authorization**  
Message Type: **E**

**FPM00 - No Generalization**

If the entries exist for the above criteria in the message mapping customizing table, then a corresponding alternate message is reported. For example, if an alternate message mapping exists in the customizing table for mapping at G0 level (that is, for all the corresponding fields) with the alternative message text *No Authorization to change Purchase Order*, this alternate message is displayed.

If no alternate message mapping exists in the message mapping table, then the original message is displayed.

**FPM01 (Default Generalization)**

If no mappings exist in the customizing table for mapping at G0 level, but there is an entry at, for example, G5 level for the same *Context, Category and Message Type*, the generalization concept is adopted automatically and the alternate message text at G5 level is displayed.

**Mapping Message Variables**

Message variables in the original message are copied into the corresponding message variables in the alternate message, according to the following conditions:

- If the original message contains message variables and the alternate message does not contain message variables, the message variables are not displayed.
- If the alternate message and the original message contain the same number of message variables, there is a one to one mapping between the message variables; message variable &1 in the alternate message is copied to message variable &1 in the original message, message variable &2 in the alternate message is copied to message variable &2 in the original message, and so on.
- If the alternate message does not contain the same number of message variables as that of the original message, then only the message variables described in the alternate message are displayed and there is a one to one mapping of the same, that is, if the original message has message variables &1, &2, and &3 and the alternate message has only message variables &2, then only message &2 is shown in the alternate message and the remaining message variables are ignored.
If the alternate message has variables & & & & instead of &1 &2 &3 &4, then the first variable & corresponds to message variable &1, the second variable & to message variable 2, the third variable & to message variable 3 and so on.

**API Changes for Message Mapping**

The following additional parameters are available for message mapping in the Message Manager API:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Opt</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV_CONTEXT</td>
<td>Importing</td>
<td>Y</td>
<td>FPM_S_MSG_MAPPER-MSG_CONTEXT</td>
</tr>
<tr>
<td>IV_CATEGORY</td>
<td>Importing</td>
<td>Y</td>
<td>FPM_S_MSG_MAPPER-MSG_CATEGORY</td>
</tr>
<tr>
<td>IV_GENERALIZATION</td>
<td>Importing</td>
<td>Y</td>
<td>FPM_MSG_GENERALIZE</td>
</tr>
</tbody>
</table>

When reporting a message from an application, you can pass these optional parameters to use in message mapping. Example values are shown below:

IV_CONTEXT = SRM
IV_CATEGORY = Limit exceeded
IV_GENERALIZATION = FPM01 (Default Generalization)

The following methods have been enhanced with the above optional parameters corresponding to the Message Mapper:

- IF_FPM_MESSAGE_MANAGER~REPORT_T100_MESSAGE
- IF_FPM_MESSAGE_MANAGER~REPORT_BAPIRET2_MESSAGE

**Customizing Tables for Message Mapper**

The following tables are available for message mapping:

- **FPM_T_MSG_MAPPER**
  - This is the main customizing table for mapping messages.
- **FPM_MSGCATEGORY**
  - This table is for customizing message categories.
- **FPM_CATEGORY**
  - This is a text table for FPM_MSGCATEGORY.

The fields for each table are described in detail below.

**Table FPM_T_MSG_MAPPER**

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Type</th>
<th>Description</th>
<th>Field Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMESPACE</td>
<td>FPM_NAMESPACE</td>
<td>Namespace for Message Mapper.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>MSG_CONTEXT</td>
<td>FPM_CONTEXT</td>
<td>FPM application context name. Indicates, for example, the business unit or role in which the message is mapped.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>MSGID</td>
<td>SYMSGID</td>
<td>Message class of system message.</td>
<td>Optional</td>
</tr>
</tbody>
</table>
MSGNO | SYMSGNO | Message number of system message. | Optional
--- | --- | --- | ---
MSGTY | SYMSGTY | Message type of system message. | Optional
MSG_CATEGOR Y | FPM_MSGCATEGORY | Message category of system message. | Optional
ALT_MSGID | SYMSGID | Message class for alternate message. | Optional
ALT_MSGNO | SYMSGNO | Message number for alternate message. | Optional
ALT_MSG | FPM_ALT_MSG | Alternate message. Type your message text here. | Optional
ALT_MSGTY | SYMSGTY | Message type for alternate message. | Optional
HIDE_MSG | BOOLEAN | Hides the message. | Optional
FINAL | BOOLEAN | Ensures that the message mapping cannot be edited. | Optional

Table FPM_MSGCATEGORY

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Type</th>
<th>Description</th>
<th>Field Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_CATEGORY</td>
<td>FPM_MSGCATEGORY</td>
<td>Message category.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>NAMESPACE</td>
<td>FPM_NAMESPACE</td>
<td>Namespace for Message Mapper.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

Table FPM_CATEGORYT

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Type</th>
<th>Description</th>
<th>Field Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_CATEGORY</td>
<td>FPM_MSGCATEGORY</td>
<td>Message category.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>NAMESPACE</td>
<td>FPM_NAMESPACE</td>
<td>Namespace for Message Mapper.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>LANGU</td>
<td>SPRAS</td>
<td>Language key.</td>
<td>Optional</td>
</tr>
<tr>
<td>MSG_CATEGORY_ NAME</td>
<td>FPM_MSGCATEGORY_ NAME</td>
<td>Description of the message category.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Maintenance Views for Message Mapper

The following views are available for message mapping:

- **FPM_V_MSG_MAPPER:**
  The maintenance view for table FPM_T_MSG_MAPPER. When a SAP application has marked an entry as Final, then it cannot be overridden by Customers. NAMESPACE and MSG_CONTEXT fields are mandatory.

- **FPM_VMSGCATEGORY:**
  The maintenance view for table FPM_MSGCATEGORY and FPM_CATEGORYT. NAMESPACE and MSG_CATEGORY fields are mandatory.
**FPM Error Page**

The Error Page allows FPM applications to exit from the application in a clean way.

Sometimes an application may face a serious problem and it is not possible to continue further. In that case application can display an error page which describes the problem to the end user. No further navigation is possible after navigating to error Page.

The picture below shows an example of how error page will look.

---

**Structure**

The structure of error page is as follows:

- **The Title is always Error.**
  This cannot be influenced by different applications.
- **A short text or short description about the problem can be displayed in message area.**
- **The Error Details part will have sub parts to display more information**
  - Technical Exception will display the technical problem as provided by the application.
  - Error description will display the long text of the problem.
  - Additional information will display additional information in form of Knowledge Management doc created using transaction SE61 that applications might want to display.

---

**Features**

The features of error page are:
• Navigation to error page via API

The IF_FPM interface has an additional DISPLAY_ERROR_PAGE method. This method can be called from application to display the error page.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY_ERROR_PAGE</td>
<td>IO_ERROR_DETAILS– _type IF_FPM_ERROR_DETAILS</td>
<td>This method is called from application passing the error data, to display the error page.</td>
</tr>
</tbody>
</table>

• Support for T100 messages, Bapiret2, Exception object and OTR to pass error data.

There is a factory class for creating error data. This factory class will enable the applications to create error data in a structured format using any of the following sources, which will be displayed on the error page.

