

Floorplan Manager for Web Dynpro ABAP - Developer's Guide -



**SAP NetWeaver 7.0
Enhancement Package 3**



Copyright

© Copyright 2011 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.
Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.






Java is a registered trademark of Sun Microsystems, Inc

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, xApps, xApp, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

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

Type Style	Description
<i>Example text</i>	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.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	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.
<code>Example text</code>	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.
<code>Example text</code>	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE TEXT	Keys on the keyboard, for example, F2 or ENTER.

Table of Contents

Copyright.....	2
Icons in Body Text	3
Typographic Conventions	3
Floorplan Manager.....	12
Getting Started.....	12
User Interface Building Blocks	13
IF_FPM_UI_BUILDING_BLOCK Interface	13
Creating a Simple FPM Application	15
Creating a Web Dynpro Component.....	15
Adding Views to your Web Dynpro Component	16
Creating a Web Dynpro Application	16
Using Application Parameters.....	17
Creating an Application Configuration with the FPM Configuration Editor	18
Configuring FPM_GAF_COMPONENT	18
Configuring FPM_IDR_COMPONENT	19
Testing your FPM Application	19
FPM Application Creation Tool	20
Starting the ACT.....	20
Creating a New Application using the ACT	20
FLUID (Flexible UI Designer).....	21
Launching FLUID in Different Modes	22
Structure and Layout of FLUID	22
Changing the Layout of FLUID	27
Working with FLUID	28
Adding an Existing UIBB to your Application.....	28
Changing the Title of a Step inside an Application based on the GAF Floorplan	28
Editing the Form Component inside an Application	29
Adding a New Button to the Toolbar in a Floorplan Component of an Application.....	29
Moving back to a Floorplan Component from a GUIBB Component.....	29
Limitations	30
Wire Model.....	30
IF_FPM_UIBB_MODEL Interface	30
IF_FPM_FEEDER_MODEL Interface.....	30
FPM on BOL	33
Creating a GUIBB on BOL	33
Creating an FPM Application on BOL	34
Design Time with the FPM Configuration Editor.....	35
Floorplan Instances in the FPM Configuration Editor	35
OIF Instance	36
GAF Instance	36
OVP Instance.....	37
Adding and Activating Sub-Steps for GAF Applications	37

FPM Toolbar	37
Differences between an OIF and a GAF Toolbar	38
Adding Elements to a Toolbar	38
Adjusting the Toolbar Dynamically	38
Toolbar Buttons	39
Toolbar Button Events	40
IF_FPM_CNR_GAF Interface	41
Accessing the API for a GAF application:	41
GAF Specific Parameters	42
IF_FPM_CNR_OIF Interface	44
Accessing the API for an OIF application:	44
OIF Specific Parameters	45
FPM Complete Preview	46
FPM Identification Region (IDR)	46
Adjusting the IDR Dynamically	47
Adding a Link to the FPM Configuration Editor in the IDR	47
IF_FPM_IDR Interface	47
Providing a Link to the FPM Configuration Editor in the IDR	48
Quick Help	49
Creating Quick Help	49
Procedure	49
Variants	50
Configuring Variant Selection	50
Initial Screen	51
Skipping the Initial Screen	51
Confirmation Screen	52
FPM Event Loop	53
Raising Standard Events	53
Triggering the FPM Event Loop	53
Triggering Application-Specific Events	54
Reacting to Framework Events	54
Key Web Dynpro Methods	55
Different Categories of Web Dynpro Interfaces	55
Overview Page Floorplan (OVP)	56
Structure of an OVP	56
Page	56
Section	57
UIBBs / GUIBBs	57
Stacking	58
Page Master	58
Personalization	60
Personalization Editor	61
Toolbars	63
External Navigation Menus	63
Default Actions	64
Edit / Display Mode	65

Processing Mode for Collapsed UIBBs ('Lazy Load')	66
Technical UIBBs.....	67
Initial Search Page & External Navigation	67
OVP-Related FPM Events for Navigation	71
Dynamic Changes at Runtime	73
OVP CNR API.....	74
Application Configuration Controller API.....	75
Setting a Default ALV View for a Freestyle UIBB	76
Design Time Settings in the FPM Configuration Editor	77
Rendering the ALV Views during Runtime.....	77
FPM Dialog Boxes	78
Structure.....	78
Features	79
Creating and Configuring an FPM Dialog Box.....	80
Triggering Dialog Boxes from a Toolbar Button.....	81
Opening and Closing FPM Dialog Boxes.....	81
Event Processing in Dialog Boxes	82
The MV_IS_DIALOG_MODE Attribute	82
Sample Coding to Call A Dialog Box	82
Opening a Dialog Box using Direct API	82
Opening a Dialog Box by Raising an FPM Event.....	82
Message Manager for FPM Dialog Boxes	83
Error Page of an FPM Dialog Box.....	84
Enabling/Disabling Dialog Box Buttons at Runtime	84
Sample Code to Set the Status of the Dialog Box button.	84
FAQs on FPM Dialog Boxes	84
Generic User Interface Building Block (GUIBB)	85
Feeder Classes	85
Structure	85
Features.....	86
Context Menus in GUIBBs	86
Methods of IF_FPM_GUIBB_CTXT_MENU Interface.....	86
Form Component (GUIBB FORM GL2)	88
Structure	88
IF_FPM_GUIBB_FORM Interface.....	89
Group Layout in a Form	93
Form Component (GUIBB FORM)	93
Structure	93
IF_FPM_GUIBB_FORM Interface.....	94
Using the CHECKBOX_GROUP Display Type in a Form	97
List ATS Component (GUIBB List ATS).....	98
Feeder Class.....	98
Configuration.....	100
Data Exchange	100
Actions	102

Features.....	104
Advanced Features.....	105
Changes to Elements from 'Old' List Component (GUIBB List).....	107
List Component (GUIBB LIST).....	108
Structure	108
IF_FPM_GUIBB_LIST_PAGING Interface.....	113
Additional Information on the List Component	115
FPM Events and the List Component	116
Rendering GUIBB List as ALV	116
Hierarchical List Component (GUIBB TREE).....	117
Structure	117
IF_FPM_GUIBB_TREE Interface.....	119
Additional Information on the Hierarchical List Component	124
FPM Events and the Hierarchical List Component.....	124
Search Component (GUIBB SEARCH)	125
Structure	126
Integration.....	127
IF_FPM_GUIBB_SEARCH Interface	127
Enter, Reset, and Clear Buttons	133
Result List.....	133
Exclude Criteria.....	135
Dependent Searches	135
Launchpad Component (GUIBB LAUNCHPAD).....	135
Structure	135
IF_FPM_GUIBB_LAUNCHPAD Interface	138
Tabbed Component (GUIBB TABBED COMPONENT).....	139
Structure	140
Changing the Tabbed Component Dynamically at Runtime	140
POWL Component (GUIBB POWL).....	141
Pre-requisites.....	141
The POWL Component in FPM	142
Configuring a POWL Component in FPM	142
The POWL Component at Runtime	145
Actions from Detail UIBB.....	146
Navigation to Error Page.....	146
Composite Component (GUIBB Composite)	147
Structure	147
Editing the Composite Component	147
Changing the Composite UIBB dynamically at Runtime	147
Analytical Components	148
Analytics List Component.....	149
Component Configuration	149
Tree Component with Analytics Feeder Class.....	151
Search Component with Analytics Feeder Class.....	153
Chart Component with Analytics Feeder Class	154
Structure	155

IF_BS_ANLY_GUIBB_CHART Interface	156
Chart Configuration for the Floorplan Manager	159
Chart Appearance	160
Chart Customizing File	161
Adding a Chart Component	161
FPM Events and the Chart Component	162
Application-Specific Analytics UIBBs	163
Analytical Application Programming Interface (API)	163
FPM Event Loop for Analytics and Planning	164
REUSE UIBB (RUIBB)	166
Attachment RUIBB	166
RUIBB Interface	166
Adding the Attachment RUIBB in FLUID	170
Attachment RUIBB Features	171
Notes RUIBB	176
RUIBB Interface	176
Adding the Notes RUIBB in FLUID	180
Notes RUIBB Features	181
Value/Input Helps for Generic UIBBs (GUIBBs)	184
Assignments in the Field Description	185
DDIC Value Help	185
OVS	185
Freestyle Value Help	185
Fixed Values	186
Drag-and-Dropping Data between UIBBs	186
Enabling Drag-and-Drop	186
Configuring Drag-and-Drop	187
Events and Event Parameters	188
Class, Methods and Parameters of Drag-and-Drop	188
Event Processing during <i>Drag-and-Drop</i>	190
Handling Drop in UIBBs	190
Dynamically Changing Drag-and-Drop	190
Context Based Adaptations (CBA)	190
Basic Concepts	192
Adaptation Schema	192
Adaptation Dimension	192
Adaptation Context	192
Inheritance of Component Configurations	192
Step-By-Step Example	192
Adding an Attachment UIBB for Managers	192
Adapting the Address Layout	197
Avoiding Unnecessary FPM Events	201
Setting the Adaptation Context Locally	201
Hiding of UIBBs	203
Navigation with Launchpads	204

Including a Launchpad in the User Interface	205
General Settings of Launchpads.....	206
Transporting a Launchpad	206
IF_FPM_NAVIGATION API	206
Integration: Navigation in the Event Loop	210
IF_FPM_NAVIGATE_TO API	211
Restarting a WD ABAP Application	212
Extracting Launchpad Content and Launch Service.....	213
Suspend and Resume	213
Suspending via Static Launchpad Customizing for URL Application Category.....	214
Suspending via Static Launchpad Customizing for Web Dynpro ABAP or Web Dynpro Java Application	214
Suspending via Launchpad API.....	214
Resuming a Suspended Application	215
Handling Dialog Boxes	215
Triggering a Data-Loss Dialog Box in the FPM Event Loop	216
Handling Application-Specific Dialog Boxes	216
Deferring Current Event Processing	216
Registering a Dialog Box.....	216
Resuming the Event.....	218
IF_FPM_WORK_PROTECTION Interface	218
FPM Message Management.....	219
Using the FPM Message Manager	220
IF_FPM_MESSAGE_MANAGER Interface	221
Methods for Reporting Messages	221
Mandatory Parameters.....	227
Methods for Raising Exception Messages.....	227
Method for Clearing Messages	228
Handling of FPM Message Manager in Non-FPM Dialog Boxes.....	229
Message Manager – ON_NAVIGATE Event	229
FPM Message Manager FAQ	230
Message Mapper	231
Enabling Message Mapper	231
Message Mapping Fields	231
Message Context	231
Message Categories	231
Message Namespace	232
Message Source	232
Generalization	232
Changing Message Types	235
Hiding Messages.....	235
Hiding Messages and Generalization	235
Logging Messages	235
Generalization	236

Mapping Message Variables.....	237
API Changes for Message Mapping	238
Customizing Tables for Message Mapper	239
Maintenance Views for Message Mapper.....	240
FPM Error Page.....	240
Structure.....	241
Features	241
Handling of Transactions	243
Using the Transaction Interface	243
Transaction Interface FAQ.....	244
IF_FPM_TRANSACTION Interface	244
Resource Management	246
Releasing a Component.....	246
Settings for Transient Behaviour.....	248
Setting the Transient Flag.....	249
Using IF_FPM_RESOURCE_MANAGER to Veto Release Decision.....	249
Using an FPM Application Controller.....	250
Implementing the Application Controller	250
IF_FPM_APP_CONTROLLER Interface	250
Using an Application-Specific Configuration Controller	251
Implementing an AppCC Component	251
Methods	252
Features	252
Implementing an AppCC Class.....	254
Sharing Data between UIBBs from Different Components	254
Using a Shared Data Component	254
Other Options for Sharing Data	255
Determining Navigation State Information at Runtime.....	255
Embedding an FPM Application	256
Constraints.....	257
FPM CHIP Integration.....	258
Structure of the UCW	258
Multi-Instantiability.....	258
Communication between FPM CHIPS	259
Creating a CHIP for a Single UIBB	259
Appendix I: Authorization Profiles.....	261
Appendix II: Building FPM Applications on BOL with NW703/WEBCUIF702	261
GUIBB Configuration with Generic BOL Feeder Class.....	261
Search GUIBB	261
Form GUIBB	262
List GUIBB	264
Tree GUIBB	265

Floorplan Configuration.....	266
BOL-Specific Settings.....	266
Wiring.....	267
OVP Application with Ex-place Navigation.....	269
Break-out Scenarios.....	272
Feeder Class Redefinition.....	272
Connector Class Redefinition.....	276
Transaction-Handler Class Redefinition.....	276
Freestyle UIBBs.....	276
Application Controllers.....	279
Special Topics.....	280
FPM BOL CHIP Integration.....	280
Appendix III: Guidelines for Edit Scenarios for List ATS UIBB.....	281
Objectives.....	281
Prerequisites.....	281
Change Log.....	282
Application Scenarios.....	283
Extension of Feeder Interface.....	283
Unique Key Mode.....	284
To-Dos for Application.....	284
Moving of Rows.....	285
Programming Examples.....	285
Stable Line Order Mode.....	287
To-Dos for Application.....	287
Moving of Rows.....	288
Programming Examples.....	289
Own Delta Handling.....	292
To-Dos for Application.....	292
No Delta Handling.....	292
To-Dos for Application.....	292
Appendix IV: Multi-Value Fields.....	293
How to Use a Multi-Value Field.....	293

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 such as 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 requirements are 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, FLUID (Flexible UI Designer), which allows you to edit your application and to configure it at design time.

The FPM event loop and its 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.

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

Method Name	Method Description
FLUSH	<p>This is the first method called after the FPM event loop has been started.</p> <p>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.</p> <p>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.</p>
NEEDS_CONFIRMATION	<p>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'.</p>
PROCESS_EVENT	<p>Within this method the UIBB completes the following tasks:</p> <ul style="list-style-type: none"> • Checks for local consistency (validation, missing data, etc). • Perform the actual event processing. <p>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.</p> <p>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 <code>IF_FPM_TRANSACTION_CHECK_BEFORE_SAVE</code>.</p>

	<p>Besides the consistency check this method contains the actual processing of the event. For this, the current event can be identified through the attributes <code>MV_EVENT_ID</code> and <code>MO_EVENT_DATA</code> on the passed on event instance <code>io_event</code>. Depending on whether the event is processed successfully or not, the exporting parameter <code>EV_RETURN</code> must be filled with either <code>IF_FPM_CONSTANTS~GC_EVENT_RESULT-OK</code> or <code>IF_FPM_CONSTANTS~GC_EVENT_RESULT-FAILED</code>.</p> <p>A typical implementation of <code>PROCESS_EVENT</code> is shown below:</p>  <pre> 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. </pre> <p>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 <code>EV_RETURN = IF_FPM_CONSTANTS~GC_EVENT_RESULT-DEFER</code>.</p>
<p>AFTER_FAILED_EVENT</p>	<p>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.</p>  <p>Selecting an option in a 'Lead' field in a table triggers 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.</p> <p>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.</p>
<p>PROCESS_BEFORE_OUTPUT</p>	<p>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.</p>

Creating a Simple FPM Application

The following pages explain how to create a very simple, *Hello World*, FPM application based on a Guided Activity Floorplan (GAF). The application will contain 2 road steps.

This process is performed in the *Web Dynpro ABAP Workbench (transaction SE80)*.

A more detailed explanation of FLUID and how to use it follows thereafter.

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 usually present:

- a floorplan-specific component (`FPM_GAF_COMPONENT` or `FPM_OIF_COMPONENT`)
- a component for the Header Area (`FPM_IDR_COMPONENT`) - not present in OVP floorplan configurations

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

1. You construct an FPM application by completing the following steps:
2. Create a Web Dynpro Component with the required UIBBs and implement the Web Dynpro interface `IF_FPM_UI_BUILDING_BLOCK`.
3. Create a Web Dynpro Application and specify parameters according to which floorplan instance you are using.
4. Using the FPM configuration editor, FLUID (Flexible UI Designer), create a configuration for the application.
5. Test your application.

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 of the *Create Object Entry Directory* dialog box, enter the relevant package.
6. Save your entry. The preview displays your new (inactive) Web Dynpro Component.
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* in the toolbar.
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.
 - a. Ensure you are in edit mode.
 - b. Expand the *Views* node and double-click the existing view. The view appears in the preview.
 - c. In the *Layout* tab, right-click the *ROOTUIELEMENT* container and choose *Add Element*.
 - d. In the *Create Element* dialog box, add your own ID and select the type of UI element you want to add.
 - e. In the *Properties* Section, enter `HELLO` in the *Text* property. Choose *Save* and your text appears in the preview.
2. Choose *Activate*.
 - a. In the *Activation* dialog box, select all associated, inactive components and choose *OK*.
3. Add a second view:
 - a. Right-click the *View* node and choose *Create*. Give your view a name and choose *OK*.
 - b. 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):
 - a. Right-click the *Windows* node and choose *Create*.
 - b. In the *Web Dynpro: Create Window* dialog box, enter a Window name and choose *OK*.
 - c. The preview automatically displays the *Window* tab. In the *Window Structure* column, there is a node with your new Window's name.
 - d. 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).
 - e. 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 the *Web Dynpro Applications* folder and choose *Create*.
2. 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.
3. Enter the following information to create a GAF application:
 - a. **Component:** `FPM_GAF_COMPONENT`

- b. *Interface View*: FPM_WINDOW
 - c. *Plug Name*: Default
4. Save your entries.
 5. 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.

Parameter	Parameter Description
FPM_SHOW_MESSAGE_LOG	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.
FPM_MAXIMUM_MESSAGE_SIZE	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.
FPM_HIDE_CLOSE	With this parameter, you can hide the <i>Close</i> button on the FPM toolbar for your application.

There are also predefined Web Dynpro parameters which can be chosen using the value help. The parameter `WDENABLEUIELEMENTSHIDE` controls whether users can hide UI elements via a right mouse-click („implicit personalization“). The implicit default is `TRUE`. However, according to UI guidelines, this option should be disabled. To do this, this parameter should be added and the value should be left blank.