Class: CL_FPM_ERROR_FACTORY.

Table: Static methods of class: CL_FPM_ERROR_FACTORY

<table>
<thead>
<tr>
<th>Method name</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_FROM_BAPIRET2</td>
<td>IS_BAPIRET2, IV_TECHNICAL_EXCEPTION, IV_ADDITIONAL_INFO, IV_ERROR_ID, RO_ERROR_DETAILS</td>
<td>It is used to created error data if the application has error information in form of bapiret2.</td>
</tr>
<tr>
<td>CREATE_FROM_T100</td>
<td>IS_T100, IV_TECHNICAL_EXCEPTION, IV_ADDITIONAL_INFO, IV_ERROR_ID, RO_ERROR_DETAILS</td>
<td>It is used to created error data if the application has error information in form of T100 table.</td>
</tr>
<tr>
<td>CREATE_FROM_OBJECT</td>
<td>IO_EXCEPTION_OBJ, IV_TECHNICAL_EXCEPTION, IV_ADDITIONAL_INFO, IV_ERROR_ID, RO_ERROR_DETAILS</td>
<td>It is used to created error data if the application has error information in form of exception object.</td>
</tr>
<tr>
<td>CREATE_FROM_OTR</td>
<td>IV_OTR</td>
<td>IV_TECHNICAL_EXCEPTION</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td></td>
<td>IV_ADDITIONAL_INFO</td>
<td>IV_ERROR_ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RO_ERROR_DETAILS</td>
</tr>
</tbody>
</table>

It is used to created error data if the application has error information in form of OTR.

<table>
<thead>
<tr>
<th>CREATE_FROM_EMPTY_DATA</th>
<th>IV_ERROR_ID</th>
<th>RO_ERROR_DETAILS</th>
</tr>
</thead>
</table>

To get error object when no error data is available.

```plaintext
data: lr_fpm type ref to if_fpm.

lr_fpm = cl_fpm_factory->get_instance( ).

**In case of T100 error data**

data: lv_t100 type symsg.
lv_t100-msgty = 'E'.
lv_t100-msgid = 'APB_FPM_DEMO'.
lv_t100-msgno = 10.

call method lr_fpm->display_error_page( cl_fpm_error_factory=>create_from_t100( is_t100 = lv_t100
iv_error_id = 'TEST_APPLICATION'
iv_technical_exception = 'Test:Null Object Exception'
iv_additional_info = 'FPM_DEMO_SFLIGHT_ADD_INFO')).
exit.
```

- Logs can be viewed using transaction SLG1

  Logs can be later seen using SLG1 transaction in the corresponding system under the Object: FLOORPLAN_MANAGER object, the RUNTIME sub object, or the External ID (Error ID if provided). The error ID can be passed from application while calling the error page. The log contains the long text and short text of the problem.

### Handling of Transactions

Transactions can be handled in a systematic manner in FPM by implementing the Web Dynpro interface **IF_FPM_TRANSACTION**. This interface guarantees you the following advantages:

- There is a logical sequence in which the interface methods are called.
- The transaction steps can be split up into the sequence in which they are supposed to be processed.
- There is a check – save – validate sequence that provides high transaction integrity.
- The check – save – fail – recover sequence provides the required robustness to the transaction.
Using the Transaction Interface

1. In the Web Dynpro ABAP Workbench, select a component that will contain the business logic to be executed on a save event. This could be any component known to the FPM (including any UIBB or Shared Data component used by the UIBBs of your application).
2. In the preview, choose Implemented Interfaces and, in edit mode, add the IF_FPM_TRANSACTION interface.
3. Save your entry.
4. In the Action column, choose the Reimplement button and ensure that the icon in the Implementation State column turns green.
5. Activate your component. In the Activation dialog box, ensure that all elements are selected and choose Save. You have now implemented the Transaction interface and if you open the Component Controller component, you can see the methods associated with it on the Methods tab.
6. In any Save event, data needs to be saved to a database. This can be realized in the following ways:
   - Use a shared data component (see section on shared data for further details).
   - Use direct context binding.
   - Use an assistance class.

   The decision to use any or a combination of the above methods is taken by the application developer.

Transaction Interface FAQ

- On what event will these methods be called? These methods will be invoked by the standard FPM SAVE event.
- Can I have the FPM call these methods on my own custom event? No. These methods are called as part of the standard FPM event loop and hence will not react to custom events.
- Can I have multiple components implement this interface in the same application? Yes. The FPM will call all the methods on all the implementing components. But our general recommendation is to use only one central component for transaction handling.

IF_FPM_TRANSACTION Interface

This Web Dynpro interface provides you with methods to handle transactions in a systematic manner by FPM. This is an optional interface; an application can handle the transactions independently, without implementing the interface.

Methods

The IF_FPM_TRANSACTION interface contains the methods described in the table below. Note that once the interface is implemented, the FPM identifies the corresponding component that has the method to be called, in sequence and calls the methods on this component in the same sequence as defined below.
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK BEFORE SAVE</td>
<td>This method has a return parameter which indicates whether the validation before a save to the database is successful. Use this method as a trial for saving, and return a true if the trial save was successful and false if it was not.</td>
</tr>
<tr>
<td>SAVE</td>
<td>This method is used to perform the actual save and any possible commit. It is called when the CHECK BEFORE SAVE has returned a false (note that the semantic of the return parameter of the CHECK BEFORE SAVE is negative and reads ‘rejected’. In other words, a false value for rejected means that the CHECK BEFORE SAVE was successful). If there are errors while saving, you must return rejected = true so that the AFTER REJECTED SAVE can be called. If the save was indeed successful, then the method AFTER COMMIT is called. Refer to the flow chart for more details.</td>
</tr>
<tr>
<td>AFTER COMMIT</td>
<td>You can perform cleanup activities such as releasing database locks, releasing other resources, triggering an event for processing after a successful commit.</td>
</tr>
<tr>
<td>AFTER REJECTED SAVE</td>
<td>Here you can perform your roll back activities. You can also release locks and resources.</td>
</tr>
</tbody>
</table>

The methods are called in the sequence depicted in the figure below:
A detailed sequence diagram of the method calls can be found in the FPM Design Document.
Resource Management

As of SAP NetWeaver 7.0 enhancement package 1 it is possible for UIBBs to be made transient in their behavior. Transient behavior means that UIBBs, which are not visible, can be removed from memory so as to increase the performance and the memory footprint of the application.

The transient behavior is applicable to OIF, GAF, and OVP floorplans.

In addition to freestyle UIBBs, this feature has been extended for Tabbed UIBBs, Form GUIBBs, List GUIBBs, and FPM dialog boxes.

With regard to the Tabbed GUIBB, transient behavior is applicable for

1. Pure Master-Detail (the application is built only on the Master Detail)
2. Mixed Master-Detail (one of the view switches is a Tabbed component)

With regard to FPM dialog boxes, the component is released only while closing a dialog box. This is because the parent UIBB is still displayed when opening a dialog box and therefore cannot be released.