Alternatively, you may uncheck the corresponding checkbox in the application parameters block of the application configuration.

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_GAF_COMPONENT` interface.

Procedure

1. In the *Object Navigator*, right-click your new Web Dynpro Application and choose *Create/Change Configuration*. The *Editor for the Web Dynpro ABAP Application Configuration* screen 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 *New*. 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_GAF_COMPONENT`
 - `FPM_IDR_COMPONENT`
5. You will create configurations for both of these components. Select the individual component rows and choose the *Assign Configuration Name* button to enter names for both of the components. The names you enter are displayed as links.
6. Choose one of the links. The *Editor for the Web Dynpro ABAP Application Configuration* screen appears.
7. Choose *New* to create the new component configuration. The FPM configuration editor, FLUID, is displayed. You can now configure this component.

Note that 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 `FPM_GAF_COMPONENT`

1. The *Guided Activity Schema* displays 1 main step containing 1 UIBB placeholder (expand the row to see this if it is not immediately visible). 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 toolbar.
3. Choose the *Attributes* button on the main page toolbar to display the *Attributes* panel.
4. Choose the row containing the first UIBB placeholder to display its attributes.
5. Set the following attributes to the first window (with accompanying view) of your Web Dynpro component (containing the text 'Hello'):
 - *Component* (use the input help and search function to find your component).
 - *Window Name* (once you have entered the component name, the input help displays the list of views for that component).
6. Set the attributes of the second UIBB placeholder so those of the second window (with accompanying view) of your Web Dynpro component.
7. Choose *Save*.

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 FPM_IDR_COMPONENT

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

1. Return to the application hierarchy which displays both component configurations of your application configuration. You can do this in the following ways:
 - Click the link in the breadcrumb (just above the *General Settings* panel in the work area)
 - Open the *ABAP Workbench* (transaction SE80), locate your application configuration and choose *Start Configurator*. In the *Editor for the Web Dynpro ABAP Application Configuration* choose *Continue in Change Mode*.
2. Choose the link for the IDR component. FLUID opens and displays the attributes for the element *IDR Basic*.
3. In the *Attributes* panel enter an application title and choose *Save*.

Result

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

Testing your FPM Application

Procedure

In the main page toolbar, choose *Additional Functions and Test*.

A new browser window opens displaying the text Hello for the first step in the GAF roadmap. The title of the application (which you entered in the IDR component) should also be visible.

Choose the second step in the roadmap to display Welcome to the World of FPM.

FPM Application Creation Tool

The FPM *Application Creation Tool (ACT)* significantly reduces the effort involved in creating a new FPM application.

The tool itself is a WD application, provided by FPM, which allows application developers to create FPM applications and their corresponding configurations for all three available floorplans (OIF, GAF, and OVP).

The *ACT* also allows users to create applications for adaptable FPM components.

Starting the ACT

1. Open the *ABAP Workbench (transaction SE80)* and open the *APB_FPM_CONF* package.
2. Navigate to the *Web Dynpro -> Web Dynpro Applications* folder in the hierarchy and choose the *FPM_CFG_BO_MODEL_ACT* WD application.
3. Choose *Test* from the context menu.
The system opens a new browser displaying the ACT.

Creating a New Application using the ACT

To create an FPM application, complete the following steps:

1. Enter the following information:
 - Enter the name of an existing WD application or enter a name for a new application.
 - Description (optional)
2. Choose *Apply Namespace* in the toolbar to automatically apply a namespace to the WD application and the corresponding configurations.
3. (Optional) To create the application as an adaptable component, select the *Create Adaptable Configuration* checkbox. Enter an adaptation schema from the dropdown list. FPM automatically proposes a name for the adaptation configuration.
4. Choose a floorplan from the *Select Floorplan* dropdown list. FPM proposes names for the following components in the *Proposed Configuration Names* table:
 - *Application Configuration*
 - *Floorplan (Component) Configuration*
 - *Header (IDR) Configuration* (for OIF and GAF floorplans only)
5. (Optional) You can edit the configuration names proposed by FPM and enter descriptions for the configurations.

6. Choose the *Next* button on the main toolbar and enter the package and transport details.
7. Choose *Save* on the main toolbar. The ACT creates the new FPM application and component configurations and displays the following links in the browser:
 - *Launch Configuration Editor*
 - *Test Application*



The ACT creates only the components and layout of an application; it does not create the code for it. If you created your application without choosing an existing WD component, your application appears empty at runtime (except for the icons on the FPM toolbar).

8. Choose one of the following links to continue:
 - a. *Launch Configuration Editor*
Choose this link to edit the component or application configurations. This link displays the *Hierarchy Browser* which displays all the individual component configurations which you have just created.
 - b. *Test Application*
Choose this link to execute the newly created FPM application.

You can configure the application configuration at a later date in the following ways:

- Open the *ABAP Workbench (transaction SE80)* and navigate to the relevant WD application folder. Choose the application and choose the *Start Configurator* button on the toolbar. This opens the application in edit mode using the FPM configuration editor, FLUID.
- Run the application and choose the *Configure Page* icon on the toolbar.

FLUID (Flexible UI Designer)

FLUID is the configuration editor for FPM application configurations and their individual components. It replaces all previous individual FPM configuration editors.

You view or edit your application configurations and their components in design mode using FLUID.

You use it to enhance the application user interfaces and fit them to your business needs.

Note that any properties that you set in FLUID take precedence over settings that have been made in relevant feeder classes.

Launching FLUID in Different Modes

You can launch FLUID in the following ways:

- **In Administrator Mode**
From the application UI at runtime, use the links *Customize Page* (to access the floorplan component) and *Show Customizable Areas* (to access the GUIBB component).

Changes you make to a configuration in this mode are made on the **Customizing** level. FLUID displays a colored bar above the title bar of your configuration to indicate that you are working on the customizing level.

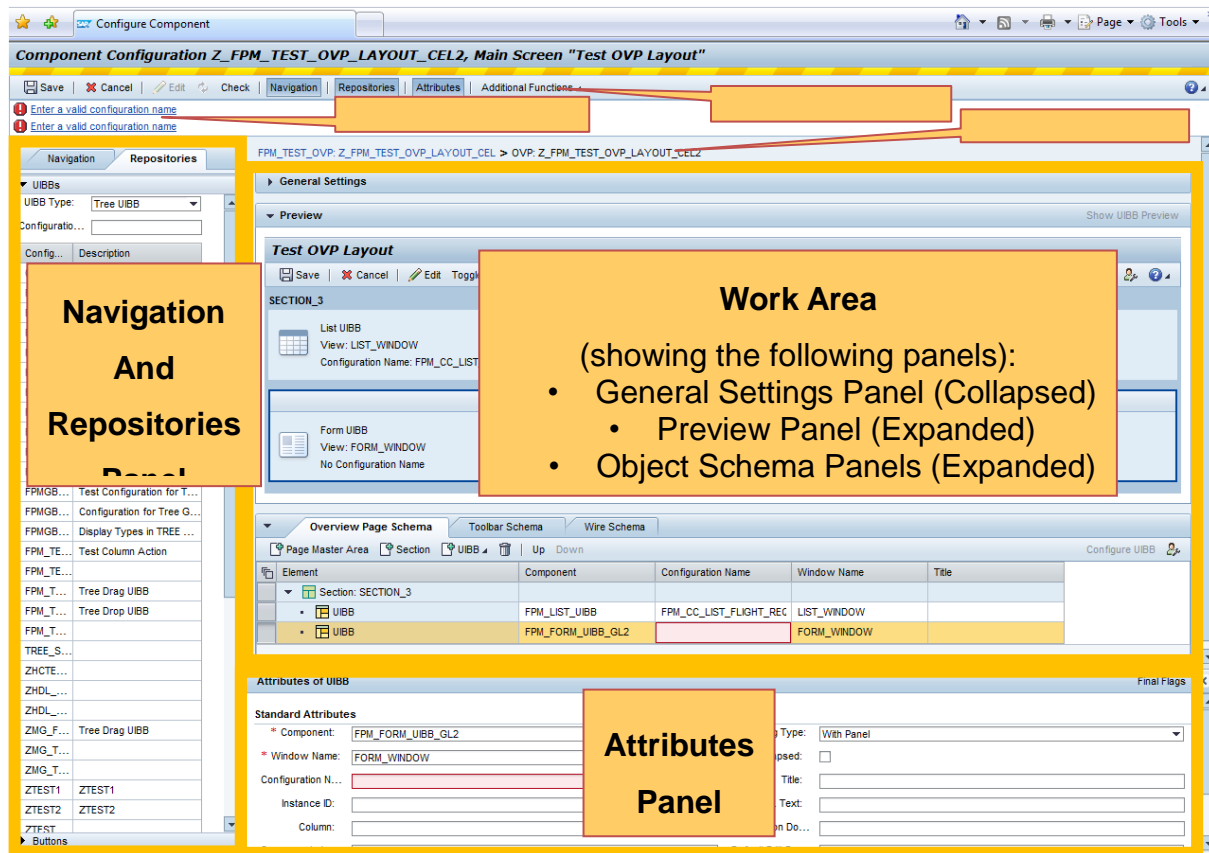
When you choose either of the buttons mentioned above, and no Customizing of a configuration exists, the system displays a Customizing dialog box for you to create one.

- **In Expert (Developer) Mode**
From the application UI at runtime, use the links *Configure Page* (to access the floorplan component) and *Show Configurable Areas* (to access the GUIBB component).

Changes you make to a configuration in this mode are made on the **Configuration** level.

Structure and Layout of FLUID

The screenshot below shows the general layout of FLUID layout:



Title Bar

This displays the name of the component configuration and the active screen or page within it.

Page Toolbar

As well as switching between editing and display modes, this toolbar provides you with the following buttons:

- Check (makes consistency checks)
- Repositories (controls the display of the *Repositories* panel)
- Navigation (controls the display of the *Navigation* panel)
- *Attributes* (controls the display of the *Attributes* panel)
- Additional Functions:
 - Deep-Copy (copies application configurations and their individual component configurations)
 - New Window
 - Show Properties (such as application author)
 - Test (displays the application at runtime)
 - Enhance (for creating modification-free enhancements)
 - Reset (for a complete reset of customizing changes)

Message Area

The *Message Area* is the primary area for the display of system messages. You can determine the appearance of the *Message Area* using the *Message Settings* properties located in the *General Settings* panel.

Breadcrumb

Use this to move to different components within your application configuration. When navigating between different components, FLUID ensures that the original work mode (edit or display) for each component remains the same.

Work Area

The *Work Area* is divided into a number of panels, each of which is described in the following sections.

General Settings Panel

The information displayed in the *General Settings* panel is component-dependent. Information that relates to the floorplan or GUIBB component as a whole is displayed here. This panel is divided into the following sections:

- **Classification Settings**
Allows you to classify component configurations, for example as 'Financials'; you can then determine that configurations classified as 'Financials' are provided with additional UIBBs (for example, the Chart GUIBB, which is not available to all users of the FPM framework).
- **Transient Settings**
Allows you to determine the transient behavior of the application.
- **Message Settings**
Allows you to determine the appearance of the *Message Area* and whether or not the message log is displayed.
- **Additional Settings**
Additional settings are specific for the OVP floor plan and the individual GUIBB components. They relate to the floorplan or GUIBB component as a whole (for example, the width of a list) and not to individual UI elements.

In the *General Settings* panel you can also access the following settings:

- Final flags
- Floorplan settings:
 - Application controller settings
 - Event action types
 - Message Mapper settings
- Feeder class settings (specific to GUIBBs)
- Drag-and-Drop settings (specific to GUIBBs and visible in this panel only when defined in the feeder class)

Preview Panel

The *Preview* panel is the developer's main design-time area, displaying the interface of the application as a set of configurable components. The way objects are displayed in this panel depends on which level of your application, floorplan or UIBB, you are on:

Floorplan Level

Individual UIBBs appear in the *Preview* panel as separate boxes; each UIBB identified by its component and configuration names.

You can navigate into the individual UIBBs directly from the *Preview* panel. You can tell if a UIBB is configurable by moving the mouse over the UIBB box; the box changes color and an icon appears in the top right corner of the box. Click the icon to navigate directly to the UIBB component. If the UIBB component is actually a GUIBB component, you remain inside FLUID; otherwise, you are transferred to the standard Web Dynpro editor.

You can also select the individual toolbar elements in this panel and edit their attributes.

UIBB Level

The *Preview* panel displays the UIBB as it appears at runtime, for example, as a table or a form.

On both levels, a **context menu** allows you, for example, to navigate to and configure the different components in your application, add or delete UIBBs, add new sections and toolbar elements.

You can also access the properties of toolbar elements from this panel and move elements around on the toolbar.

You can drag and drop items between the *Repositories* panel and the *Preview* panel.

Object Schema Panel (<Floorplan>/ <GUIBB> Schema Panel)

This panel outlines the structure of the individual GUIBB or floorplan and displays its UI elements. You can move individual elements within the schema by using the *Up* and *Down* buttons or by dragging them to a new location. You can select elements in the schema and edit their attributes in the *Attributes* panel.

This panel also provides you with the following actions:

- Add or remove individual UI elements to or from the GUIBB or floorplan, for example:
 - Add or delete a main or sub-step in a GAF component
 - Add or delete a section and UIBB to and from an OVP component
 - Add or delete a column in a list or hierarchical list component
 - Add a Master Page Area to an OVP floorplan
- Configure a UIBB

You can switch directly from this view to editing the individual UIBBs. Use the breadcrumbs above the *Work Area* to navigate back again.

- Configure the IDR (of OIF and GAF components)

You can drag and drop items between the *Repositories* Panel and the Object Schema Panel.

Toolbar Schema Panel

This panel displays the toolbar plan and individual toolbar UI elements in your application. You can add additional toolbar elements and edit their attributes displayed in the *Attributes Panel*. Note that the available toolbar elements vary according to the type of page you are working with. The position of some toolbar elements is determined by the FPM framework and is unchangeable.

Wire Schema Panel

This panel displays the individual wires between the UIBBs. It is available only for the floor plan components and the Composite UIBB.

It allows you to add and remove wires and also gives you access to the *Graphical Wire Editor*, a tool for displaying the wiring in your application in a graphical, easy-to-understand way.

Graphical Wire Editor

This editor allows you to view, edit and create the wires you need for your application. The editor supports *drag-and-drop*. A central work area, the *Wiring Pane*, displays UIBBs that are currently wired, along with information regarding their outputs and connector classes. UIBBs that are not wired are displayed in the *Available UIBBs* repository panel, and can be dragged into the work area. UIBBs are represented graphically as boxes. Outputs are displayed as colored arrows on the edges of the boxes, with labels displaying the output type.

To create a wire, drag an arrow from one box onto another box. The *Connector Data* dialog box appears, allowing you to choose the *Connector Class* and related *Connector Parameters*. To display this dialog box at any time, double-click the connecting lines between the UIBBs. Labels of connecting lines are provided by the connector interface.

Wires that you created using the *Graphical Wire Editor* appear on the *Wire Schema* tab.

Attributes Panel

The attributes of configurable UI elements are displayed in this area. Whether you can edit these elements depends on the UI element you have selected. You can see changes that you make to an attribute in this panel also in the *Preview* panel. In the *Attributes Panel*, you also have access to the following:

- Settings for *Final Flags*
- Properties dependent on *Display Types*
- Action assignments for buttons

You can display the *Attributes* panel using the *Attributes* toggle-button on the main toolbar.

Navigation/Repositories Panel

You can use this panel to display the *Navigation* or *Repositories* panels. Display them using the respective toggle-buttons on the page toolbar.

Navigation

The *Navigation* panel allows you to move between and select the pages you would like to configure in your application, for example, the initial screen, the main screen of an application, an edit page, or a dialog box.

Here, you can also add new, delete and copy pages in your application. The type of pages you can add depends on the type of floorplan instance.

You can store multiple variants of a selected floorplan for each FPM application. A variant provides you with 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. See 'Variants' section in this document.

Repositories


The *Repositories* panel provides you with a list of repository items (for example, component configurations, fields for forms and lists, search criteria) which can be dragged to and from the *Object Schema* or *Preview* panel in the *Work Area*.

Different component configurations exist for different floorplans and GUIBBs. You can search for specific component configurations by entering value in the *Component*, *View* or *Configuration* fields or you can display all component configurations for a specific type of UIBB.

The *Repositories* panel also contains a *Buttons* section from which you can drag various button type elements to and from the *Toolbar Schema* or *Preview* panel.

Changing the Layout of FLUID

You can change the layout of FLUID in the following ways:

- Personalize the *Work Area*
 - Display panels in a collapsed or expanded state
 - Drag and drop panels within the *Work Area*; drag and drop panels one under the other or place them on top of each other to form a tabstrip
- Use the *Personalize* button () in the panels to control, for example, which fields are displayed on the UI, and to discard any previous settings you made.
- Adjust the size of the *Work Area* using the splitter controls located at the bottom of and on the left side of the *Work Area*
- Switch the *Navigation*, *Repositories* and *Attributes* panels on or off



Changes that you make are maintained when you open the application again.

Working with FLUID

Working with FLUID is easy once you understand its layout. The following simple examples aim to get you started.



It is assumed in the following examples that you edit configurations in developer mode.

Adding an Existing UIBB to your Application

You add existing UIBBs (freestyle, GUIBB, RUIBB) to the floorplan component configurations of your application.

1. Run the application and choose the *Configure Page* icon on the toolbar. FLUID is opened and displays the application in design mode.
2. Choose the *Navigation* button in the main toolbar to display the *Navigation* panel and choose the page in which the UIBB will be inserted.
3. Choose the object schema panel (*Guided Activity Schema*, *Overview Page Schema*, *Object Instance Schema*, and so on); this is the panel in which you insert your UIBBs.
4. Select an appropriate row in the table and choose *Add UIBB* in the toolbar of the object schema panel.
5. In the dropdown list of the *Add UIBB* button, choose the type of UIBB that you want to add to your floorplan.