Releasing a Component

Technically, a UIBB is an interface view and this, by itself, cannot be released from memory, hence the FPM releases the component containing the UIBB based on certain rules. These rules are as follows:

- The application must use the new schema available from EhP1 onwards.
- The application developer must have set the transient flag to true via the FPM Configuration Editor.
- The FPM framework finds that it is technically feasible to release the component.
- The UIBBs implement the Resource Manager interface and do not veto the transient decision passed by the application via the ON_HIDE method.
- The UIBB has not implemented the Resource Manager interface (meaning that it does not have the possibility to veto).

A UIBB is defined and identified by the following key: configuration + component + interface view.

The transient behavior can be specified only during design time at the level of the application and not at the level of a UIBB or its usage.

The transient behavior of the UIBBS can be handled in one of the following ways:

- One UIBB per component
The component contains only one interface view which is used as a UIBB. When the
UIBB is removed from the view assembly, the component that contains this UIBB is
released.

- Multiple UIBBs per component

The component is released only when all the interface views that behave as UIBBs are
no longer part of the view assembly, and the next set of UIBBs (for the forthcoming
view assembly) does not contain a UIBB from this component. In such a case, the
component is only released when all the interface views of this component are no
longer part of the visible view assembly. Note that when one of the interface views
(UIBBs) is removed from the view assembly, the component remains alive if other
interface views of the same component are still part of the view assembly or part of
the next view assembly.

If the application developer has set the global flag to transient, meaning that the UIBBs
(components) can be released, then the FPM will investigate whether the component can be
released.

There are instances when, even if the application developer has set the default to transient, the
component containing the UIBBs cannot be released. These instances are described below:

- The component is held as a used component by another component.
- There are UIBBs from the same component that is still being displayed.
- The component implements an FPM interface that does not allow it to be released.

Interfaces which restrict the release of the component are:

- IF_FPM_SHARED_DATA
- IF_FPM_TRANSACTION
- IF_FPM_APP_CONTROLLER

Any of the CONF Exit interfaces (for example):

- IF_FPM_OIF_CONF_EXIT
- IF_FPM_GAF_CONF_EXIT
- IF_FPM_TABBED_CONF_EXIT

To evaluate whether to release a component, the FPM completes the following steps:

1. FPM checks for the presence of the Global flag in the Global Settings dialog box in
   the FPM Configuration Editor. If it is not present, then it will treat all the UIBBs for
   this application as non-transient and hence will not release any components.
2. FPM reads the configuration global flag to see if the configuration is set to transient. If
   the configuration is non-transient, then this information is passed on to the UIBBs and
   FPM ignores the transient behavior; that is, it does not release the components.
3. FPM reads the configuration and sees that the global flag is set to transient. The
   following options are then available:
FPM checks the technical feasibility of the component being released and if it is not feasible, it retains the component.

If it is technically feasible to be released, it checks if the \texttt{IF\_FPM\_RESOURCE\_MANAGER} interface has been implemented by the component. If the interface is implemented, it calls the \texttt{on\_Hide} method else releases the component.

In the \texttt{on\_Hide} method, it checks for the veto value from the UIBB. If the UIBB has not vetoed the release state, then FPM releases it.

Otherwise, it will retain it.

### Settings for Transient Behaviour

Depending on how you want your application to use transient behavior, you can make the settings described in the following table.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Transient Flag</th>
<th>Implement \texttt{IF_FPM_RESOURCE_MANAGER}</th>
<th>Veto</th>
<th>Coding with the resource manager to handle application data</th>
</tr>
</thead>
<tbody>
<tr>
<td>I do not want transient behavior for any UIBB and I do not want to release any memory.</td>
<td>False</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>I do not want transient behavior for any UIBB but I would like to release some resources.</td>
<td>False</td>
<td>Yes</td>
<td>NA</td>
<td>Yes Based on need from business logic.</td>
</tr>
<tr>
<td>I want all my UIBBs to be transient. I do not have the need to release any resources explicitly.</td>
<td>True</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>I want all my UIBBs to be transient. I want to release some resources explicitly.</td>
<td>True</td>
<td>Yes</td>
<td>No</td>
<td>Yes Based on need from business logic.</td>
</tr>
<tr>
<td>I want only some of the UIBBs to be released. I would like to retain some due to business logic reasons.</td>
<td>True</td>
<td>Yes</td>
<td>Yes (only for those that do not need to be released).</td>
<td>Based on need from business logic.</td>
</tr>
<tr>
<td>I only want some of the UIBBs to be released. Some UIBBs I would like to retain due to business logic reasons. For those UIBBs that are transient, there is no need to release any resources.</td>
<td>True</td>
<td>Only on those that need to veto (or not be released).</td>
<td>Yes (only for those that do not need to be released).</td>
<td>NA</td>
</tr>
</tbody>
</table>
Setting the Transient Flag

Procedure

1. Start the FPM Configuration Editor for your application and go to the component configuration screen.
2. Choose Change and select Global settings.
3. The Global Settings dialog box contains the field for the transient setting.
4. Use the F4 help for Transient State. For the transient behavior, choose T.
5. Save the configuration.

Result

All the UIBBs in the application are now transient.

Using IF_FPM_RESOURCE_MANAGER to Veto Release Decision

Procedure

1. Open transaction SE80 and open the Web Dynpro component of your application.
2. Add IF_FPM_RESOURCE_MANAGER to Implemented Interfaces tab.
3. In the Component Controller, on the Methods tab, the ON_HIDE method is visible.

The ON_HIDE method has an importing parameter called IV_RELEASE_COMPONENT, which provides information to the UIBB about the FPM’s decision on the release feasibility for the component containing this UIBB. The UIBB reacts to this parameter only if the value is true. If the UIBB does not want itself to be released, then it sets the exporting parameter EV_VETO_RELEASE to true (the default is false).

FPM will use the veto parameters only if the IV_RELEASE_COMPONENT is true. If the UIBB sets the veto to true, then the component containing the UIBB is not released, even if it is capable of being released.

The sample code below demonstrates this:

```abap
method ON_HIDE.
  data: lv_veto type bool_d.
  IF IV_RELEASE_COMPONENT = abap_true.
    "do some business logic here and based on it, set the flag
    lv_veto = abap_true.
    ENDIF.
  IF lv_veto = abap_true." some bus.
    EV_VETO_RELEASE = abap_true. "This UIBB will not be released.
    ENDIF.
endmethod.
```

The following table is helpful in understanding the final action taken by FPM.
Using an FPM Application Controller

Sometimes it is necessary for the application to participate in all FPM events that happen during the entire lifetime of the application, with one arbitrary single component instance. This might be necessary for controlling and steering the application as a whole.

This is not possible, for example, with simple UIBBs since the methods provided by the Web Dynpro interface `IF_FPM_UI_BUILDING_BLOCK` only participate in the FPM event loop when the corresponding UIBBs are visible at the time the event loop happens or become visible after the current event loop has finished successfully. Furthermore the UIBBs cannot make assumptions about the sequence in which they are called. Therefore, an application controller is provided that closes this gap and provides the possibility to control and steer the application as a whole.

Implementing the Application Controller

The application controller is a singleton instance of a Web Dynpro component provided by the application. In order to use a Web Dynpro component as an application controller, complete the following steps:

1. Choose a Web Dynpro component and implement the Web Dynpro interface `IF_FPM_APP_CONTROLLER`.
2. Insert the component you have chosen into the OIF or GAF component configuration.