The UIBB is added in the row that you selected. If you choose a Generic UIBB (GUIBB) from the list (for example, Form Component), the system automatically completes the fields *Component* (Name) and *Window Name*. You need only to enter the name of your configuration. If you choose a freestyle UIBB, you must enter details for all three fields, *Component* (Name), *Window Name* and *Configuration Name*.

6. (Optional) Choose the *Feeder Class* and *Edit Parameters* buttons to select the relevant feeder class and its parameters.
7. Choose *Save*.
8. To see your changes at runtime, choose *Test* under the *Additional Functions* button on the toolbar.



You can only edit your UIBB further, using FLUID, if you have entered a configuration name.

Changing the Title of a Step inside an Application based on the GAF Floorplan

1. Run the application and choose the *Configure Page* icon on the toolbar. FLUID is opened and displays the application in design mode.
2. Choose the *Navigation* button in the main toolbar to display the *Navigation* panel and choose the page on which the GAF roadmap exists (you can see the roadmap immediately in the *Preview* panel).
3. Choose the relevant step in the *Preview* panel or in the *Guided Activity Schema*.
4. Choose the *Attributes* button in the main toolbar to display the *Attributes panel* and enter a new name for the step in the *Main Step Name* (or *Substep Name*) field.
5. Choose *Save*.

6. To see your changes at runtime, choose *Test* under the *Additional Functions* button on the toolbar.

Editing the Form Component inside an Application

1. Run the application and do **either** of the following steps:
 - a. Choose the *Configure Page* button on the toolbar.
 - i. FLUID is opened and displays the application in design mode. Choose the *Navigation* button in the main toolbar to display the *Navigation* panel and find the page on which the form component exists (you can see the form immediately in the *Preview* panel).
 - ii. Choose the form component in the *Preview* panel; the box changes color and a button appears in the top right corner of the box.
 - iii. Choose the button in the box.
 - Or
 - b. Choose the *Show Configurable Areas* button on the toolbar.
 - i. Any UIBB that is configurable changes color when you hover the mouse over it. Click on the configurable form component in your application.
2. FLUID now displays the settings relevant for the form component only (and not for the floorplan component). Notice that there are now different fields in the *General Settings* tab; these are fields specific to the form component.
3. Make your changes and choose *Save*.
4. To see your changes at runtime, choose *Test* under the *Additional Functions* button on the toolbar.

Adding a New Button to the Toolbar in a Floorplan Component of an Application

1. Run the application and choose the *Configure Page* icon on the toolbar. FLUID is opened and displays the application in design mode.
2. Choose the *Toolbar Schema*.
3. Choose the *Add Toolbar Element* button in the toolbar of the *Toolbar Schema*. The *Select Toolbar Elements* dialog box opens.
4. Choose a toolbar element from a range of buttons, button-choices and links. The toolbar element now appears in the *Toolbar Schema* and is visible also in the *Preview* panel.
5. Choose the *Attributes* button on the main toolbar to display the *Attributes* panel and edit the fields of your new toolbar element (for example, enter a name for your button in the *Text* field or assign an FPM event ID).
6. Choose *Save*.
7. To see your changes at runtime, choose *Test* under the *Additional Functions* button on the toolbar.

Moving back to a Floorplan Component from a GUIBB Component

Once you have made changes to the GUIBB component (for example, a form or list component), you can navigate back to the floorplan component by choosing the relevant link in the *Breadcrumb* above the *Work*

Area. The breadcrumb allows you to move between the GUIBB component and the floorplan component, and finally back to the Hierarchy Browser.

Limitations

It is recommended that you use Internet Explorer 7 or higher.

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.

The following table describes the methods of the feeder class model:

Method Name	Method Description
GET_NAMESPACE	Returns the namespace of the underlying application area. Method is called at design time.
SET_CONNECTOR	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.
GET_IMPORT_KEY	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.
GET_OUTPORTS	Provides a table of outports comprising the object key, the port type an identifier and a descriptive text. Method is called at design time.
GET_OUTPORT_DATA	Returns an object reference carrying the actual data identifier for a certain port. Method is called at runtime.



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_FPM_CONNECTOR connector interface comprises an interface, IF_FPM_CONNECTOR_DEF, defining the access by the FPM framework and an interface IF_FPM_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

Method Name	Method Description
GET_PARAMETER_LIST	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.

GET_PARAMETER_VALUE_SET	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.
INITIALIZE	With this method the connector is initialized with the parameter values. This method is called by the FPM runtime upon UIBB instantiation.
SET_INPUT	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.

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

Method Name	Method Description
GET_OUTPUT	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.
CREATE_ENTITY	Creates and returns a data entity which can be arbitrarily typed. This method can be called by an action handler of the UIBB for example in the PROCESS_EVENT method of a feeder class.
IS_CREATE_ALLOWED	Returns a Boolean indicator whether entity creation is allowed. This method can be called by the UIBB at PBO to dynamically control the activation of create buttons for example to maintain the action usage parameter in the GET_DATA method of a feeder class.

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

Method Name	Method Description
START	Receives basic data like the FPM message handler and application parameters. This method is called once at application start.
AFTER_FLUSH	This method is called after FLUSH has been called for all current UIBBs. It can be used to flush buffers.
AFTER_PROCESS_EVENT	This method is called after PROCESS_EVENT has been called for all current UIBBs. It can be used for handling transactional

	events for example <code>SAVE</code> or <code>CHECK</code> . Moreover, it can be used to collect messages which here not handled inside UIBBs and to forward them to the FPM message handler.
<code>AFTER_PROCESS_BEFORE_OUTPUT</code>	This method is called after <code>PBO</code> has been called for all current UIBBs. It can be used to collect messages at the latest possible point in time before screen output.
<code>AFTER_NEEDS_CONFIRMATION</code>	This method is called after <code>NEEDS_CONFIRMATION</code> has been called for all UIBBs. It can be used to analyze and add confirmation requests.
<code>IS_DIRTY</code>	This method can be used to indicate a dirty state for the work protection mode

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 must 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 retrieve the feeders default configuration. You may adjust it according to your needs.



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

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`.

Creating an FPM Application on BOL

Below are the main steps for creating FPM applications on BOL objects. For a more detailed description of this procedure, see Appendix 'Building FPM Applications 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. Choose 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.
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 outport 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 your 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.



You can also create your UIBBs on BOL out of the floorplan configuration. If you choose 'FPM_BOL' as the application area in the *General Settings*, the standard BOL feeder classes are set as default when creating configuration for forms, lists, search and tree GUIBBs out of the floorplan configuration via drill down.

Design Time with the FPM Configuration Editor

FLUID allows you to 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.
- Within the actions area there is a button choice *Add UIBB*. It has the entries for adding a form, a list, a tabbed component, and a search, and so on. 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 FLUID depends on the type of floorplan instance you are using in your application.

OIF Instance

Note that this floorplan has been superseded by the Overview Page (OVP) floorplan.

All new applications for object instances should be built using the OVP floorplan.

In an OIF application, FPM displays your UIBBs in multiple tabs. The *Object Instance Schema* 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.
- **Sub-View:**
You add your UIBBs to the sub views. An FPM application must have at least one UIBB for each sub view. FLUID 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.

GAF Instance

In a GAF application, FPM displays your UIBBs as individual steps in the overall roadmap. For GAF applications, the *Guided Activity Schema* 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. FLUID 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.
- **Substep:**
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 subview.

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

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.

In FLUID, navigate to the floorplan component and on the *Guided Activity Schema*, choose the main step for which you want to add a substep. Choose the *Add Substep* button on the schema toolbar.

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 latest 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 *Check*, *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 (*You Can Also* and *Related Links*)



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`. The *Close* button can be activated - see SAP Note 1526176.

Differences between an OIF and a GAF Toolbar

OIF Application

There is only one toolbar in every OIF variant. 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 main step toolbar.

Adding Elements to a Toolbar

1. In FLUID, 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 *Toolbar Schema*. 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.



You can also drag toolbar elements from the *Repositories* panel on to the *Toolbar Schema*. See the section on using FLUID.

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.

Toolbar Button Name	Toolbar Button Description
Activation Function	The button within this category is placed on the first position on the toolbar. Basically, it is the most important event on the screen. That is why this button is intended primarily to be used as <i>Save</i> button. As most applications require a <i>Save</i> button, the <i>FPM</i> configuration editor automatically adds this button to your configuration by default. The FPM Event <code>FPM_Save</code> is set as the default FPM Event ID but you can edit this.
Alternate Function	If another important event (besides the activation function) exists, applications can define an alternate function. This is placed directly besides the activation function button.
Other Function	All other application-specific events are defined within the <i>Other Functions</i> area. These buttons are placed after the standard buttons.
You can Also/ Related Links (navigation)	These two toolbar elements provide navigation menu options (in a button-choice) 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	See previous remark about this button in this section. See also SAP Note 1526176.
Exit to Main Step (GAF only)	This is available only to substeps. 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 (GAF only)	Extra attributes are available for <i>Next Step</i> in the final roadmap step.
Final Action (GAF only)	You can explicitly define your own final action. This button is displayed directly next to the Next Step button in the roadmap.

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.

Toolbar Element Attribute	Toolbar Element Attribute Description
Element ID	Enter an Element ID if you want to change the properties of a toolbar element dynamically during runtime.
Sequence Index	This allows you to choose the order in which your application-specific UI elements (for example <i>Other Function</i> buttons,) appear on the toolbar or in the hierarchy. The toolbar elements which FPM automatically adds to the toolbar cannot be rearranged using this attribute.

Repeat Sel. Action (Repeat Select Action)	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 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.

Toolbar Button	Event Raised	Floorplan Instance
<i>Activation Function</i>	self-defined via configuration	OIF
<i>Alternate Activation</i>	self-defined via configuration	OIF
<i>Check</i>	<code>GC_EVENT_CHECK</code>	OIF
<i>Close</i>	<code>GC_EVENT_CLOSE</code>	OIF and GAF
<i>Delete Object</i>	<code>GC_EVENT_DELETE_CURRENT_OBJECT</code>	OIF
<i>Edit</i>	<code>GC_EVENT_EDIT</code>	OIF
<i>Exit to Main Step</i>	<code>GC_EVENT_EXIT_TO_MAIN_STEP</code>	GAF
<i>Load Draft</i>	<code>GC_EVENT_LOAD_DRAFT</code>	OIF
<i>New</i>	<code>GC_EVENT_NEW</code>	OIF
<i>Next Object</i>	<code>GC_EVENT_NEXT_OBJECT</code>	OIF
<i>Next Step</i>	If no final action is defined: <code>GC_EVENT_NEXT_STEP</code> If a final action is defined: the self-configured event in the final action node and the next step event are raised	GAF
<i>Other function</i>	self-defined via configuration	OIF and GAF

<i>Previous Object</i>	gc_event_previous_object	OIF
<i>Previous Step</i>	gc_event_previous_step	GAF
<i>Print</i>	gc_event_print	OIF
<i>Print Preview</i>	gc_event_print_preview	OIF
<i>Read Only</i>	gc_event_read_only	OIF
<i>Redo</i>	gc_event_redo	OIF
<i>Refresh</i>	gc_event_refresh	OIF
<i>Save As</i>	gc_event_save_as	OIF
<i>Save Draft</i>	gc_event_save_draft	OIF and GAF
<i>Send</i>	gc_event_send	OIF
<i>Start Over</i>	gc_event_start_over	OIF
<i>Undo</i>	gc_event_undo	OIF

Toolbar Button-Choice	Event Raised	Floorplan Instance
<i>Send</i>	self-defined via configuration	OIF
<i>Print</i>	self-defined via configuration	OIF
<i>Print Preview</i>	self-defined via configuration	OIF
<i>New</i>	self-defined via configuration	OIF

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:



```
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 following table.

Method Name	Method Description
DEFINE_BUTTON	With this method either standard buttons or application-specific

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

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:

Location of UI Elements	Parameters
Main Step	<code>VARIANT_ID</code> <code>MAINSTEP_ID</code>
Sub-Step	<code>VARIANT_ID</code> <code>MAINSTEP_ID</code> <code>SUBVARIANT_ID</code> <code>SUBSTEP_ID</code>
Initial Screen	<code>SCREEN</code>

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



```
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
skipped >
        iv_mainstep_id     = < optional; 'mainstep_1'; current mainstep if skipped
>
        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 properties
of application-specific buttons 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 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>

```

```

lo_cnr_gaf->define_button_choice(
    EXPORTING
        iv_variant_id      = < optional; e.g.'variant_1'; current variant if
skipped >

        iv_mainstep_id     = < optional; 'mainstep_1'; current mainstep if
skipped >
        iv_subvariant_id   = < optional; 'subvariant_xyz' >
        iv_substep_id      = < optional; 'substep_99' >
        iv_function         = < e.g. OTHER_FUNCTIONS (appl-specific button-
choices)>
        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 > ).

```

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:



```

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 following table.

Method Name	Method Description
DEFINE_BUTTON	With this method either standard buttons or application-specific buttons can be created and edited. The parameter <code>IV_FUNCTION</code> defines the button type (see <code>IF_FPM_CONSTANTS=>gc_button</code>). The <code>ELEMENT_ID</code> is needed if application-specific buttons must be changed subsequently.
DEFINE_BUTTON_CHOICE	With this method either standard button-choices or application-specific button-choices can be created and edited. The parameter <code>IV_FUNCTION</code> defines the button-choice type (see <code>IF_FPM_CONSTANTS=>gc_button_choice</code>). The <code>ELEMENT_ID</code> is needed if application-specific buttons must be changed subsequently.
CREATE_SEPARATOR	Use this method to create a separator at runtime in the <code>OTHER_FUNCTIONS</code> area (application-specific).
DEFINE_YOU_CAN_ALSO	Use this method to define launchpads for the <i>You Can Also</i> menu bar for (see Navigation API chapter).
DEFINE_RELATED_LINKS	Use this method to edit the menu bar for <code>RELATED_LINKS</code> (see Navigation API chapter).
GET_BUTTONS	This method determines which buttons (and their configurations) are to be shown in the toolbar.
GET_BUTTON_CHOICES	This method determines which button-choices (and their configurations) are to be shown in the toolbar.
GET_SEPARATORS	This method determines the positions of the separators in the

	toolbar (only in the <i>Other Functions</i> area).
GET_RELATED_LINKS	This method determines the contents of the <i>Related Links</i> menu in the toolbar.
GET_YOU_CAN_ALSO	This method determines the contents of the <i>You Can Also</i> menu in the toolbar.

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:



```

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 >
    iv_hotkey         = < optional; Button-Hotkey >
    iv_action_type    = < optional; action type of the button event -
standard or validation-independent >
    iv_image          = < optional; Button-Icon >
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 `UIBB_PREVIEW` method with the `IV_INTERFACE_VIEW` importing parameter, which is the actual visible view, and the `EV_PREVIEW_WINDOW` exporting parameter.

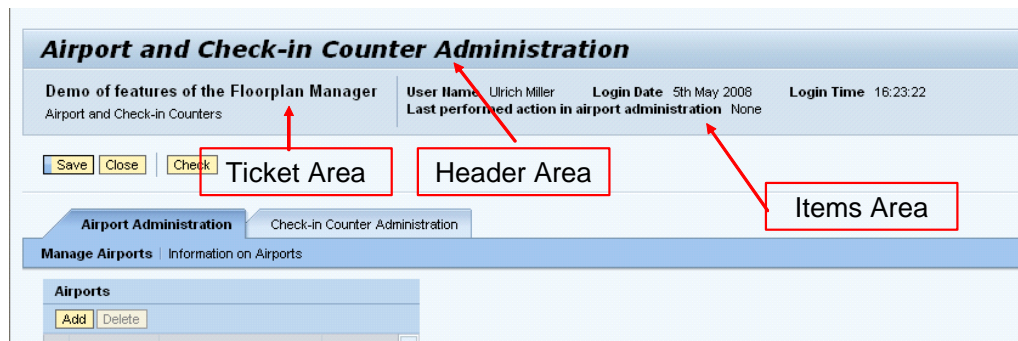
The application itself needs to check that the window which is returned is defined and will work at design time.

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:

Methods for IDR Header Area

Method Name	Method Description
GET_APPLICATION_TITLE	Retrieves the title text and its tooltip.
SET_APPLICATION_TITLE	Displays a new title text and tooltip in the header area.
SET_HEADER_VISIBILITY	Makes the header area visible or invisible.

Methods for IDR Ticket Area

Method Name	Method Description
GET_TICKET	Retrieves the texts of the ticket top, ticket bottom and their tooltips.
SET_TICKET	Displays new texts of the ticket top, ticket bottom and their tooltips.
SET_TICKET_VISIBILITY	Makes the ticket area visible or invisible.

Methods for Items Area

Method Name	Method Description
ADD_ITEM_GROUP_BY_VAL	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 <code>CHANGE_ITEM_GROUP_BY_VAL</code> . The method <code>ADD_ITEM_GROUP_BY_REF</code> 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.
<code>CHANGE_ITEM_GROUP_BY_VAL</code>	Changes the label and values that were passed to the IDR via the method <code>ADD_ITEM_GROUP_BY_VAL</code> .
<code>ADD_ITEM_GROUP_BY_REF</code>	Similar to <code>ADD_ITEM_GROUP_BY_VAL</code> . 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. 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.
<code>ADD_NAVIGATION_ITEM</code>	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 <code>instance</code> and <code>role</code> . 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.
<code>CHANGE_NAVIGATION_ITEM</code>	Use this method to edit a pair of label / navigation links that you added using the method <code>ADD_NAVIGATION_ITEM</code> . 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.
<code>ADD_IMAGE_ITEM</code>	Adds pairs of label/icons to the IDR.
<code>CHANGE_IMAGE_ITEM</code>	Edits a label/icon pair that you added using the method <code>ADD_IMAGE_ITEM</code> .
<code>CONTAINS_ITEM_GROUP</code>	Checks whether a certain item group exists within the IDR.
<code>REMOVE_ITEM_GROUP</code>	Removes a certain item group from the IDR.
<code>INITIALIZE_ITEMS</code>	Clears all items from the IDR.
<code>SET_ITEMS_VISIBILITY</code>	Edits the visibility of the item area (the visibility status of all items, not just single items).

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`.
 3. 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 helpful explanation of a subview, initial screen, main step, or substep at runtime. 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 either the *Text* or *Documentation Object* attributes of an *Explanation* element.

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. Locate the required application in the ABAP Workbench and launch FLUID.
2. Ensure you are in edit mode.
3. On the object schema tab, select the subview, main step, substep, or initial screen and choose *Add Explanation* in the toolbar.

4. In the *Text* field in the *Attributes* panel, overwrite the suggested text with the quick help text you would like to be displayed to the user at runtime.
5. Save your changes.

Creating Quick Help Linking to a Documentation Object

1. To create a documentation object, choose ► *SAP Menu* → *Tools* → *ABAP Workbench* → *Utilities* → *SE61 – Documentation*⚡.
2. Choose *General Text* as the document class.
3. Enter a technical name for the documentation object.
4. Choose *Create* and enter the desired quick help text.
5. Choose *Save Active*.
The documentation object is now created and can be assigned as a quick help.
6. Locate the required application in the ABAP Workbench and launch FLUID.
7. Ensure you are in edit mode.
8. On the object schema tab, select the subview, main step, substep, or initial screen and choose *Add Explanation* in the toolbar.
9. In the *Documentation Object* field in the *Attributes* panel, enter the technical name of the documentation object.
10. Save your changes.

Variants

In some cases, the final configuration of an OIF view switch, GAF roadmap or OVP page 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. 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` (respective `IF_GAF`). This handler object provides the API with information to manipulate the floorplan configuration at runtime.
2. To select the variant, use the `SET_VARIANT` method of this object as follows:

OIF Instance



```
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( ).
  ...
endmethod
```

GAF Instance



```
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( ).
```



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

Initial Screen

The *Initial Screen* is an optional screen. It is composed of one or more UIBBs.

Open the *Navigation* panel in FLUID to add an initial screen.

FPM adds the *Continue* 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:



```
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):



```

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 iv_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
endif
endif

```



For an initial screen that contains only GUIBBs (without free-style UIBBs and without an AppCC), you must use the `GET_DATA` method of the feeder class. There is no `PROCESS_BEFORE_OUTPUT` method available for applications if only GUIBBs are used (and therefore, it is not possible to raise the FPM event `LEAVE_INITIAL_SCREEN`).

Confirmation Screen

Confirmation screens are available for all floorplans.

The floorplan type determines when and where exactly a confirmation screen appears in an application. In OIF and OVP instances, the confirmation screen appears only when the object currently being processed in the application is deleted. In a GAF instance, it appears as the final step at the end of the roadmap to inform the user that the action he or she has just executed has been successfully completed

You can add confirmation screens to your application via the *Navigation* panel in FLUID (*Add Page* button).

You can configure separate confirmation screens for individual variants for OIF and GAF instances. In OVP instances, you configure as many confirmation screens as you need and specify which confirmation screen you must navigate to when deleting an object and raising the `IF_FPM_CONSTANTS=>GC_EVENT-DELETE_CURRENT_OBJECT` event.

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:



```
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 )
```

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):

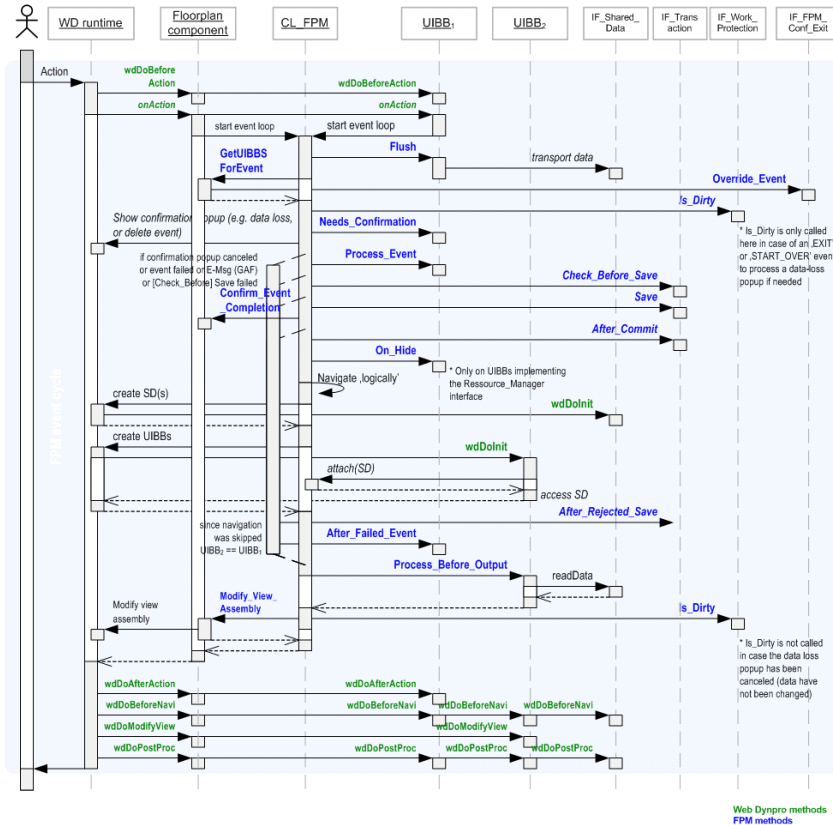


```

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:

Method	Method Description
DOINIT	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).
DOBEFOREACTION	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.
Action handler (ONACTION...)	<p>The registered Web Dynpro action handler is called. You then have the following options:</p> <ul style="list-style-type: none"> • 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:

- IF_FPM_UI_BUILDING_BLOCK
- IF_FPM_TRANSACTION, IF_FPM_WORK_PROTECTION
- 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:

- IF_FPM_APP_CONTROLLER

- IF_FPM_SHARED_DATA
- IF_FPM_OIF_CONF_EXIT (or IF_FPM_GAF_CONF_EXIT)

Overview Page Floorplan (OVP)

You can use the OVP floorplan (OVP) to model an application user interface that displays an overview of the most important data of a business object instance to the user and that enables editing, deleting and creating new data. An OVP application consists of a set of pages and the navigation between these pages.

The pages and the navigation between them can be configured in the FLUID.

Technically speaking, the OVP is based on the WD component `FPM_OVP_COMPONENT`. An OVP configuration is therefore a component configuration of that WD component.

Structure of an OVP

Page

Each OVP configuration consists of at least one page (also known as a Content Area). Depending on its business semantics, a page is of one of the following page types:

- **Initial Page**
This page is optional and can either be used to present a table of objects (for example, recently-used objects or a result list of a search) that can be selected for further processing, or to enforce input of important data that is required before the Main Overview Page of the business object instance is displayed.
- **Main Overview Page**
This page is the heart of any OVP application. The Main Overview Page presents all the data of a business object instance to the user that is relevant within the application context. Depending on the application, editing the business object instance is possible in-place directly on the Main Overview Page.
- **Sub-Overview Page**
If a business object is very complex and consists of sub-objects, an OVP application can provide one or more Sub-Overview Pages in addition to the Main Overview Page.
- **Edit Page**
Depending on the application, editing the business object instance can also be done on separate, dedicated Edit Pages.
- **Confirmation Page**
The OVP application can be terminated by displaying a Confirmation Page (for example, if the business object instance is to be deleted)
- **Dialog Box**
Dialog boxes are used for short interactions with the user that must be completed before continuing with a task.

Section

A page consists of one or more sections which you use to structure a page.

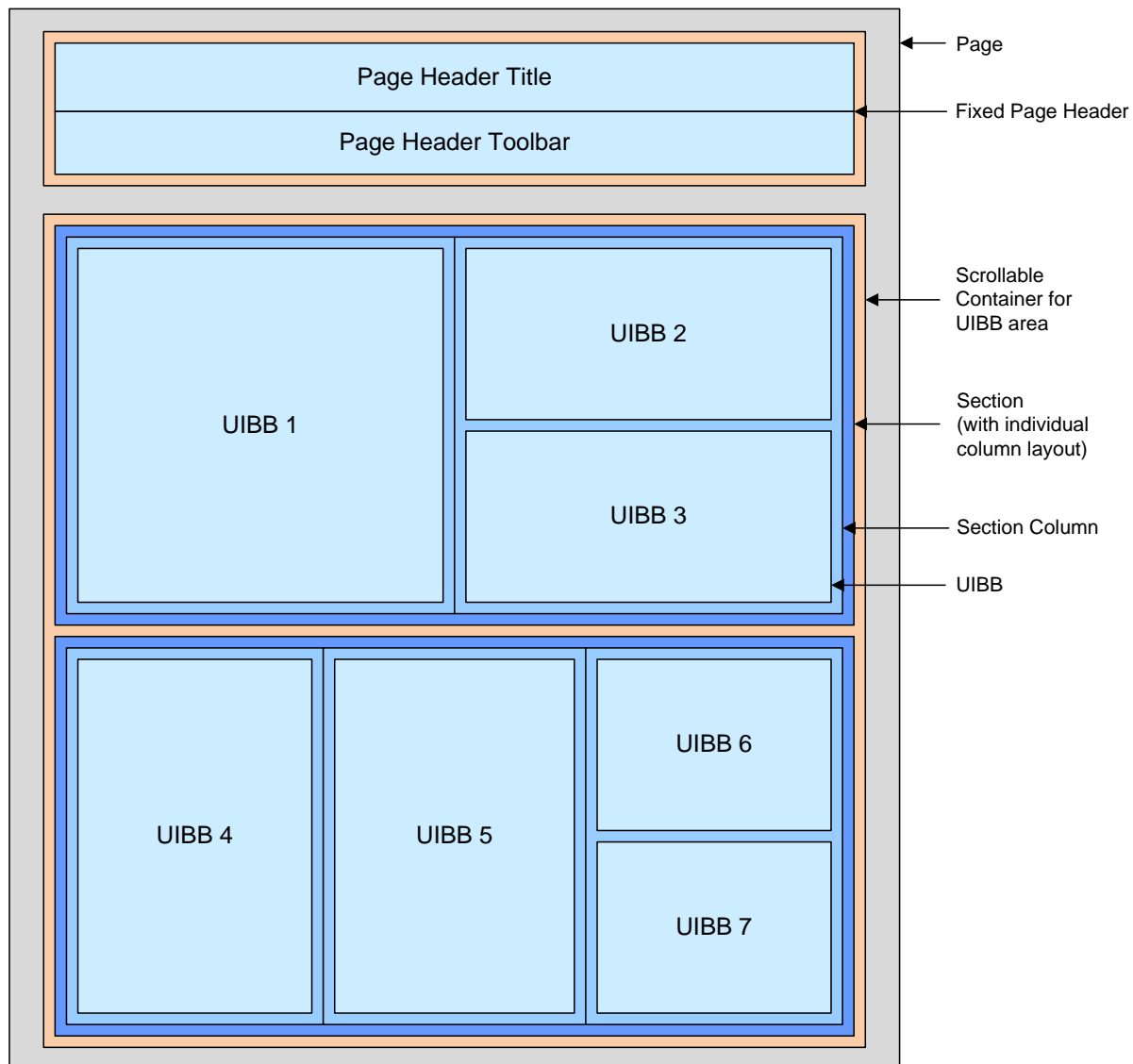
Each section has its own layout and can contain numerous application-specific user interface building blocks (UIBBs) and/ or generic user interface building blocks (GUIBBs).

Various layouts are possible for a section, differing mainly in the number of columns into which the section is divided. You assign the UIBBs and GUIBBs to the individual columns within a section.

UIBBs / GUIBBs

A UIBB or a GUIBB inside a section is rendered either as a panel or flat without a panel. If rendered as panel, it is called an 'Assignment Block'. Its title is displayed in the panel header. An assignment block may have an own toolbar. The buttons, toggle buttons, and button choices of that toolbar are also rendered in the panel header.

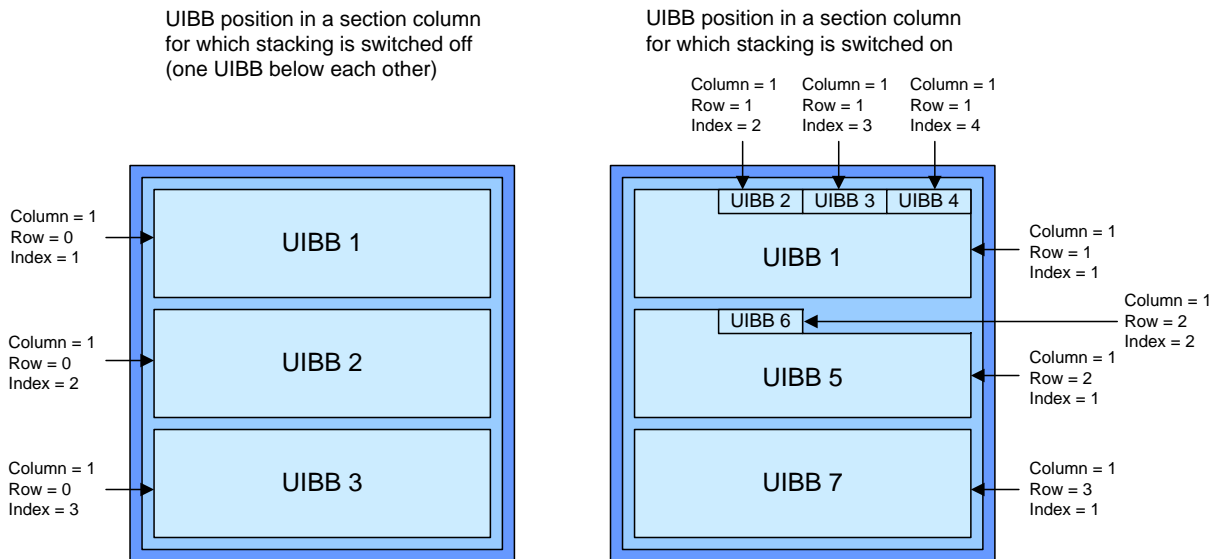
The following figure illustrates the structure of an OVP page:



Stacking

You can enable each section column for 'Stacking'. If stacking is enabled for a section column, the UIBBs can be dragged at runtime and dropped onto each other in that column, thus building a stack in which each UIBB is represented by a tab. This is known as a *Stacked Panel*.

In the configuration editor, UIBBs can be stacked using appropriate values for the properties *Column* (= *Location*), *Row*, and *Index* (see figure below):



There are two possible states for a stacked panel:

- Expanded**
 In an expanded stacked panel there is exactly one UIBB that is expanded, that is, its property *Collapsed* is set to *FALSE*. All other UIBBs are collapsed, that is, their property *Collapsed* is set to *TRUE*. The expanded UIBB is the active UIBB in the stacked panel. Its property *Default In Stack* is set to *TRUE*.
- Collapsed**
 In a collapsed stacked panel all UIBBs are collapsed, that is, their property *Collapsed* is set to *TRUE*. There is, though, exactly one UIBB in the stacked panel with the property *Default In Stack* set to *FALSE*. If the stacked panel is expanded, this UIBB becomes the active UIBB.

Page Master

The Page Master can either be a list or a hierarchical tree. It allows an object selection that is valid for all assignment blocks on the page. On selection of an object in the Page Master, all assignment blocks must react and, depending on the use case, probably refresh their content.

A Page Master is not displayed in a collapsible assignment block but in a non-collapsible panel in a special screen area.

There are two possible positions for a Page Master:

- **Above**
Page Master is above the assignment block area with full width and all the assignment blocks below a separator
- **Left**
Page Master is left of the assignment block area with full height and all the assignment blocks on the right side of a separator. Here, it is recommended that you use 1/3 of the screen width for the Page Master and 2/3 for the assignment blocks.

According to UI guidelines there is a header '*Details: <identifier of selected item>*' above the assignment blocks. Normally the master table has a default selection (first line). If it is necessary to start with no selection in the master table, the details are not shown and a text is displayed ('No Detail Selected'). The latter functionality is not provided by the FPM, but it is in the responsibility of the application to behave in this way

The Page Master table can be hidden with a toggle-button in the page toolbar. This button can be named according to the use case (for example, in the screenshot below it is 'Overview').

The screenshots below show an Overview Page with a Page Master on the left side and an Overview Page with a Page Master on-top, respectively.

The screenshot displays the SAP Fiori interface for 'Adcom Computer - Opportunity ABC'. It illustrates two different Page Master (FPM) configurations:

- Left Configuration (Red Box):** The Page Master is positioned on the left side of the screen, containing a table of 'Open Opportunities'. The 'Assignment Block area' (blue box) occupies the right side of the screen, displaying 'Opportunity Details' for the selected item.
- On-Top Configuration (Red Box):** The Page Master is positioned at the top of the screen, containing a 'Header' (red text) and 'Opportunity Details' (blue box). The 'Assignment Block area' (blue box) occupies the bottom portion of the screen.

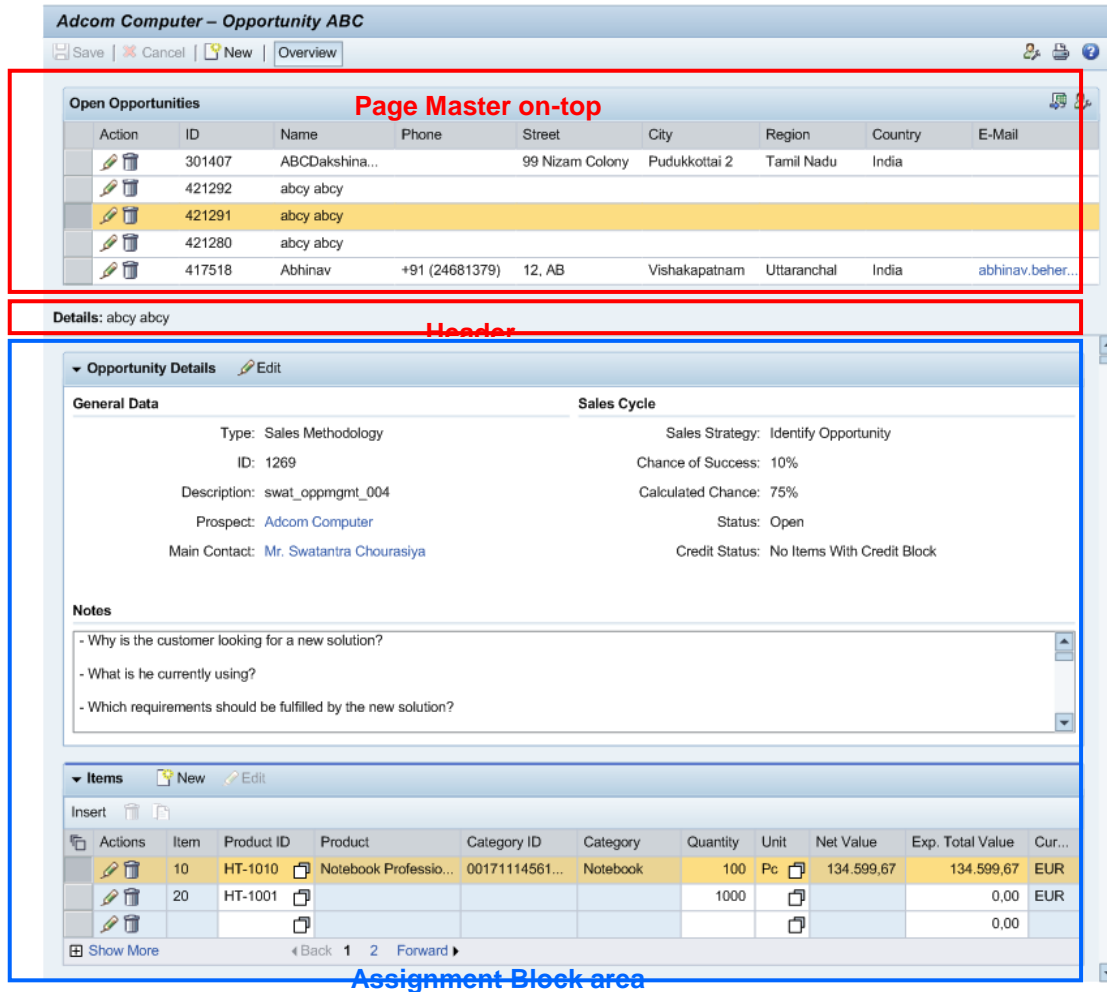
The 'Open Opportunities' table in the left configuration is as follows:

Action	ID	Name
	301407	ABCDakshina..
	421292	abcy abcy
	421291	abcy abcy
	421280	abcy abcy
	417518	Abhinav
	301407	ABCDakshina..
	421292	abcy abcy
	421291	abcy abcy
	421280	abcy abcy
	417518	Abhinav
	301407	abcy abcy
	421292	ABCDakshina..
	421291	abcy abcy
	421280	abcy abcy
	417518	Abhinav
	301407	Abhinav
	421292	abcy abcy
	421291	Abhinav
	421280	ABCDakshina..
	417518	abcy abcy
	301407	Abhinav
	421292	ABCDakshina..
	421291	abcy abcy
	421280	abcy abcy
	421291	abcy abcy

The 'Opportunity Details' section in the right configuration shows the following information:

- General Data:**
 - Type: Sales Methodology
 - ID: 1269
 - Description: swat_opprgmt_004
 - Prospect: Adcom Computer
 - Main Contact: Mr. Swatantra Chourasiya
- Notes:**
 - Why is the customer looking for a new solution?
 - What is he currently using?
 - Which requirements should be fulfilled by the new solution?
- Items Table:**

Actions	Item	Product ID	Product	Category ID	Category	Quantity
	10	HT-1010	Notebook Professo...	00171114561...	Noteb...	100
	20	HT-1001				1000



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 Off	Personalization On
The expand/collapse state of panels does not persist.	The expand/collapse state of the panels persists.
Dragging and dropping of UIBBs is possible and their arrangement does not persist.	Dragging and dropping of UIBBs is possible and their arrangement persists.
No <i>Personalization Editor</i> is available.	A <i>Personalization Editor</i> can be opened at runtime, allowing you to perform additional activities.

Currently, the following attributes are stored in the personalization settings:

- Content Area: *ID*
- Section: *ID, Layout Type, Index*
- Section Column: *Stacking*
- UIBB: *UIBB key, Column (=Location), Row, Index, Hidden, Collapsed, Default In Stack*
- Page Master Area: *Position, Sash Position, Sash Position Mode*

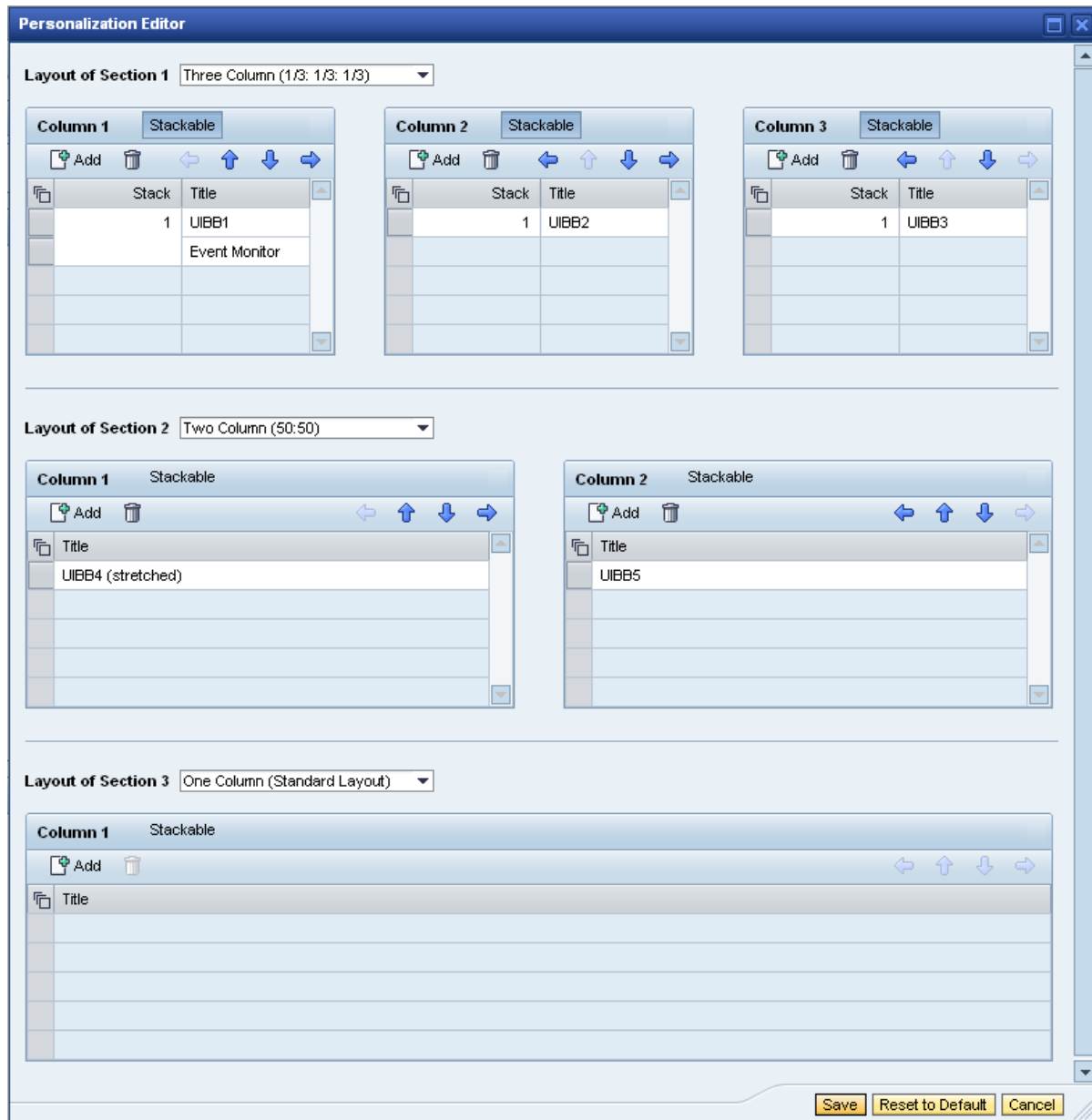
Note that the OVP always considers all UIBBs of a page when storing personalization settings that are in any way relevant for an UIBB. This means that potentially the state of all UIBBs is stored in personalization even though there was just a change in the properties of one UIBB. If there is already customizing data available only the delta changes compared to the customizing data are stored in personalization.



If there are personalization settings for a particular page on top of customizing data and the customizing data for that page is changed or deleted afterwards the resulting page layout might be unexpected when the application is started again in end-user mode. This is a consequence of the stored delta changes in personalization.

Personalization Editor

To open the *Personalization Editor* at runtime, click the *Personalize* icon in the page toolbar. The following is a screenshot of the *Personalization Editor*.



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
- Determine whether UIBBs in a section column should be stacked
- 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, press the toggle button *Stackable* on the respective section column table.

When the application is run in administration 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 administrator personalization (if it exists); if there is no administrator personalization, choosing *Reset to Default* restores the original configuration settings.

CAUTION.

Unexpected effects might occur if an application enables personalization but also performs dynamic changes at runtime via the OVP CNR API `IF_FPM_CNR_OVP` or the OVP Application Controller API `IF_FPM_OVP`. The interaction of personalization and dynamic changes might become apparent on the screen when the end-user resets the personalization settings for an OVP page. During such a reset, the personalization settings of the page will be deleted and the state of the initial page configuration **plus the performed API changes** will be restored. The reason for the latter is that the API changes should not get lost during the reset (which was a basic requirement when the personalization functionality was developed). Whereas dynamic changes at runtime, that are independent of personalization data, should not be a problem, a conflict might arise in the case of dynamic API changes that have been performed on personalized data. In this case, the reset might lead to unexpected results. It is the responsibility of the application to avoid such situations.



Be aware that the dynamic changes at runtime are not directly stored in the personalization settings as this is not an action performed by the end-user. However, dynamic changes might have an effect on the page layout which the end-user can personalize afterwards. In this case, the complete current layout of the page is stored in the personalization settings, including the changes that were made dynamically.

Toolbars

In total, there are three different kinds of toolbars in an OVP configuration:

- Global toolbars (Page Header toolbar)
- Assignment Block toolbars (UIBB toolbar)
- Page Master toolbars

You can define buttons, toggle buttons, and button choices for each kind of toolbar in the *Toolbar Schema* of the configuration editor.

However, there is an additional method for assignment blocks and page master toolbars. If the UIBB, on which the assignment block or the page master is based, exposes actions 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 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.

External Navigation Menus

As with the OIF and GAF, the OVP supports navigation to external applications from the runtime UI through the use of a launchpad. Currently, the following links are available for the floorplans:

- *You Can Also*
- *Related Links*

The names are defined by the UI guidelines. You can configure both links in the configuration editor. Technically, they are identical. However, they can be configured with a different parameter set. In contrast to the OIF and GAF, the OVP allows you to configure *You Can Also* and *Related Links* per single OVP page.

During runtime, *You Can Also* and *Related Links* are rendered as left-aligned button-choices after the standard and application-specific buttons, toggle buttons, and button choices in the page toolbar. This is according to UI Guidelines 2.0. The menu items of the button-choices are the entries that are defined in the Customizing for the respective launchpad (transaction LPD_CUST).

In addition to the static configuration, the OVP provides the possibility to get, add, change, and remove the external navigation data during runtime. For this, the interface `IF_FPM_CNR_OVP` offers the following methods:

- `GET_EXTERNAL_NAVIGATIONS`
- `ADD_EXTERNAL_NAVIGATION`
- `CHANGE_EXTERNAL_NAVIGATION`
- `REMOVE_EXTERNAL_NAVIGATION`

The methods can be called from feeder classes, application-specific UIBBs or application configuration controllers. An interface reference of `IF_FPM_CNR_OVP` can be retrieved via the FPM service manager in the following way:

```
DATA: lo_fpm          TYPE REF TO if_fpm,
      lo_fpm_cnr_ovp TYPE REF TO if_fpm_cnr_ovp.

* Get reference to FPM OVP CNR API
lo_fpm = cl_fpm_factory=>get_instance( ).
lo_fpm_cnr_ovp ?= lo_fpm->get_service( cl_fpm_service_manager=>gc_key_cnr_ovp ).
```

The above methods are also available for the interface `IF_FPM_OVP`. They can be called in method `OVERRIDE_EVENT_OVP` of an OVP application configuration controller using the object reference that is passed to this method.

Default Actions

The OVP allows the definition of default actions, that is, of an action that is triggered automatically when the Enter key is pressed. Such a default action can, but need not, be related to a button element (normal button, toggle button, or button-choice). An example of where a button-related default action makes sense is an Initial Page with a *Continue* button that should automatically be triggered on pressing Enter.

Default actions that are related to a button element can be defined by selecting the entry Enter as a hotkey for the corresponding button element. This can be done in the configuration editor at design time, or by using the OVP CNR API or the OVP Application Configuration Controller API at runtime. Note that only one button element on a page can have the hotkey Enter. This is ensured by the configuration editor as well as by the runtime APIs.

Default actions that are not related to a button element can be defined by specifying an FPM event on page level, or technically speaking, on the 'Content Area' level. This can also be done in the configuration editor at design time or by using the runtime APIs of the OVP.

If both a default action related to a button element and a default action that is not related to a button element are defined on a page, the default action that is related to the button element has the higher priority; on Enter, the action of the button element is triggered, not the action that is defined on the page level.

Edit / Display Mode

According to UI guidelines there is an explicit edit and display mode for an overview page, the behavior of which should be the following:

- Normally, an overview page opens in display mode. This means all assignment blocks show their data read-only and there are no editable fields. The buttons *Save* and *Cancel* in the page toolbar are disabled. An *Edit* button in the assignment block title bar offers the possibility to change the data of this assignment block. A central *Edit* button in the page toolbar of the overview page (optional) changes all assignment blocks to edit mode. If one or several assignment blocks of the overview page are in edit mode, the *Save* and *Cancel* buttons in the page toolbar are enabled.
- After pressing the *Edit* button of an assignment block, the application normally switches to edit mode in-place and gives the user the possibility to edit the data. The *Edit* button is disabled when the assignment block is in edit mode. An assignment block that was switched to edit mode stays in this mode until the user clicks *Save* (saving all changes in all assignment blocks) or *Cancel* (discarding all changes in all assignment blocks) in the page toolbar. A save or cancel event sets all assignment blocks on the overview page back to display mode (if the overview page has a display mode).

You can enable an explicit edit and display mode in an Overview Page using one of the following ways:

- In the *General Settings* of the OVP floorplan configuration
- Using the application parameter `FPM_EDIT_MODE`
- Using the URL parameter `FPM_EDIT_MODE`

The first alternative has the lowest priority whereas the last one has the highest. This means that the application parameter overrules the configuration setting whereas the URL parameter overrules both the application parameter and the configuration setting.

Currently, the following values are supported:

- ' ': No Mode Handling (default)
- 'E': Edit
- 'R': Read-Only

For the value *No Mode Handling* (the default setting), no special logic is processed by the FPM regarding display or edit mode. This means that the application behaves in the same way it did before this feature was introduced. There is no automatic enablement or disablement of *Edit*, *Save* or *Cancel* buttons done by the FPM. The UIBBs do not receive information about their current edit mode state.

For the value *Read-Only* the application starts at runtime displaying all UIBBs as read-only, whereas for the value *Edit*, the application starts at runtime displaying all UIBBs in edit mode; the latter is typically used for

'create' scenarios. There is an automatic enablement or disablement of *Edit*, *Save* or *Cancel* buttons done by the FPM. All GUIBBs receive information about their current edit mode state through the importing parameter `IV_EDIT_MODE` of method `GET_DATA` of the respective feeder interface. All free-style UIBBs can access information about their current edit mode through method `GET_UIBB_EDIT_MODE` of the FPM runtime interface `IF_FPM`.

If the edit and display mode is used in an application a local *Edit* button can be configured in the UIBB toolbar. This button must trigger the FPM event `FPM_LOCAL_EDIT`. If this button is pressed at runtime the UIBB is transferred into edit mode and is no longer read-only and the button will be automatically disabled. In the global toolbar, *Save*, *Cancel* and *Edit* buttons can be configured with the FPM events `FPM_SAVE`, `FPM_CANCEL`, and `FPM_EDIT`, respectively. `FPM_SAVE` and `FPM_CANCEL` transfer the application globally into display mode, that is, no UIBB stays in edit mode. `FPM_EDIT` transfers the application globally into edit mode, that is, all UIBBs are transferred into edit mode. In global edit mode, the global edit mode button and all local edit mode buttons are disabled. In global display mode, the global save and cancel buttons are displayed.

As well as the *Edit* button there are also other buttons allowed for a UIBB that transfer the UIBB directly into local edit mode. One example is an *Add* button for lists. To enable automatic switching of a GUIBB into local edit mode when such a button is pressed, a new indicator `IS_IMPLICIT_EDIT` is available in the structure `FPMGB_S_ACTIONDEF` which is used in the feeder class method `GET_DEFINITION` for defining non-exposable or exposable feeder class actions. To enable switching of an application-specific UIBB into local edit mode when pressing an application-specific button, two cases have to be distinguished: for exposable actions, the structure `FPMGB_S_ACTIONDEF` is used again for the action definition and therefore the indicator `IS_IMPLICIT_EDIT` is available; for non-exposable actions, a new attribute can be set at the FPM event of the button in order to enforce switching to the edit mode. This attribute is defined in class `CL_FPM_EVENT` and its name is `MV_IS_IMPLICIT_EDIT`.

Important Precondition:

The UIBBs in the configuration must support edit mode, that is, they need to react appropriately on their current edit mode state.

Processing Mode for Collapsed UIBBs ('Lazy Load')

Broadly speaking, this feature allows you to determine whether collapsed UIBBs should be immediately instantiated (and take part in the FPM event loop) when they are visible on the page, independent of their collapsed/expanded state, or whether the UIBBs should be instantiated in a 'lazy' manner and take part in the FPM event loop only if they are wire sources or after they become visible and expanded.

The processing mode is specified at application level, not at individual UIBB level.

You can set the processing mode in the following ways:

- In the *General Settings* of the OVP configuration
- Using the application parameter `FPM_COLLAPSED_UIBB_PROC_MODE`
- Using the URL parameter `FPM_COLLAPSED_UIBB_PROC_MODE`

Alternative 1 has the lowest priority, alternative 3 has the highest. This means that the application parameter overrules the configuration setting whereas the URL parameter overrules both the application parameter and the configuration setting.

Currently, the following values are supported for the processing mode:

- ‘ ‘ – Participate (in FPM Event Loop) (Default)
UIBBs which are visible on the page are taken into account during the FPM event loop. The UIBBs are instantiated by the FPM runtime immediately when they are visible on the page. The instantiation of the UIBB and their participation in the FPM event loop is independent of the collapsed/expanded state of the UIBBs.
- ‘D’ – Defer Participation (in FPM Event Loop; Lazy Instantiation)
Collapsed UIBBs which are visible on the page are not taken into account during the FPM event loop unless they are wire sources or Composite UIBBs that contain UIBBs that are wire sources. They are instantiated 'lazy' which means that they are instantiated only after they have been expanded. Note that Composite UIBBs must be instantiated before they have been expanded in order to determine which UIBBs they contain.

For FPM applications using wiring, the FPM runtime automatically prevents 'lazy loading' for UIBBs which are declared as wire sources, that is, if a UIBB is a wire source it will be instantiated and take part in the FPM event loop even if it is in a collapsed state. This technique ensures that lazy loading spoils the logical dependencies between UIBBs. Note, however, that wiring is the only declarative dependency definition known to the FPM runtime. If there are dependencies explicitly hard-coded in application UIBBs, feeder classes, or application controllers, and so on, application development is responsible to ensure that correct application behavior is not spoiled by lazy loading.

WARNING.

The use of the processing mode for collapsed UIBBs with values other than the default value is critical since not all UIBBs are prepared to be instantiated and to take part in the FPM event loop only if they are visible and expanded or if they are a wire source. For example, if there is coding in a UIBB that reacts on the FPM event `FPM_START`, the processing mode *D* should not be used. Collapsed UIBBs on the page (and which are not wire sources) would not be processed during this start event and the specific logic for the start event would not be executed.

Technical UIBBs

Application areas in the Business Suite may use a kind of generic controller UIBB that never has a screen presence but must participate in the FPM event loop to fulfill its controller functionality. The Technical UIBBs feature allows you to define a UIBB that is never shown on the user interface but that is processed in the FPM event loop as a normal visible UIBB.

To specify a UIBB as a technical UIBB, select the value *T* (*Hidden but Processed in Event Loop (Technical)*) for the attribute *Hidden Element* of the UIBB in the configuration of the OVP Floorplan.

Initial Search Page & External Navigation

According to UI guidelines, an OVP application can have an initial page. An initial page is used if it is necessary for an end-user to enter data before going to the main screen. The initial page can also be used to present a table of objects (for example, recently used objects) that can be selected for further processing. In particular, an initial page can contain a search and a search result list. A result list entry may contain one or several links which, when clicked, enforce navigation to further details of the selected object. In addition, a result list may offer a dedicated button that is intended for the creation of a new object.

For example, an OVP application for purchase orders could have an initial screen that enables the end-user to search for certain purchase orders. The purchase orders are then displayed in a result list below the purchase order search criteria. This result list could contain a column showing the *Purchase Order ID* and another column showing the *Contact Name*. Clicking the *Purchase Order ID* would trigger navigation to an overview page for the purchase order details, whereas clicking the *Contact Name* would trigger navigation to an overview page for contact details. Both overview pages, however, belong to different OVP applications. In addition, the purchase order result list could offer a dedicated button that is intended for the creation of a new purchase order. Clicking this button would restart the OVP with a different initial page.

Ideally, the navigation from an initial search page to the overview page of the same object would be done in-place. In principle, this is no problem for the OVP floorplan. However, since there is no general in-place navigation concept for SAP applications that are realized with different technologies, the current guidelines prescribe to open a new application window whenever applications or reports are launched from the result list of a search. In the OVP floorplan this kind of navigation is realized with the launchpad and suitable launchpad customizing.

Technical Details:

1. When an OVP application is started, the standard FPM event `FPM_START` is triggered. The target page is determined in the method `GET_UIBBS_FOR_EVENT` of the OVP floorplan. This can either be a page of type 'INIT' (Initial Screen), if there is one, or a page of type 'MAIN' (Main Overview Page).

The standard process of selecting the target page is as follows: The default initial page will be the starting page. If there is no default initial page, any other initial page is selected instead. If no initial page exists, the default main overview page is selected as the target page. If no default main overview page exists, any other main overview page is selected instead. If there is neither an initial page nor a main overview page, an error message is shown.

There is the possibility to overrule the standard process. For this, an additional URL parameter `FPM_START_PAGE_ID` or an application parameter `FPM_START_PAGE_ID` can be specified (the URL parameter has the higher priority). The value of the parameter must be a valid ID of an initial page or of a main overview page of an OVP application. In this case, the OVP is started with the respective page.

To display a search page as the start page of an OVP application, the search page has to be configured in the configuration editor as the default or the only initial screen, or the application has to be started using the URL parameter or application parameter `FPM_START_PAGE_ID`. The search page must contain a Search GUIBB as well as its built-in result list or a List GUIBB.

2. When the user has executed the search on the initial search page, the result list is shown. To enable the external navigation from the result list entries to an OVP application, the result list has to contain at least one 'link' column. For 'create' scenarios, a dedicated button in the result list toolbar that triggers an external navigation is also possible. Clicking a cell in a link column or the dedicated button for 'create' scenarios must trigger the navigation event `FPM_NAVIGATE`. As a minimum, the navigation event must be supplemented with suitable event parameters that identify an application via launchpad customizing. The following

FPM DDIC structures have been created to support the maintenance of the navigation event parameters in the configuration editor:

FPM_S_EXTERNAL_NAVIGATION_KEY:

COMPONENT	COMPONENT TYPE	DATA TYPE	LENGTH	SHORT DESCRIPTION
LPD_ROLE	APB_LPD_ROLE	CHAR	10	Role
LPD_INSTANCE	APB_LPD_INSTANCE	CHAR	32	Instance
LPD_APPLICATION_ALIAS	APB_LPD_APPL_ALIAS	STRING	0	Application Alias

FPM_S_EXTERNAL_NAVIGATION_BASE:

COMPONENT	COMPONENT TYPE	DATA TYPE	LENGTH	SHORT DESCRIPTION
.INCLUDE	FPM_S_EXTERNAL_NAVIGATION_KEY		0	External navigation key
SOURCE_ATTRIBUTE_NAME	FPM_SOURCE_ATTRIBUTE	CHAR	30	Source Attribute

FPM_S_EXTERNAL_NAVIGATION_INFO:

COMPONENT	COMPONENT TYPE	DATA TYPE	LENGTH	SHORT DESCRIPTION
.INCLUDE	FPM_S_EXTERNAL_NAVIGATION_BASE		0	External navigation base information
INITIAL_PAGE_PROC_MODE	FPM_INITIAL_PAGE_PROC_MODE	CHAR	1	Processing Mode of Initial Page
EDIT_MODE	FPM_EDIT_MODE	CHAR	1	Edit Mode for OVP Floorplan
START_PAGE_ID	FPM_CONTENT_AREA_ID	STRING	0	Start Page of the OVP Floorplan

DDIC structure `FPM_S_EXTERNAL_NAVIGATION_KEY` defines the general key for a launchpad entry that allows starting an external application. This DDIC structure is included in DDIC structure `FPM_S_EXTERNAL_NAVIGATION_BASE` which additionally contains the field component `SOURCE_ATTRIBUTE_NAME`. When clicking a search result list cell, this field may contain the name of an alternative list column that carries the relevant object key that is required to process the initial search page in the background. DDIC structure `FPM_S_EXTERNAL_NAVIGATION_BASE` is itself contained in DDIC structure `FPM_S_EXTERNAL_NAVIGATION_INFO`. This structure offers three more

field components, `INITIAL_PAGE_PROC_MODE`, `EDIT_MODE`, and `START_PAGE_ID`. The first one specifies the processing mode of the initial screen of the OVP application. It is described below in more detail. The second field component specifies the edit mode of the OVP application. The third field component specifies the page that should be displayed when the OVP application is started. In particular, this field component is relevant in 'create' scenarios.