To do this, open the component configuration with the FPM Configuration Editor. Choose Display and choose Global Settings. In the dialog box, enter the component.

Regarding the behavior of instantiating the Web Dynpro components and their participation within the FPM event loop, the Web Dynpro interfaces provided by the FPM can be divided into two categories:

![Note]

When using the interfaces `IF_FPM_APP_CONTROLLER` and `IF_FPM_OIF_CONF_EXIT` (or `IF_FPM_GAF_CONF_EXIT`) together, they must be implemented by the same Web Dynpro component. Furthermore, it is recommended to implement the Web Dynpro interface `IF_FPM_SHARED_DATA` also in that Web Dynpro component (but only if this Web Dynpro interface is needed).
IF_FPM_APP_CONTROLLER Interface

This Web Dynpro interface provides you with methods to allow the application to participate in all FPM events that happen during the entire lifetime of the application.

Methods

This interface contains similar methods to the Web Dynpro interface IF_FPM_UI_BUILDING_BLOCK.

The interface IF_FPM_APP_CONTROLLER has two corresponding methods with the prefix BEFORE_ and AFTER_ for each of the IF_FPM_UI_BUILDING_BLOCK methods, for example, BEFORE_PROCESS_EVENT and AFTER_PROCESS_EVENT.

As the names suggest, the method BEFORE_PROCESS_EVENT is called immediately before another call to the corresponding UIBB method PROCESS_EVENT; the AFTER_PROCESS_EVENT is called immediately after all calls to PROCESS_EVENT are finished.

Using an Application-Specific Configuration Controller

Using an application-specific configuration controller (AppCC) allows you to do the following:

- Make global checks (checks affecting more than one UIBB)
- Make global adjustments for FPM events
- Read the structure of your application at runtime
- Change the structure of your application dynamically

This is the place where all actions affecting more than one single UIBB can be performed. Using an AppCC is optional; implement an AppCC only if you need one of the features which the AppCC offers.

Implementing an AppCC Component

To provide your application with an AppCC, you implement one of the following Web Dynpro interfaces in a Web Dynpro component:

- IF_FPM_OIF_CONF_EXIT for an OIF application
- IF_FPM_GAF_CONF_EXIT for a GAF application

This Web Dynpro component is either one of the components already used within your application or is a completely new one. To declare the AppCC component to FPM, proceed as follows:
1. Start the FPM Configuration Editor for your application component and open the Component Configuration screen.
2. In the control region, choose Change → Global Settings.
3. In the Global Settings dialog box, enter the Web Dynpro Component and the Configuration Name.
4. Choose Save.

If your AppCC has declared a static usage to a component implementing IF_FPM_SHARED_DATA, this shared data component is instantiated and attached automatically by the FPM framework. This ensures that all components within your application, which access the shared data component, will see the same instance of it.

**Methods**

The AppCC interface contains only one method for each floorplan application:

- OVERRIDE_EVENT_OIF
- OVERRIDE_EVENT_GAF

These methods pass an object of type IF_FPM_OIF (or IF_FPM_GAF), which serves as an API for the applications. The OVERRIDE_EVENT_OIF (or OVERRIDE_EVENT_GAF) method is called at the start of event processing on all visible UIBBs immediately after the FLUSH method has been called.

**Features**

The AppCC application programming interface provides you with the following features:

- Cancelling events

  With the AppCC you can perform global checks which apply to more than one UIBB. For checking purposes, the event is stored as an attribute in the IF_FPM_OIF (respectively IF_FPM_GAF) interface of the AppCC. You can cancel an event out of the AppCC by calling the CANCEL_EVENT method of the AppCC.

- Selecting a variant

  If there is more than one variant configured, you can select a specific variant to be used in an event by calling SET_VARIANT method in the IF_FPM_OIF respectively IF_FPM_GAF interface.

- Adjusting events

  The IF_FPM_OIF respectively the IF_FPM_GAF interface provides the currently processed FPM event as a changeable attribute. Therefore, it is possible to change an event by adding, removing, or changing event parameters. You also can replace an event.
As the AppCC is called right at the beginning of the event loop, changing an event has the same result as if changed event had been raised instead of the original event.

- Reading the configuration at runtime

The AppCC provides you with several methods which allow you to read the configuration data at runtime. The following table gives you an overview of all methods available for all types of floorplans.

<table>
<thead>
<tr>
<th>Method</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_CURRENT_STATE</td>
<td>Returns the current navigation state within the application.</td>
</tr>
<tr>
<td>GET_VARIANTS</td>
<td>Returns a list of all available variants.</td>
</tr>
<tr>
<td>GET_UIBB_KEYS</td>
<td>Returns a list of all UIBB assigned to a specified main step, substep, main view or subview.</td>
</tr>
<tr>
<td>GET_UIBB_KEYS_FOR_CONF_STEP</td>
<td>Returns a list of all UIBB assigned to a confirmation screen.</td>
</tr>
<tr>
<td>GET_UIBB_KEYS_FOR_INIT_SCREEN</td>
<td>Returns a list of all UIBB assigned to an initial screen.</td>
</tr>
</tbody>
</table>

The following table gives you an overview of all methods available for an OIF application.

<table>
<thead>
<tr>
<th>Method</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_MAINVIEWS</td>
<td>Returns a list of all main views for a given variant.</td>
</tr>
<tr>
<td>GET_SUBVIEWS</td>
<td>Returns a list of all subviews for a given main view.</td>
</tr>
</tbody>
</table>

The following table gives you an overview of all methods available for a GAF application.

<table>
<thead>
<tr>
<th>Method</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_MAINSTEPS</td>
<td>Returns a list of all main steps for a given variant.</td>
</tr>
<tr>
<td>GET_SUBSTEP_VARIANTS</td>
<td>Returns a list of all substep variants for a given main step.</td>
</tr>
<tr>
<td>GET_SUBSTEPS</td>
<td>Returns a list of all substeps for a given substep variant.</td>
</tr>
<tr>
<td>GET_HIDDEN_MAINSTEPS</td>
<td>Returns a list of all hidden main steps for a given variant.</td>
</tr>
</tbody>
</table>

- Changing the configuration at runtime

The AppCC provides you with several methods if you want to change the configuration data at runtime. The following table gives you an overview of all methods available for all types of floorplans.
### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_UIBB</td>
<td>Adds dynamically another UIBB to a main view, subview, main step, or substep.</td>
</tr>
<tr>
<td>REMOVE_UIBB</td>
<td>Removes dynamically another UIBB to a main view, subview, main step, or substep.</td>
</tr>
</tbody>
</table>

The following table gives you an overview of all methods available for an OIF application.

<table>
<thead>
<tr>
<th>Method</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_MAINVIEW</td>
<td>Adds dynamically another main view at runtime.</td>
</tr>
<tr>
<td>REMOVE_MAINVIEW</td>
<td>Deletes dynamically a given main view at runtime.</td>
</tr>
<tr>
<td>ADD_SUBVIEW</td>
<td>Adds dynamically another subview at runtime.</td>
</tr>
<tr>
<td>REMOVE_SUBVIEW</td>
<td>Deletes dynamically a given subview at runtime.</td>
</tr>
<tr>
<td>RENAME_MAINVIEW</td>
<td>Renames dynamically a given main view at runtime.</td>
</tr>
<tr>
<td>RENAME_SUBVIEW</td>
<td>Renames dynamically a given subview at runtime.</td>
</tr>
<tr>
<td>SET_SELECTED_SUBVIEW</td>
<td>Changes the target subview within a given main view.</td>
</tr>
<tr>
<td></td>
<td>This method must only be used in order to enforce a given main view to switch to</td>
</tr>
<tr>
<td></td>
<td>the provided subview instead of the default subview.</td>
</tr>
</tbody>
</table>

The following table gives you an overview of all methods available for a GAF application.

<table>
<thead>
<tr>
<th>Method</th>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENAME_MAINSTEP</td>
<td>Renames dynamically a given main step at runtime.</td>
</tr>
<tr>
<td>RENAME_SUBSTEP</td>
<td>Renames dynamically a given substep at runtime.</td>
</tr>
<tr>
<td>ENABLE_MAINSTEP</td>
<td>Enables or disables a given main step at runtime.</td>
</tr>
<tr>
<td>HIDE_MAINSTEP</td>
<td>Hides a given main step within the roadmap.</td>
</tr>
<tr>
<td></td>
<td>The affected main step will not be visible as a main step in the roadmap anymore.</td>
</tr>
<tr>
<td></td>
<td>Nevertheless, the hidden main steps continue to be processed in the background in</td>
</tr>
<tr>
<td></td>
<td>order to keep the business logic untouched.</td>
</tr>
</tbody>
</table>

### Implementing an AppCC Class

You can use a simple ABAP OO class as an AppCC. Enter the name of an ABAP OO class, implementing the ABAP OO interface `IF_FPM_<floorplan>_CONF_EXIT`, in the WD component name in the Global Settings dialog box (if a WD component with the same name already exists, the WD component is used).
The ABAP OO AppCC offers the same methods and options as the WD AppCC and works in the same way. The only difference is that the AppCC itself cannot be configured.

Sharing Data between UIBBs from Different Components

When the UIBBs of an application are implemented in several components, there is often the need to share data between these components. Technically, there are several approaches which you can take to achieve this. This is described in the following chapters.

For this purpose, the FPM offers Shared Data components.

This is an optional FPM feature which meets most applications’ demands. However, if needed, it can be replaced by other technical alternatives as described in Other Options for Sharing Data.

Using a Shared Data Component

A shared data component is a Web Dynpro component which implements the IF_FPM_SHARED_DATA interface. This interface contains no methods or attributes but serves as a marker interface only. Each component (for example UIBB, FPM_OIF|GAF_CONF_EXIT component) which wants to use a shared data component needs to declare a usage to the shared data component. For this, the technical type of the usage does not need to refer to IF_FPM_SHARED_DATA (this would mean that it would not have accessible methods/attributes) but link to the actual component itself. The lifecycle handling is now handled completely by the FPM. Whenever a component is instantiated by the FPM (for example a UIBB which is configured for a given screen), the FPM analyzes all usages of that component. If it detects a usage pointing to a component which implements the IF_FPM_SHARED_DATA interface, a singleton of this shared data component is automatically attached to the usage.

As a result, an application must proceed as follows to share data using the shared data interface:

- Create a component which implements the IF_FPM_SHARED_DATA interface.
  This component contains methods to retrieve data from the business logic and exposes the extracted UI data via its Web Dynpro context or interface methods.

- Each component accessing this shared data defines a usage of the shared data component. This usage is automatically instantiated by the FPM.
  The consuming component can now communicate with the shared data component via Web Dynpro context mapping, attribute access or method calls.
Other Options for Sharing Data

There are other options to share data between Web Dynpro components besides the FPM shared data concept. There are occasions when it is best not to use a Shared Data component, as detailed below:

- There is already an application-specific API available which serves as a ‘data container’ and can be accessed by several components.
- The data needs to be shared not only between Web Dynpro components but also between other entities, such as ABAP OO classes, function groups, etc.
- The amount of data to be shared is so large that putting it into a Web Dynpro context would result in performance and memory consumption issues.

In these cases, the application can consider using techniques such as the following:

- An ABAP OO class which is accessible as a singleton, so that all consumers share the same instance.
- A function group with appropriate function modules.

Determining Navigation State Information at Runtime

For some use cases it is necessary to determine the current navigation state of the application at runtime. To support this, the IF_FPM interface provides the method GET_RUNTIME_INFO. This method allows for example to determine which subview is currently selected in OIF or which substep will be displayed after the event in GAF.

There are always two states which can be determined: The state before the current event and the target state after the event. As the target state might change during the event loop there are the following restrictions regarding the point in time when this API can be used.

The **current** State can be determined from **NEEDS_CONFIRMATION** to **PROCESS_BEFORE_OUTPUT** event

The **target** State from **NEEDS_CONFIRMATION** to **WDDOMODIFYVIEW**

If launched at the wrong point in time the API will launch a **CX_FPM_FLOORPLAN** Exception. This also means that at event FLUSH and in the AppCC the state info is not available (or in case of several events within one roundtrip it might be somehow outdated)

```plaintext
DATA: lo_fpm TYPE REF TO if_fpm,
     ls_fpm_info TYPE fpm_s_runtime_info.
lo_fpm = cl_fpm=>get_instance( ).
ls_fpm_info = lo_fpm->get_runtime_info( ).

* check whether it is a OIF or GAF floorplan
IF ls_fpm_info-floorplan =
   if_fpm_constants=>gc_floorplan-oif.
   DATA: lo_oif_info TYPE REF TO if_fpm_oif_info,
```
ls_current_state TYPE fpm_s_oif_info,
ls_target_state TYPE fpm_s_oif_info.

* now here comes the cast from type object to type
   if_fpm_oif_info.
   lo_oif_info ?= ls_fpm_info-floorplan_info.
   ls_current_state = lo_oif_info->get_current_state( ).
   ls_target_state = lo_oif_info->get_target_state( ).
ENDIF.

* now same thing for GAF...
IF ls_fpm_info-floorplan =
   if_fpm_constants=>gc_floorplan-gaf.

DATA: lo_gaf_info TYPE REF TO if_fpm_gaf_info,
   ls_current_state TYPE fpm_s_gaf_info,
   ls_target_state TYPE fpm_s_gaf_info.

* now here comes the cast from type object to type if_fpm_gaf_info.
   lo_gaf_info ?= ls_fpm_info-floorplan_info.
   ls_current_state = lo_gaf_info->get_current_state( ).
   ls_target_state = lo_gaf_info->get_target_state( ).
ENDIF.

Embedding an FPM Application

FPM was designed for building standalone applications. However, it is possible (with some restrictions) to embed an FPM application within another Web Dynpro application.

To do this, proceed as follows:

1. Create a usage for the component FPM_OIF_COMPONENT for OIF applications (or FPM_GAF_COMPONENT for GAF applications) within the embedding component.
2. Embed the FPM_WINDOW Interface View within one of the views of the embedding component.
3. Manually create the FPM component to be used (as you must provide the configuration key of the floorplan component). This is best done as soon as possible.