To realize the navigation, the feeder class for the result list must expose the FPM event `FPM_NAVIGATE` in its action definition. Additionally, a DDIC structure, which includes the structure `FPM_S_EXTERNAL_NAVIGATION_BASE`, must be assigned as an event parameter structure. In the configuration of the result list, a column can be configured with display type *Link to Action*. Instead of the *Standard Action* default action assignment, the feeder class action `FPM_NAVIGATE` should be configured. For 'create' scenarios, a button that is based on the feeder class action `FPM_NAVIGATE` can be configured. As event parameters for the *Link to Action* or the button, the launchpad key role and instance ID can be maintained as well as the processing mode. If no launchpad key is maintained, a dynamic launchpad entry is created at runtime restarting the current application configuration externally, that is, ex-pace. Using the processing mode of the initial screen, it is possible to execute the search in the background, parameterized with the data from the selected result list entry providing a unique result; the main overview page is opened, displaying the data for this result.

To display details of a selected object, when navigating externally from an initial search page result list of an OVP application to another (or the same) OVP application, it must be ensured that the initial search page is not shown again but is processed in the background instead. For 'create' scenarios, an appropriate page must be displayed which allows data for the new object to be entered. The event which is raised when clicking a navigation link in the search page result list or on the dedicated button for the creation of new objects contains a special event parameter `INITIAL_PAGE_PROC_MODE`. This parameter can have the following values:

- (Blank) – Normal
Everything works as if the OVP application is started new and without any special logic
- E – Execute Search
The Search GUIBB triggers the FPM event `FPM_EXECUTE_SEARCH` in PBO in order to execute the search automatically in the background. The required search parameters are derived from the URL before the search is executed. However, no further navigation is triggered.
- L – Execute Search & Leave
The Search GUIBB triggers the FPM event `FPM_EXECUTE_SEARCH` in PBO. However, afterwards, the standard FPM event `FPM_LEAVE_INITIAL_SCREEN` is also triggered. It is expected that the search execution provides a unique search result. As a consequence, the OVP will no longer show the initial search page but navigate directly to the (default) main overview page. Alternatively, the target

page can be specified directly in the FPM event `LEAVE_INITIAL_SCREEN` using the event parameter `TARGET_CONTENT_AREA`.

- **C – Create**

The OVP is started new. However, the event parameter `START_PAGE_ID` is taken into account and enables the OVP to start directly with the specified page. This is required for 'create' scenarios in which a page other than the initial search page must be displayed.

3. Field content of the clicked link will be transported to the externally launched application as a URL parameter; the value is taken from the list cell which raised the event. However, if a different component name was specified in the field `SOURCE_ATTRIBUTE_NAME`, the content of this field is transmitted instead.

The name of the corresponding URL parameter is

`FPM_NAVI_SOURCE_KEY_ATTR_<component_name>`. If the processing mode is set in the configuration, the search is automatically executed with the parameter having the same name as the component name of the result list. If the component name in the search is different from the result list, a parameter mapping in the launchpad configuration needs to be defined (and hence, navigation via a static launchpad entry is mandatory). The result list feeder class needs to make sure that it transfers the first record to the main page content when the `FPM_LEAVE_INITIAL_SCREEN` event is triggered in the background processing.

EXAMPLE.

A search for purchase orders has a component `PO_ID` in the search attributes and search result list. The result list feeder class exposes the event `FPM_NAVIGATE` with the event parameter structure `FPM_S_EXTERNAL_NAVIGATION_INFO`. All that application development must do is to choose the display type *Link to Action* for component `PO_ID` and to assign the FPM event `FPM_NAVIGATE` in the result list configuration. In the event parameters, the field `INITIAL_PAGE_PROC_MODE` has to be set to *L (Execute Search & Leave)*.



If more than one key field is required for a unique search result, the single field component `SOURCE_ATTRIBUTE_NAME` in the `FPM_S_EXTERNAL_NAVIGATION_INFO` structure is not sufficient. In this case, the application can include the structure `FPM_S_EXTERNAL_NAVIGATION_INFO` into an application-specific structure containing additional field components for the key information. The additional field component names should begin with 'SOURCE_ATTRIBUTE_NAME'. Alternatively, if five field components for the key information are required, the FPM DDIC structure `FPM_S_EXT_NAVIGATION_INFO_ADD5` may be used.

OVP-Related FPM Events for Navigation

Navigation between the various OVP pages is controlled by dedicated FPM events. These FPM events can either be raised by configured button elements, by button elements that are created at runtime using the runtime APIs of the OVP, or by being simply programmed in the application.

The following FPM events are relevant for enabling navigation between OVP pages:

Event ID (to be entered in configuration editor)	Description
FPM_BACK_TO_MAIN	Leave edit page and navigate back to last main overview page (without saving the data).
FPM_CALL_FULL_SCREEN	Navigate to edit page. If the edit page marked as <i>Default</i> in configuration is not the desired one, the target edit page can be specified with the property <i>Target Page</i> in configuration.
FPM_CALL_DEFAULT_EDIT_PAGE	Navigate to UIBB default edit page specified with the property <i>Default Edit Page</i> on the UIBB.
FPM_CALL_SUBOVERVIEW_PAGE	Navigate from main overview to sub-overview page. If the sub-overview page marked as <i>Default</i> in configuration is not the desired one, the target sub-overview page can be specified with the property <i>Target Page</i> in configuration.
FPM_CHANGE_CONTENT_AREA	Navigate to a particular page specified with the property <i>Target Page</i> in configuration.
FPM_CLOSE_DIALOG	Close FPM dialog box. This event is raised automatically by the standard buttons on the FPM dialog box.
FPM_DELETE_CURRENT_OBJECT	Navigate to confirmation page. If the confirmation page marked as <i>Default</i> in configuration is not the desired one, the target confirmation page can be specified with the property <i>Target Page</i> in configuration.
FPM_DONE_AND_BACK_TO_MAIN	Leave edit page and navigate back to last main overview page (without saving the data).
FPM_LEAVE_INITIAL_SCREEN	Leave initial screen and navigate to main overview page. If the main overview page marked as <i>Default</i> in configuration is not the desired one, the target main overview page can be specified with the property <i>Target Page</i> in configuration.
FPM_OPEN_DIALOG	Open the FPM dialog box specified with the property <i>Target Page</i> in configuration and
FPM_SAVE_AND_BACK_TO_MAIN	Transactional event. Leave edit page and navigate back to last main overview page.
FPM_START	Start event of every FPM application. Navigate to initial page (if it exists) or otherwise to main overview page.

Special FPM Event Parameters

A Sub-Overview Page or an Edit Page may have the buttons *Previous* and *Next* in their page toolbar for scrolling through various items in an item list.

If the currently displayed item on such a page is the first item in the item list, *Previous* must be disabled; if it is the last item in the item list, *Next* must be disabled. The OVP runtime can control the properties of the *Previous* and *Next* buttons automatically. However, in order to do so, the OVP runtime must know two things at the point of navigation to the Sub-Overview Page or the Edit Page: the total number of items in the item list and the index of the currently selected item.

This information can be passed to the OVP via the following new FPM event parameters:

- **FPM_ITEM_LIST_TOTAL_ROWS**
Specifies the total number of items in the item list
- **FPM_ITEM_LIST_SELECTED_ROW**
Specifies the index of the currently selected item

The parameters can be set on all OVP navigation events that trigger navigation to a Sub-Overview or Edit Page. These are as follows:

- **FPM_CALL_FULL_SCREEN**
- **FPM_CALL_DEFAULT_EDIT_PAGE**
- **FPM_CALL_SUBOVERVIEW_PAGE**
- **FPM_CHANGE_CONTENT_AREA**

When using these parameters, the OVP runtime enables or disables the *Previous* and *Next* buttons depending on the index of the currently selected item. The index is automatically decreased when the FPM event `FPM_PREVIOUS_OBJECT` is triggered (which happens when the *Previous* button is pressed) and increased when the FPM event `FPM_NEXT_OBJECT` is triggered (which happens when the *Next* button is pressed).

Dynamic Changes at Runtime

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

- **OVP CNR API (similar to that of the OIF and GAF)**
Use this interface to perform changes to:
 - Toolbar buttons
 - Simple page, UIBB, page master area, or page master UIBB properties
 - External navigation menus
 - Basic application parameters
 - Page selector
- **OVP Application Configuration Controller (AppCC) API**
Use this interface to get complete access to the OVP including the page composition and layout.

Further details for both these methods are described below.

OVP CNR API

The interface `IF_FPM_CNR_OVP` provides the following methods:

Method	Description
<code>GET_CONTENT_AREAS</code>	Provides a list of all content areas currently available
<code>GET_CURRENT_CONTENT_AREA</code>	Gets the current content area
<code>CHANGE_CONTENT_AREA_RESTRICTED</code>	Changes some restricted attributes of a content area
<code>GET_SECTIONS</code>	Provides a list of all sections of a content area
<code>GET_UIBBS</code>	Provides a list of UIBBs for a content area/ section
<code>CHANGE_UIBB_RESTRICTED</code>	Changes some restricted attributes of a UIBB
<code>GET_PAGE_MASTER_AREA</code>	Gets the page master area of a content area
<code>CHANGE_PAGE_MASTER_AREA_REST</code>	Changes some restricted attributes of a page master area
<code>GET_PAGE_MASTER_UIBBS</code>	Provides a list of page master UIBBs of a content area
<code>CHANGE_PAGE_MASTER_UIBB_REST</code>	Changes some restricted attributes of a page master UIBB
<code>GET_EXTERNAL_NAVIGATIONS</code>	Gets a list of external navigation menus
<code>ADD_EXTERNAL_NAVIGATION</code>	Adds a new external navigation menu
<code>CHANGE_EXTERNAL_NAVIGATION</code>	Changes an existing external navigation menu
<code>REMOVE_EXTERNAL_NAVIGATION</code>	Removes an external navigation menu
<code>GET_TOOLBAR_ELEMENTS</code>	Gets a list of existing toolbar elements
<code>GET_TOOLBAR_BUTTON</code>	Gets attributes of a toolbar button
<code>GET_TOOLBAR_TOGGLE_BUTTON</code>	Gets attributes of a toolbar toggle button
<code>GET_TOOLBAR_BUTTON_CHOICE</code>	Gets attributes of a toolbar button-choice
<code>ADD_TOOLBAR_BUTTON</code>	Adds an existing toolbar button
<code>ADD_TOOLBAR_TOGGLE_BUTTON</code>	Adds an existing toolbar toggle button
<code>ADD_TOOLBAR_BUTTON_CHOICE</code>	Adds an existing toolbar button-choice
<code>CHANGE_TOOLBAR_BUTTON</code>	Changes an existing toolbar button
<code>CHANGE_TOOLBAR_TOGGLE_BUTTON</code>	Changes an existing toolbar toggle button
<code>CHANGE_TOOLBAR_BUTTON_CHOICE</code>	Changes an existing toolbar button-choice
<code>REMOVE_TOOLBAR_ELEMENT</code>	Removes a toolbar element
<code>GET_DEFAULT_ACTION</code>	Gets the default action of a content area
<code>ADD_DEFAULT_ACTION</code>	Adds the default action to a content area
<code>CHANGE_DEFAULT_ACTION</code>	Changes the default action of a content area
<code>REMOVE_DEFAULT_ACTION</code>	Removes the default action of a content area
<code>GET_APPLICATION_PARAMETERS</code>	Gets the application parameters
<code>CHANGE_APPLICATION_PARAMETERS</code>	Changes the application parameters
<code>GET_PAGE_SELECTOR</code>	Gets the page selector dropdown list label & entries

CHANGE_PAGE_SELECTOR	Changes the page selector dropdown list label & entries
SET_GENERIC_BUTTON_ACTION_TYPE	Sets the action type of the generic OVP buttons
SET_SIDE_PANEL_LINK	Defines the side panel link
SET_TAG_VALUE	Sets a value for a tag

Application Configuration Controller API

An application WD component or an ABAP class that should act as an application controller can implement the interface `IF_FPM_OVP_CONF_EXIT`. Its method `OVERRIDE_EVENT_OVP` passes a reference to the OVP interface of `IF_FPM_OVP` to the application which provides full runtime access to the OVP floorplan, including the page composition and layout. The interface `IF_FPM_OVP` provides the following methods:

Method	Description
GET_CONTENT_AREAS	Provides a list of all content areas currently available
GET_CURRENT_CONTENT_AREA	Gets the current content area
ADD_CONTENT_AREA	Adds a new content area to the application
CHANGE_CONTENT_AREA	Changes an existing content area
REMOVE_CONTENT_AREA	Removes an existing content area
GET_SECTIONS	Provide a list of all sections of a content area
ADD_SECTION	Adds a new section to a content area
CHANGE_SECTION	Changes an existing content area section
REMOVE_SECTION	Removes an existing content area section
GET_UIBBS	Provides a list of UIBBs for a content area
ADD_UIBB	Adds a UIBB to an existing content area/ section
CHANGE_UIBB	Changes an existing UIBB
REMOVE_UIBB	Removes an existing UIBB
GET_PAGE_MASTER_AREA	Gets attributes of the page master area
ADD_PAGE_MASTER_AREA	Adds a new page master area to a content area
CHANGE_PAGE_MASTER_AREA	Changes an existing page master area
REMOVE_PAGE_MASTER_AREA	Removes an existing page master area from a content area
GET_PAGE_MASTER_UIBBS	Provides a list of page master UIBBs for a content area
ADD_PAGE_MASTER_UIBB	Adds a new page master UIBB to a content area
CHANGE_PAGE_MASTER_UIBB	Changes an existing page master UIBB to a content area
REMOVE_PAGE_MASTER_UIBB	Removes an existing page master UIBB from a content area
GET_EXTERNAL_NAVIGATIONS	Gets a list of external navigation menus
ADD_EXTERNAL_NAVIGATION	Adds a new external navigation menu
CHANGE_EXTERNAL_NAVIGATION	Changes an existing external navigation menu
REMOVE_EXTERNAL_NAVIGATION	Removes an external navigation menu

GET_TOOLBAR_ELEMENTS	Gets a list of existing toolbar elements
GET_TOOLBAR_BUTTON	Gets attributes of a toolbar button
GET_TOOLBAR_TOGGLE_BUTTON	Gets attributes of a toolbar toggle button
GET_TOOLBAR_BUTTON_CHOICE	Gets attributes of a toolbar button-choice
ADD_TOOLBAR_BUTTON	Adds an existing toolbar button
ADD_TOOLBAR_TOGGLE_BUTTON	Adds an existing toolbar toggle button
ADD_TOOLBAR_BUTTON_CHOICE	Adds an existing toolbar button-choice
CHANGE_TOOLBAR_BUTTON	Changes an existing toolbar button
CHANGE_TOOLBAR_TOGGLE_BUTTON	Changes an existing toolbar toggle button
CHANGE_TOOLBAR_BUTTON_CHOICE	Changes an existing toolbar button choice
REMOVE_TOOLBAR_ELEMENT	Removes a toolbar element
GET_DEFAULT_ACTION	Gets the default action of a content area
ADD_DEFAULT_ACTION	Adds the default action to a content area
CHANGE_DEFAULT_ACTION	Changes the default action of a content area
REMOVE_DEFAULT_ACTION	Removes the default action of a content area
GET_APPLICATION_PARAMETERS	Gets the application parameters
CHANGE_APPLICATION_PARAMETERS	Changes the application parameters
GET_PAGE_SELECTOR	Gets the page selector dropdown list label & entries
CHANGE_PAGE_SELECTOR	Changes the page selector dropdown list label & entries
SET_GENERIC_BUTTON_ACTION_TYPE	Sets the action type of the generic OVP buttons
GET_EVENT	Gets the current FPM event
SET_EVENT	Sets or changes the current FPM event
IS_EVENT_CANCELLED	Returns whether the current FPM event is cancelled
CANCEL_EVENT	Cancels event processing
IF_FPM_WIRE_MODEL~ADD_WIRE	Adds a wire between two existing UIBBs (parameter IV_VARIANT_ID is ignored in OVP)
IF_FPM_WIRE_MODEL~REMOVE_WIRE	Removes a wire between two UIBBs (parameter IV_VARIANT_ID is ignored in OVP)
IF_FPM_WIRE_MODEL~GET_WIRES	Gets a list of all existing wires between UIBBs (parameter IV_VARIANT_ID is ignored in OVP)

Setting a Default ALV View for a Freestyle UIBB

When you embed freestyle UIBBs 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. Open FLUID and add this freestyle UIBB to your floorplan.
3. In the object schema tab, choose the row containing the UIBB to display its attributes.
4. The table `Configurable ALV Tables`, is displayed.
5. 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:

```
method wddoinit .
    data lo_cmp_usage type ref to if_wd_component_usage.
    data lo_fpm type ref to cl_fpm.
    data lt_conf_comp_usage type fpm_t_uibb_conf_comp_usage.
    data ls_conf_comp_usage type fpm_s_uibb_conf_comp_usage.
    data lv_component_name type wdy_component_name.
    data ls_config_key type wdy_config_key.

    * get name of freestyle UIBB
    lv_component_name = wd_this->wd_get_api()->get_component_info()->get_name().

    * get tree of configurable ALV usages
    lo_fpm ?= cl_fpm_factory=>get_instance().
    lt_conf_comp_usage =
        lo_fpm->if_fpm~mo_conf_comp_usage-
        >get_conf_comp_usage_tree( iv_component_name = lv_component_name ).

    * pick one configuration key from tree
    read table lt_conf_comp_usage into ls_conf_comp_usage with key child_component_usage_name
    = 'ALV'.
    move-corresponding ls_conf_comp_usage to ls_config_key.

    * pass configuration key to ALV usage
    lo_cmp_usage = wd_this->wd_cpuse_alv().
    if lo_cmp_usage->has_active_component() is initial.
        lo_cmp_usage->create_component( configuration_id = ls_config_key ).
    endif.
```

```
endmethod.
```

For more information on ALV views, see *SAP List Viewer in Web Dynpro for ABAP* in the SAP NetWeaver Library.

FPM Dialog Boxes

FPM provides a framework for supporting dialog boxes. Dialog boxes are supported in all floorplans.

Structure

A dialog box is essentially a page in a floorplan, containing at least one UIBB (freestyle UIBBs or GUIBBs).

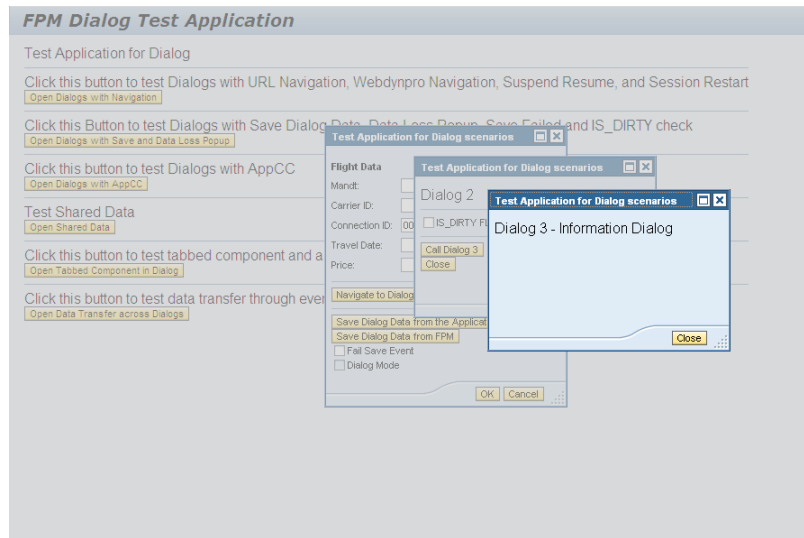
The CNR and IDR regions are not available in the FPM dialog boxes.

A dialog box is identified by a unique Dialog Box ID.

FPM supports up to three levels of dialog boxes (opening one dialog box from within another dialog box).

- Dialog boxes on levels 1 and 2 display any of the following button sets:
 - *Ok* and *Cancel*
 - *Close*
 - No button
- Dialog boxes on level 3 display only the *Close* button. It is not possible to open another FPM dialog box from a dialog box on level 3.

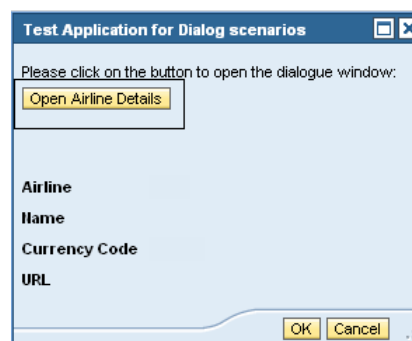
Example of the 3 Levels of FPM Dialog Boxes



Features

Note the following features of FPM dialog boxes:

- Provision of two buttons *OK* and *Cancel* with default actions
The dialog box buttons are handled by FPM itself.
- Support for 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).



- 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.
- Configurable Layout and Button Texts

FPM dialog boxes can have one or more UIBBs and their layout can be configured. For more information see [Configuring FPM Dialog Boxes](#).

- **Error Page**
The FPM error page can be triggered in FPM dialog boxes. For more information see [FPM Error page in Dialog Boxes](#).
- **Navigation**
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.
- **Data-Loss Handling**
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.
- **Events**
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](#)

- **Data Transfer**
FPM dialog boxes support data transfer across dialog boxes through event data.
- **Enablement of Dialog Box Buttons**
FPM dialog boxes support enabling and disabling of buttons (except the *Cancel* Button) at runtime.

Creating and Configuring an FPM Dialog Box

You create a dialog box in the same way that you create other pages in a floorplan component configuration.

The following procedure assumes you have already created your UIBB component configurations.

Create and configure FPM dialog boxes in the FPM configuration editor, FLUID, as follows:

1. On the *Navigation* tab, add a new page of type *Dialog Box*.
2. Choose the *Attributes* button on the main toolbar to display the attributes of this new page.

3. Enter an ID and name for the dialog box page.
4. On the <Floorplan> Schema tab, choose *Add UIBB*.
5. Choose the type of UIBB which you want to add to your dialog box and enter the following information:
 - Component
 - Window Name
 - Configuration Name
6. Add further UIBBs if necessary using the *Add UIBB* button.
7. To make changes to any of the following attributes of the dialog box, choose the dialog box on the *Navigation* page and edit the relevant attributes:
 - Title
 - Layout of UIBBs
 - Size
 - Tooltip texts for the various buttons in the dialog box
 - Alternative texts for the various buttons in the dialog box

Note that you can create multiple dialog boxes in a single application.

Triggering Dialog Boxes from a Toolbar Button

It is also possible to trigger dialog boxes by choosing a button in the toolbar. To do this, first create a button in the toolbar and then provide an event ID for the button. Choose the *Toolbar* element on the *Toolbar Schema* tab. In the *Attributes* panel, 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 floorplan 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:

Method Name	Parameters	Description
<code>OPEN_DIALOG_BOX</code>	<code>IV_DIALOG_BOX_ID</code>	This method is used in the application to open the

	IS_DIALOG_BOX_PROPERTIES (optional)	dialog box.
	IO_EVENT_DATA (optional)	

- 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.

Attribute Name	Type	Description
MV_IS_DIALOG_MODE (Read Only)	FPM_DIALOG_STATE	Indicates whether application is in dialog box or in main screen.

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 Ir_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( Ir_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.

Lifetime/Visibility	Message Behavior
Automatic	<p>FPM takes care of the visibility based on the UI guidelines. The <i>Automatic</i> messages in an FPM dialog box are cleared after every roundtrip or if a new message is raised on a dialog box.</p> <p>On Dialog Close:</p> <ul style="list-style-type: none"> – <i>Automatic</i> messages are not carried forward to the parent, or the previous level dialog box. <p>On Dialog Open:</p> <ul style="list-style-type: none"> – <i>Automatic</i> messages are cleared on the parent window, or on parent dialog box from where the dialog box is opened.
View	<p>The message is visible as long as the view to which the message is bound is available,</p> <p>On Dialog Close</p> <ul style="list-style-type: none"> -Messages with <code>MANU_VIEW</code> lifetime are not carried to the parent or the parent dialog box. <p>On Dialog Open:</p> <p>Messages with <code>MANU_VIEW</code> lifetime reappear on the dialog box when it is reopened until it is cleared manually.</p> <p>These messages are not carried forward from parent window or the parent dialog box on the dialog box.</p>
Controller	<p>The message is visible as long as the controller that has raised the message is available.</p> <p>On Dialog Close:</p> <p>Messages with <code>MANU_CONT</code> lifetime would be passed to the parent if it is raised through the same controller.</p> <p>On Dialog Open:</p> <ul style="list-style-type: none"> -Messages with <code>MANU_CONT</code> lifetime would be passed to the dialog box from the parent window or the parent dialog if it is raised through the same controller.
Application	<p>The message is permanently displayed, in both the dialog box and the parent window throughout the application until it is manually cleared by the application</p>

	developer.
Auto Pop-up	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.

Enabling/Disabling Dialog Box Buttons at Runtime

Dialog box buttons can be enabled or disabled at runtime. The *Cancel* button cannot be disabled or enabled. An API `IF_FPM->SET_DIALOG_BUTTON_STATUS` is provided to set the status of the dialog box button.

Sample Code to Set the Status of the Dialog Box button.