In most cases, this is the Web Dynpro DOINIT method of the embedding application’s component controller, as the sample code below shows:

method Web DynproDOINIT .
data: lo_usage type ref to if_web Dynpro_component_usage,
   ls_conf_key type web Dynpro_config_key.
lo_usage = Web Dynpro_this->web Dynpro_cpuse_fpm_usage( ).
if lo_usage->has_active_component( ) = abap_true.
   lo_usage->delete_component( ).
endif.
ls_conf_key-config_id = “ID configuration of FPM component”.
* recreate component using new configuration ID try.
call method lo_usage->create_component
  EXPORTING
  component_name   = 'FPM_OIF_COMPONENT'
  configuration_id = ls_conf_key.
  catch cx_Web Dynpro_runtime_api.
endtry.

The following remarks relate to the above sample code:

- The configuration you pass is the configuration key of component  
  FPM_OIF_COMPONENT. You cannot pass the application’s configuration key.
- The code example names the usage FPM_USAGE. If you name it differently, adjust the  
  following line: lo_usage = Web Dynpro_this->Web Dynpro_cpuse_fpm_usage(  
  ).
- The example is for an OIF application; for a GAF application, replace  
  FPM_OIF_COMPONENT by FPM_GAF_COMPONENT.
- The delete_component( ) call is not necessary for simple static embedding. However,  
  you need it if you want to change the embedded FPM application in the future.

Constraints

- FPM allows only one instance running at the same time within one internal mode.  
  Therefore, you cannot embed more than one FPM application at the same time. It is  
  possible to switch the embedded FPM application, replacing one FPM application by  
  another. You can assure this if you only use one Usage to an FPM component within  
  your application. This forces you to delete the old FPM component before creating a  
  new one.
- You cannot embed an FPM application within another FPM application.
- You cannot pass a configuration key for the IDR (header area). Therefore, the header  
  appears without configuration settings; these you can set programmatically at runtime.
- You cannot pass application parameters for the FPM application, as the application is  
  now unknown to FPM.

Embedded Analytics in Floorplan Manager

You can use Embedded Analytics to embed analytical and planning content into  
transactional applications in Floorplan Manager without development effort.

Embedded Analytics is part of the Business Suite Foundation layer and, as such, is not  
available to all applications.

This section describes the steps that are required to configure Embedded Analytics in  
addition to the standard Floorplan Manager configuration.
You can use the standard application *List Report on BEx Query* (WDA_BS_ANLY_LIST) or build your own applications for list reporting and planning purposes.

**List Report on BEx Query Application**

You can use the application *List Report on BEx Query* (WDA_BS_ANLY_LIST) with application configuration APP_CFG_BS_ANLY_LIST for list reporting purposes.

**Features**

- The generic application addresses a business scenario in which reported data can be displayed as a list of single records.
- The application can display any BEx query. Hierarchies, however, are not supported.
- You can reduce the amount of searched data by specifying selection criteria on a data selection screen.
- You can save the set of specified selection criteria as a search variant for later use.
- The analytics list component for Floorplan Manager is used to display the selected data.
- You can activate specific analytics functions using the analytics list editor for Floorplan Manager in addition to the standard SAP List Viewer (ALV) functions.
- You can use the application for simple business planning by entering planning data in the input-enabled BEx query.

**Creating Analytical and Planning Applications**

You can build your own applications using the analytics list component (WDC_BS_ANLY_LIST_ALV) and the search component with analytics feeder class in any Floorplan Manager application.

Analytical applications consist of the following:

- Analytics list component
  Displays the result list of the BEx query
- Search component based on analytics feeder class
  Represents the BW variable screen
- Planning functions or planning sequences
  Triggered by FPM events
- Application controller
  Central module that controls the interaction between the different components and events

**Examples**
You can build analytical and planning applications, from simple to complex. Note that the list below is not exhaustive.

- Application with one variable screen and several analytics list components to display several queries

In this scenario, it is important that the search component be configured with the list of all the queries that are displayed in the list components. The variable screen merges the variables of all queries. In other words, variables that are used in more than one are displayed only once.

- Simple planning application

Consists of an input-enabled analytics list component, one or more buttons for planning functions and sequences, and a search component that displays the variables of the queries and planning function and /sequences. It is important that the search component be configured with the query and the list of planning functions, filters, and planning sequences. You can then, for example, enter the revaluation factor for a revaluation function on the search screen.

- Advanced planning application

Consists of one or more input-enabled queries for planning and one or more read-only queries to control the planning result or make plan-actual comparisons and buttons for planning functions/sequences. In this scenario, you can have multiple search components, one for the selection variables of the queries and one for the planning function variables. This allows you to distinguish between variables that are required centrally when starting the applications and variables that are only required when a planning function or sequence is executed.

**Procedure**

**Substep A: Creating a Web Dynpro Application**

1. Run transaction *Object Navigator (SE80)* to access the ABAP Development Workbench.
2. Select a package if you want to transport the application. If not, select *Local Objects* from the dropdown list.
3. Right-click the root node and choose *Create Web Dynpro → Web Dynpro Application*.
4. In the dialog box, enter the name and description of the application, and choose *Continue*.
5. Enter the following information to create, for example, an Overview Page (OVP) application:

   - Component
     
     FPM_OVP_COMPONENT

   - Interface View

     FPM_WINDOW
• Plug Name

Default

Note:

This procedure applies to OVP applications. You can also use the Object Instance floorplan (FPM_OIF_COMPONENT) or the Guided Activity floorplan (FPM_GAF_COMPONENT), and adapt them accordingly.

6. Save your entries. In the Create Object Directory Entry dialog box, enter a package name and choose OK.

Substep B: Creating a Web Dynpro Application Configuration and the Corresponding Component Configuration

1. In the Object Navigator, right-click the new Web Dynpro application and choose Create/Change Configuration. The configuration editor for Floorplan Manager opens in a browser window.

2. Enter the name of your configuration application in the Configuration ID field. Note that configuration names are global. In other words, you cannot use the same configuration name for different applications.

3. Choose Create. In the Create Configuration dialog box, enter the relevant package and choose OK.

4. The application configuration window displays your new configuration.

5. Enter the name of the component configuration and choose Save.

6. Choose Go To → Component Configuration.

7. Choose Create to configure the component.

8. In the Create Configuration dialog box, enter the relevant package and choose OK.

9. Choose Save.

The component configuration is now open for editing.

The configuration editor for Floorplan Manager displays the Component Configuration window for your OVP component. The FPM Component Configuration window is split into the following areas:

• Navigation hierarchy

Displays the screen elements that you can configure in your application.

• Preview

Displays the element you have selected in the hierarchy and allows you to change the attributes of the element.

• Action Area
Allows you to add various elements to your individual screens, for example, toolbar buttons and UIBBs.

A simple application requires only one page and one section. The Configuration Editor for Floorplan Manager automatically provides you with these entities with default IDs and names.

**Substep C: Editing the Component Configuration**

In the component configuration, you can do the following:

- Add an analytics list component to a section.
- Add a search component based on an analytics feeder class to a section.
- Use FPM events that trigger planning functions or planning sequences.
- Configure an application controller for analytical and planning applications.

**Analytics List Component**

This component displays data based on a BEx query in the SAP List Viewer (ALV). In the case of an input-enabled BEx query, the analytics list component is input-enabled.

**Search Component Based on Analytics Feeder Class**

This component allows you to enter BW variables for queries, filter, planning functions, and planning sequences.

**FPM Events Triggering Planning Functions or Planning Sequences**

You can use predefined events if you want to execute planning functions and planning sequences. The FPM event for executing a planning function is BSA_PLFUNC with the parameters PLFUNC and FILTER for the name of the planning function and the filter. The FPM event for executing a planning sequence is BSA_PLSEQU with parameter PLSEQU for the planning sequence.

You can use these events in toolbar elements such as buttons or button choices. You can use these toolbar elements in toolbars of a content area by choosing *Add Toolbar Element* in the *Action* area, or in the configuration of the analytics list component.

**Application Controller**

You must use an application controller (Web Dynpro component **WDC_BS_ANLY_APPCONTROLLER**) that handles the following steps in a generic way for any analytical and planning application:

- Writing back and checking data that was manually entered in the planning-enabled query
- Writing back and checking variable values that were entered
- Executing planning functions and sequences defined by means of an FPM event
To edit the application controller, start the configuration editor, choose Change Global Settings, enter WDC_BS_ANLY_APPCONTROLLER in the Web Dynpro Component field, and save your changes.

You can use several analytics list components and search components in an application. All components are optional, except for the application controller.

**Embedding Analytics and Planning into Transactional Applications**

You can embed analytical components into transactional FPM applications by performing the steps described in Creating Analytical and Planning Applications. However, you can also create your own application controller to meet your requirements instead of using the generic application controller WDC_BS_ANLY_APPCONTROLLER.

**Analytics List Component for Floorplan Manager**

The analytics list component uses the SAP List Viewer (ALV) to display the data of a BEx query in tabular form, in graphical form, or as Crystal Reports. You can sort and filter data, define your own view, and export lists to Microsoft Excel. If you choose Crystal Reports to display the data, you can change the formatting to meet your requirements. You can, for example, use your own template instead of the standard generic template. Hierarchies, however, cannot be displayed. If a BEx query contains hierarchies, the hierarchies are automatically deactivated.
Adding an Analytics List

Procedure

Substep A: Calling Up the Configuration Editor for Floorplan Manager

1. Start the configuration editor for Floorplan Manager to configure a Web Dynpro component.

Substep B: Adding an Analytics List Component

1. In the hierarchy, select the section where you would like to add a new analytics list component.
2. Choose Add UIBB → Add Analytics List Component in the action area.

The system has entered the analytics list component as a new UIBB with component WDC_BS_ANLY_LIST_ALV and view W_MAIN.

Note

Although the Configuration Name field is not marked as required, you must specify a configuration name. The configuration type, configuration variant, and sequence order are optional.

3. Choose Save.

Substep C: Configuring the Analytics List Component

1. Choose Configure UIBB.
2. In the Editor for Web Dynpro ABAP Components - Configuration screen, choose Create.
3. In the Create Configuration dialog window, enter a description and choose a package.
4. Choose OK.

Result

The system opens the analytics list editor for Floorplan Manager for the new analytics list.
Analytics List Editor for Floorplan Manager

You use the configuration editor to change the analytics list component in your application according to your business requirements.

Features

The configuration editor consists of the following work areas:

- Input field for query name
- SAP List Viewer settings
- SAP List Viewer toolbar elements

List Settings

You can define the parameters listed below in the List Settings area. These parameters control the behavior of the SAP List Viewer and of the related user settings dialog at runtime.

- Title Visible

If this checkbox is selected, the name of the query is displayed. By default, this checkbox is selected.

- Title

Title of the list. By default, the query name is retrieved. You can overwrite it with your own text.

- Filter Allowed

If this checkbox is selected, the Filter button is displayed. By default, this checkbox is selected.

- Settings Dialog Visible
- Column Selection Allowed

If this checkbox is selected, the Column Selection tab is displayed in the Settings dialog. By default, this checkbox is selected.

- Calculation Allowed

If this checkbox is selected, the Calculation tab is displayed in the Settings dialog. By default, this checkbox is selected.

- Saving Views Allowed
If this checkbox is selected, you can save custom settings pertaining to columns for later use, for example, the number of columns and the order. By default, this checkbox is selected.

- **Sorting Allowed**

If this checkbox is selected, the Sorting tab is displayed in the Settings dialog. By default, this checkbox is selected.

- **Changing Display Type Allowed**

If this checkbox is selected, the Display Type dropdown box is displayed in the toolbar (table, graphic, table and graphic, and Crystal Reports), as well as the graphic settings of the Display tab in the Settings dialog. By default, this checkbox is deselected.

- **Export Allowed**

If this checkbox is selected, the Export button is displayed in the toolbar. This allows you to export data to Microsoft Excel. By default, this checkbox is deselected.

- **Printing Allowed**

If this checkbox is selected, the Print button is displayed in the toolbar. By default, this checkbox is deselected.

- **Goto Allowed**

If this checkbox is selected, the Goto button choice is visible as a toolbar element in the header of the output table. If you have defined any targets for a specific BEx query in transaction Maintain Sender/Receiver Assignment (RSBBS), the Goto button choice is active and provides you with a list of possible targets.

If the target is a BEx query, you can select one of the following runtime environments:

- BEx Web Analyzer (ABAP)
- BEx Web Analyzer (Java)
- List Report (WD ABAP)
- Number of Rows

Contains the maximum number of allowed rows. By default, ten rows are allowed.

- **Display Columns**

Displays the number of columns. By default, all columns are displayed. If you choose 'Other Number', eight columns are displayed by default.

- **Number of Fixed Columns**
Contains the number of columns at the left hand side of the screen that are always displayed when you scroll horizontally. By default, zero columns are displayed.

**Toolbar Elements**

You can define the following parameters in the Toolbar Elements area and assign FPM events to the elements:

- **Button**

  For each button you add, you must specify a label and an FPM event ID.

- **Button Choice**

  For each button choice, you must add items and specify a label for the button choice, as well as a label and an FPM event ID for each item.

- **Separator**

  You can insert additional vertical space between toolbar elements and group elements that belong together by means of separators.

- **Link to Action**

  For each added link to action, you must specify a label and an FPM event ID.

**Note**

You can define additional parameters for all toolbar elements, except for separators.

You can use FPM events and event parameters to execute planning functions and sequences.

**Note**

Tooltips are not mandatory for the above user interface (UI) elements. However, you can use them to convey detailed information.

**Search Component with Analytics Feeder Class**

The search component with the analytics feeder class allows you to display the BW variables of one or more queries, filters, planning functions, and planning sequences.

You can enter variable values to restrict the query selection or as planning parameters. Additionally, you can save the set of variable values you have specified as a search variant for future use.

The following restrictions apply:
• The display of hierarchy variables and hierarchy node variables with manual input on the selection screen is not supported. If a query contains a hierarchy variable or a hierarchy node variable, the user is informed about the restricted selection that cannot be displayed. If you have defined a default value for a variable, the query is executed and the default value is taken into account. In the case of a mandatory variable without a default value, the query is not executed and an error message is issued.

• In SAP NetWeaver Business Warehouse (SAP NetWeaver BW), you can select the 'Not assigned' value by entering a number sign ('#'). In some cases, this is not allowed in the analytics search component:
  - For variables based on characteristics with the data types NUMC (Character String with Only Digits), DATS (Date) or TIMS (Time), you can enter a zero instead of a number sign to select all variables.
  - For variables based on characteristics with special conversion routines, for example, 0CALMONTH with conversion routine PERI6, you cannot enter a number sign.

Adding an Analytics Search Component

Procedure

Substep A: Calling Up the Configuration Editor for Floorplan Manager

1. Start the configuration editor for Floorplan Manager to configure a Web Dynpro component.

Substep B: Adding a Search Component

1. In the preview, select the location where you would like to add a new Analytics List Component.
2. Choose Add UIBB → Add Search Component in the action area.
3. Choose Save.

Result

The system has entered the search component for analytics as a new UIBB.

Configuring an Analytics Search Component

Procedure

Configuring a Search Component

1. Choose Configure UIBB.
2. On the Editor for Web Dynpro ABAP Components - Configuration screen, choose Create.
3. In the Create Configuration dialog window, enter a description and select a package.
4. The system opens the Configuration Editor for the Search Component.

Configuring a Search Component for Analytics
1. The Edit Feeder dialog box is displayed. Enter the feeder class 
   `CL_BS_ANLY_LIST_SEARCH_FEEDER` and choose Edit Parameters.
2. The Edit Parameters dialog box is displayed. You can enter following parameters:
   - Queries
   - Filters (only relevant for planning applications)
   - Planning functions (only relevant for planning applications)
   - Planning sequences (only relevant for planning applications)
   The input help for these parameters allows you to add multiple values for each 
   parameter. The values are separated by semicolons.
3. Choose OK.
4. Choose Save.

**Application-Specific Analytics UIBBs**

You can also develop application-specific analytical UIBBs for special purposes by 
implementing the `IF_FPM_UI_BUILDING_BLOCK` Web Dynpro interface.

To access analytical data in these UIBBs, you must use our analytical Application 
Programming Interface (API). With this API, you can read data, set variable values, write 
back data, and execute planning functions.

**Analytical Application Programming Interface (API)**

The Analytical Application Programming Interface (API) consists of the interface and 
classes in the package interface `BS_ANLY_LIST_REPORTING`.

The class List Reporting: Services (`CL_BS_ANLY_LIST_SERVICES`) provides methods for 
the instantiation of needed objects: data instance, filter instance, planning function 
instance, planning sequence instance, RRI instance, and selection instance.

For the instantiation, the following methods are used:

- `GET_DATA_INSTANCE`
  Retrieves an instance of a data object based on a BEx query and returns a reference to 
  `IF_BS_ANLY_LIST_DATA`.
- `GET_FILTER_INSTANCE`
  Retrieves an instance that is required to execute a planning function and returns a 
  reference to `IF_BS_ANLY_LIST_FILTER`.
- `GET_PLFUNC_INSTANCE`
  Retrieves an instance of a planning function and returns a reference to 
  `IF_BS_ANLY_PLFUNC`.
- `GET_PLSEQU_INSTANCE`
  Retrieves an instance of a planning sequence and returns a reference to 
  `IF_BS_ANLY_PLSEQU`.
- `GET_SELECTION_INSTANCE`
  The method is called with import parameters list of queries, filters, planning functions, 
  and sequences. The method retrieves an instance of a set of BW variables and returns a
reference to IF_BS_ANLY_SELECTION. The variables of the used BW queries, filters, planning functions, and sequences are merged so that no variable occurs twice in the selection instance.

- **GET_RRI_INSTANCE**
  Retrieves an instance for a Report-Report-Interface and returns a reference to IF_BS_ANLY_LIST_RRI.

The class provides method SAVE to save application data.

The following methods are required for read-only scenarios:

- **IF_BS_ANLY_LIST_DATA->READ**
  Reads data of a BEx query
- **IF_BS_ANLY_SELECTION->GET_VARIABLES**
  Reads metadata of variables
- **IF_BS_ANLY_SELECTION->READ**
  Reads variable values
- **IF_BS_ANLY_SELECTION->WRITE**
  Writes back changed variable values
- **IF_BS_ANLY_SELECTION->CHECK**:
  Checks entered variable values. Note that CHECK has to be called after WRITE, otherwise the changed variable values are not required.
- **IF_BS_ANLY_LIST_RRI->GET_RECEIVERS**:
  Retrieves the Report-Report interface targets.
- **IF_BS_ANLY_LIST_RRI->CALL_RECEIVER**:
  Launches a target.

For planning scenarios, you need additional methods:

- **IF_BS_ANLY_LIST_DATA->WRITE_CELLS**
  Writes back changed data cells if the query is input-enabled.
- **IF_BS_ANLY_LIST_DATA->CHECK**
  Checks write-back cells and sends the changes to the planning buffer. Note that CHECK has to be called after WRITE_CELLS, otherwise the changed values are not visible.
- **IF_BS_ANLY_PLSEQU->EXECUTE**
  Executes a planning sequence.
- **IF_BS_ANLY_PLFUNC->EXECUTE**
  Executes a planning function. An instance of a filter object must be sent in this method. The filter determines the set of data the planning functions processes.

**FPM Event Loop for Analytics and Planning**

The following graphic describes the FPM event loop for Analytics and Planning:
The sequence describing how the used UIBBs are processed in the Flush and Process Before Output phases is not defined.

User actions trigger the FPM event loop.

- In the Flush phase:
  - The data entered on the user interface is written back to the model. This must be done separately for all GUIBBs, for example, for the Analytics List Component and the Search Component.
    - Analytics List Component
      - `IF_BS_ANLY_LIST_DATA->WRITE_CELLS( )` is called here.
    - Search Component
      - `IF_BS_ANLY_SELECTION->WRITE( )` is called here.

- In the Before Process Event phase:
  - The user input is checked and written back to the buffer.
  - The sequence describing how the used UIBBs are processed by FPM is not defined. As a result, the ‘Process Event’ phase of the single GUIBBs cannot be
used here. Instead, the central application controller WDC_BS_ANLY_APPCONTROLLER is used. The following processing sequence must be defined in the application controller:
- First, entered plan data is submitted in the analytics list component.
  - `IF_BS_ANLY_LIST_DATA->CHECK( )` is called here.
- Then, variable values are submitted in the Search Component.
  - `IF_BS_ANLY_SELECTION->CHECK( )` is called here.
- Finally, depending on the type of user action, the following steps are also executed:
  - Execute Planning Function `IF_BS_ANLY_PLFUNC->EXECUTE( )`
  - Execute Planning Sequence `IF_BS_ANLY_PLSEQU->EXECUTE( )`
  - Save Data to database `CL_BS_ANLY_LIST_SERVICES=>SAVE( )`

- In the Process Before Output phase:
  - The new data is transferred to the user interface. This can be done separately for all GUIBBs.
  - Analytics List Component
    - `IF_BS_ANLY_LIST_DATA->READ( )` is called here.
  - Search Component
    - `IF_BS_ANLY_SELECTION->READ( )` is called here.