```
data: lv_button type wdr_popup_button.
lv_button = 7.

wd_comp_controller->lr_fpm->set_dialog_button_status( iv_button = lv_button
                                                    iv_status = abap_true ).
```

Note that `lv_button` is the value of the type of the button.

FAQs on FPM Dialog Boxes

1. 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.
2. 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.
3. 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.
4. 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.

5. How does transient behavior work 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.
6. How can I find out 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 components (WD component: `FPM_FORM_UIBB_GL2` and `FPM_FORM_UIBB`)
- List components (WD component: `FPM_LIST_UIBB_ATS` and `FPM_LIST_UIBB`)
- Hierarchical List (Tree) component (WD component: `FPM_TREE_UIBB`)
- Tabbed component (WD component: `FPM_TABBED_UIBB`)
- Search component (WD component: `FPM_SEARCH_UIBB`)
- Launchpad component (WD component: `FPM_LAUNCHPAD_UIBB`)
- POWL component (WD component: `FPM_POWL_UIBB`)
- Composite component (WD component: `FPM_COMPOSITE_UIBB`)



FPM also provides other another type of special UIBB known as a Reuse UIBB (RUIBB) which provides not only the layout of the UI but also the business logic. Details are provided in the section Reuse UIBBs.

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 so on, 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

Context Menus in GUIBBs

By implementing the interface `IF_FPM_GUIBB_CTXT_MENU`, Form, List and Tree feeder classes can provide context menus.

This interface offers an additional event (`PROCESS_CTXT_MENU`) which is called after the user launches the context menu and before the context menu is displayed. This event corresponds to the Web Dynpro event `WDDOONCONTEXTMENU`.

Due to the pure dynamic nature of these context menus, the GUIBB context menu features are restricted compared to what Web Dynpro itself offers: Menu checkboxes and radio buttons cannot be provided this way.

Methods of `IF_FPM_GUIBB_CTXT_MENU` Interface

PROCESS_CTXT_MENU:	
Allows the feeder to provide a context menu.	
Parameter	Description
<code>IV_NAME</code>	Type <code>NAME_KOMP</code> . Contains the field name if the context menu is launched from a place which can be related to a field from the field catalog. If this is not possible (for example, when the context menu is launched from the list title or from a form's group title) then <code>IV_NAME</code> is initial.
<code>I_VALUE</code>	Type <code>DATA</code> . In form UIBBs it contains the field's value, in list UIBBs the complete row. This parameter is initial when the context menu is launched from a place which is not related to this data (for example, when launching the context menu in a list from a column header <code>IV_NAME</code> will be filled, but <code>I_VALUE</code> will be empty).
<code>IO_CTXT_MENU</code>	Type <code>IF_FPM_CTXT_MENU</code> . This interface is explained in detail below.

Methods of `IF_FPM_CTXT_MENU` Interface

GET_DETAIL:
Gets the detail information for this context menu

Parameter	Description
EV_ID	Type FPM_CTXT_MENU_ID. Contains the unique ID of the context menu.
EV_TITLE	Type FPM_CTXT_MENU_TITLE. Contains the title of the context menu. This title is only visible when the context menu is used as submenu.
EV_IMAGE_SOURCE	Type FPMGB_IMAGE_SRC. Provides the source of the image of the menu. This image is only displayed when the context menu is used as a submenu.
ET_ITEM	Type IF_FPM_CTXT_MENU=> TY_T_CTXT_MENU_ITEM. Provides a list of all menu items.

GET_SUBMENU: Returns a submenu	
Parameter	Description
IV_ID	Type FPM_CTXT_MENU_ID. Specifies the ID of the context sub menu.
RO_SUBMENU	Type IF_FPM_CTXT_MENU. Represents the submenu.

REMOVE_ITEM: Removes the specified item from a context menu	
Parameter	Description
IV_ID	Type FPM_CTXT_MENU_ID. Specifies the ID of the item to be removed.

ADD_ACTIONITEM: Adds a new Action Item to the menu	
Parameter	Description
IV_TEXT	The text of the new menu entry.
IV_FPM_EVENT_ID	The FPM Event ID which will be raised when the user selects this action item.
IO_FPM_EVENT	The FPM Event to be raised when the user selects this action item. Either IV_FPM_EVENT_ID or IO_FPM_EVENT have to be provided. If both are provided IV_FPM_EVENT_ID is ignored.
IV_IMAGE_SOURCE	Provides the image source for the menu entry (optional).
IV_NEEDS_MORE_INFO	Should be set when executing the action 'additional information is required'. Another three points are listed after the text to display this information.
IV_INDEX	Allows specification of the position of the new entry (optional: if left initial, entry is appended to the end).

ADD_SUBMENU: Creates and adds a new Submenu to the Context menu	
Parameter	Description
IV_TITLE	The title of the new menu entry.

IV_IMAGE_SOURCE	Provides the image source for the new submenu (optional).
IV_INDEX	Allows specification of the position of the new entry (optional: if left initial, entry is appended to the end).
RO_SUBMENU	The new submenu.

ADD_SEPARATOR: Adds a new separator to the Context menu	
Parameter	Description
IV_INDEX	Allows specification of the position of the separator (optional: if left initial, entry is appended to the end).

Form Component (GUIBB FORM GL2)

This is a generic design template for displaying data in a form that is implemented using the Web Dynpro component `FPM_FORM_UIBB_GL2`.

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_GL2`.

Structure

A form is comprised of various sub objects:

- STANDARD_ELEMENT
Elements are descriptor/field combinations that can be configured for the display type of the field or descriptors.
- TOOLBAR (Button Row)
Contains buttons that can have actions assigned to them and can be executed in the form.
- BUTTON
A single button that can have actions assigned to it and can be executed in the form
- GROUP
A group consists of group elements, and group toolbars. You can enter a separate name for each group.
- GROUP_ELEMENT
Group element with the same structure as the standard element
- GROUP_BUTTON_ROW
Button Row with the same structure as the Button Row (Toolbar)

- GROUP BUTTON

Button element with the same structure as the Button

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_GL2`.

Integration

You can configure a form component using the configuration editor for Floorplan Manager, FLUID.

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).	
Parameter	Description
EO_FIELD_CATALOG	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 <pre>eo_field_catalog ?= CL_ABAP_STRUCTDESCR=>describe_by_name(<name>)</pre>  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 <code>Label_by_DDIC</code> , <code>LABEL_REF</code>
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 <code>REUSE_ALV_GRID_DISPLAY</code>) 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.

Parameter	Description
RT_PARAMETER_DESCR	Is returned from this method. It describes which parameter is possible. In Field TYPE, the DDIC type needs to be entered.

INITIALIZE:

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

Parameter	Description
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 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.

Parameter	Description
IT_CHANGE_LOG	Lists all changes made by the user.
IS_DATA	Is a structure containing the changed data

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.

Parameter	Description
IO_EVENT	The FPM event which is to be processed
EV_RESULT	The result of the event processing. There are 3 possible values: <pre> 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 </pre>

ET_MESSAGES	A list of messages which shall be displayed in the message region.
-------------	--

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.	
Parameter	Description
IO_EVENT	The FPM event which is to be processed.
IT_SELECTED_FIELDS	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.
ET_MESSAGES	A list of messages which shall be displayed in the message area.
EV_DATA_CHANGED	For performance reasons, the GUIBB adjusts the data in the form only if the data has been changed. To indicate this, set this flag whenever you change the data to be displayed within this feeder.
EV_FIELD_USAGE_CHANGED	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.
EV_ACTION_USAGE_CHANGD	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.
CS_DATA	The form data to be changed.
CT_FIELD_USAGE	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.
CT_ACTION_USAGE	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.

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.	
Parameter	Description

IO_LAYOUT_CONFIG	Of type IF_FPM_GUIBB_FORM_CONFIG: This object provides the API to create a default configuration for the "old" Form, is not used in FPM_FORM_UIBB_GL2
IO_LAYOUT_CONFIG_GL2	Of type IF_FPM_GUIBB_FORM_CFG_GL2: This object 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.	
Parameter	Description
IO_LAYOUT_CONFIG	Of type IF_FPM_GUIBB_FORM_CONFIG: This object provides the API to read the configuration to be saved for the "old" Form, is not used in FPM_FORM_UIBB_GL2.
ET_MESSAGES	A list of messages which shall be displayed in the message region.
IO_LAYOUT_CONFIG_GL2	Of type IF_FPM_GUIBB_FORM_CFG_GL2: This object provides the API to read the configuration to be saved

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

- **8/1 Layout**
The Form has 8 columns with names A to H. The element can be arranged in this layout
- **16/1 Layout**
The form has 16 columns with names A to P. The element can be arranged on the whole form.
- **16/2 layout**
The form has 16 columns which are separated into panels. Panel 1 column A to H, Panel 2 column I to P. Elements can be put in the first or in the second panel. No overlapping possible

Form Component (GUIBB FORM)

Note that this form component has been superseded by a new form component, `FPM_FORM_UIBB_GL2`.

This is 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 configure a form component using the configuration editor for Floorplan Manager, FLUID.

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).

Parameter	Description
EO_FIELD_CATALOG	<p>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</p> <pre>eo_field_catalog ?= CL_ABAP_STRUCTDESCR=>describe_by_name(<name>)</pre> <p> The form GUIBB supports only flat structures. When using deep structures, only the highest level fields are available.</p>
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.

Parameter	Description
RT_PARAMETER_DESCR	Is returned from this method. It describes which parameter is possible. In Field TYPE, the DDIC type needs to be entered.

INITIALIZE:

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

Parameter	Description
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 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.	
Parameter	Description
IT_CHANGE_LOG	Lists all changes made by the user.
IS_DATA	Is a structure containing the changed data

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.	
Parameter	Description
IO_EVENT	The FPM event which is to be processed
EV_RESULT	The result of the event processing. There are 3 possible values: <pre>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</pre>
ET_MESSAGES	A list of messages which shall be displayed in the message region.

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.	
Parameter	Description
IO_EVENT	The FPM event which is to be processed.
IT_SELECTED_FIELDS	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.
ET_MESSAGES	A list of messages which shall be displayed in the message area.
EV_DATA_CHANGED	For performance reasons, the GUIBB adjusts the data in the form only if the data has been changed. To indicate this, set this flag whenever you change the data to be displayed within this feeder.
EV_FIELD_USAGE_CHANGED	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.
EV_ACTION_USAGE_CHANGED	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.
CS_DATA	The form data to be changed.
CT_FIELD_USAGE	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 <code>FIXED_VALUES_CHANGED</code> for this field.
CT_ACTION_USAGE	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.

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

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.

List ATS Component (GUIBB List ATS)

The List ATS (ABAP Table Services) component is a GUIBB that is used to render lists according to the latest UI guidelines. It provides a wide range of features, such as sorting, filtering and grouping, and contains numerous personalization possibilities.

All new applications requiring a list component should use the LIST ATS component.

To utilize the features of the LIST ATS component, you must create a feeder class and a configuration for WD component `FPM_LIST_UIBB_ATS`.

Feeder Class

The application is responsible for writing and implementing the feeder class.

The feeder class must implement the interface `IF_FPM_GUIBB_LIST`; the interface provides meta data relating to columns, provides the data at runtime that shall appear in the list and contains methods that participate in the FPM event loop.

Feeder Interface

The feeder interface `IF_FPM_GUIBB_LIST` contains the following methods:

Method Name	Description
GET_DEFINITION	<p>Called once at startup. Tells the List UIBB which columns and actions are available. This is done using parameters <code>EO_FIELD_CATALOG</code> and <code>ET_ACTION_DEFINITION</code> for columns and toolbar items respectively. The columns and actions defined here do not automatically appear at runtime. They appear in the FPM configuration editor as a list of available columns and actions to choose from. Only when they have been configured will they appear at runtime.</p> <p>It is also possible to provide additional information about columns such as texts, tooltips, F4 helps, and formatting information. This is done using parameter <code>ET_FIELD_DESCRIPTION</code>.</p>
FLUSH	<p>Called as part of the FPM event loop. It is the first method of the event loop. Using parameter <code>IT_CHANGE_LOG</code> it tells the feeder class what changes have been made by the user. The changes, indicated by <code>IT_CHANGE_LOG</code>, must not be copied into the source data table (parameter <code>IT_DATA</code>); accessing the source table is not allowed in this FPM event hook.</p>
PROCESS_EVENT	<p>Also part of the FPM event loop; called directly after the <code>FLUSH</code> method. Use this method to check the Event ID and, if necessary, to carry out Event ID-specific processing. Besides the Event ID, the feeder class also gets information about the current selected rows and whether the event has been raised by this List UIBB or by some other UIBB on the screen. For more information, use the parameter <code>IO_UI_INFO</code> (interface <code>IF_FPM_LIST_ATS_UI_INFO</code>).</p> <p>Additionally, this method can be used to pass messages using the parameter <code>ET_MESSAGES</code>.</p>
GET_DATA	<p>Also part of the FPM event loop; called directly after method <code>PROCESS_EVENT</code>. It corresponds to the general UIBB method <code>PROCESS_BEFORE_OUTPUT</code>. The main purpose of this method is to exchange data between the feeder class and the List ATS UIBB (for details, refer to the heading 'Data Exchange' in this document. It is also used to change attributes of the table (parameters <code>CS_ADDITIONAL_INFO</code> and <code>CV_FIRST_VISIBLE_ROW</code>), attributes of the columns (parameter <code>CT_FIELD_USAGE</code>), and attributes of the toolbar elements (parameter <code>CT_ACTION_USAGE</code>). With <code>CV_FIRST_VISIBLE_ROW</code> it is possible to set the first visible row of the list.</p>
GET_PARAMETER_LIST	<p>Called once at startup. It is not mandatory to use this method. It is used to fill the feeder class parameters with values. The feeder class defines parameters and at design time one can assign values to those parameters in the FPM configuration editor with this method. This is used, for example, in more complex (generic) feeder classes.</p>
INITIALIZE	<p>Called once at startup. It is not mandatory to use this method. At runtime this method tells what parameters have been defined by the feeder class (<code>GET_PARAMETER_LIST</code>) and what values have been assigned to them in the configuration. This is used, for example, in more complex (generic) feeder classes.</p>
GET_DEFAULT_CONFIG	<p>Called once at startup. It is not mandatory to use this method. It has two purposes. When creating a new configuration, use this method to</p>

	generate a list of columns and toolbar elements that could be configured. This method is also called at runtime when no columns at all are configured. In this case, it is used to generate a configuration.
CHECK_CONFIG	Called at design time only. It is not mandatory to use this method. Use this method to execute application-specific checks on the configuration.

Configuration

General

As with other GUIBBs, you must create a configuration for the List ATS component; in this case, the Web Dynpro component is based on FPM_LIST_UIBB_ATS. The feeder class is linked to the configuration in the FPM configuration editor. If the feeder class has defined parameters using the method `get_parameter_list`, the configuration editor allows you to assign specific values to them. This is relevant mostly for complex (generic) feeder classes. Other parts of the configuration comprise the columns, the toolbar and the *General Settings*. It is also possible to define a configuration using the feeder class method `get_default_config`.

Columns

Only columns that are configured in the FPM configuration editor will appear at runtime. You can choose from columns that have been defined by the feeder class method `get_definition`, parameter `eo_field_catalog`. In the attributes section of the FPM configuration editor, you can determine various attributes of each configured column.

Toolbar

Only actions that are configured in the FPM configuration editor will appear at runtime. You can choose from actions that have defined by the feeder class method `get_definition`, parameter `et_action_definition`. In the attributes section of the FPM configuration editor, you can determine various attributes of each configured action.

Data Exchange

General

The data exchange happens in the feeder methods `FLUSH` and `GET_DATA`. Data entered by the user is passed using method `FLUSH` to the back end; the data from the back end is passed to the front end using method `get_data`. The following are the different data exchange modes:

- Compatibility
- Key
- Stable line

It is recommended to use either the key or stable line modes to ensure that the data exchange complies with the latest UI guidelines.

A detailed description for managing data exchange is found in the [Edit Scenarios for the New List ATS UIBB](#) (See Appendix).

Compatibility Mode

This mode exists to support feeder classes that have been coded for the former (old) List UIBB. It uses two parameters: `ct_data` and `ev_data_changed`. When the parameter `ev_data_changed` is set to `TRUE`, the List ATS GUIBB takes the data from parameter `ct_data`. There is no possibility to exchange only part of the data, the whole table is exchanged.

Key Mode

In this mode, the rows of the data table have a primary key. Arbitrary operations with data table are possible as long as the uniqueness of the keys is guaranteed.

This mode uses the parameters `ct_data`, `cs_data_changed` and `eo_itab_change_log`. The parameter `eo_itab_change_log` is a handle to the interface `if_salv_itab_change_log` which describes how the data in the data table was changed. This interface has the following methods:

Method Name	Description
GET_INDEX_MAP	Gets the indices of deleted rows (<i>before</i> image), the indexes of inserted rows (<i>after</i> image), the mapping of rows between <i>before</i> and <i>after</i> image and the indexes of rows (<i>before</i> image) which must be moved to an insert position
GET_COLUMNS_MODIFIED	Gets the names of columns for which values have been changed
GET_LINES_MODIFIED	Gets the indices of lines (<i>after</i> image) for which values have been changed
DATA_IS_NEW	The value is <code>TRUE</code> if data is completely new and there is no connection between before and after image

If the parameter `ev_data_changed` is set to `TRUE`, the List ATS UIBB distinguishes between the following cases:

<code>eo_itab-change_log</code>	Purpose
Initial	Data table is changed but the changes are not recorded → Data is new
Not Initial <code>DATA_IS_NEW = TRUE</code>	Data table is changed but the change log is not complete → Data is new
Not Initial <code>DATA_IS_NEW = FALSE</code>	Data table is changed and the change log is complete

In the 'data is new' cases, no connection between the *before* and *after* images can be constructed by the List ATS UIBB. Therefore, features such as sorting and filtering are executed automatically. These services are not to be executed automatically when only part of the data is changed.

There is a standard implementation for Key Mode for the creation of a change log, the class `cl_salv_itab_editor_key_mode`. This class has the following methods:

Method Name	Description
LOG_NEW_DATA	Resets the change log (<code>DATA_IS_NEW</code>); returns <code>TRUE</code> value
START_RECORDING	Starts recording of a change log; an existing change log is deleted
KEY_CHANGED	Notifies that a key was changed
STOP_RECORDING	Stops recording of a change log

MOVE_TO_INSERT_POSITION	Marks a line for moving to an insert position
-------------------------	---

For a detailed description of the Key Mode see 'Guideline for Edit Scenarios for New List ATS UIBB' (See Appendix).

Stable Line Mode

In this mode, the order of lines in the data table must not be changed; in particular, the table must not be sorted. Insertions and deletions of lines are allowed.

This mode uses the parameters `ct_data`, `cs_data_changed` and `eo_itab_change_log`. For a description of these parameters, refer to the 'Key Mode' section above.

There is a standard implementation for the Stable Line Mode for the creation of a change log, the class `cl_salv_itab_editor_line_mode`. This class provides methods for table operations such as insert and append instead of direct usage of ABAP statements. This class has the following methods:

Method Name	Description
START_RECORDING	Starts recording of a change log; an existing change log is deleted
SET_NEW_DATA	Gets new data and resets the change log (<code>DATA_IS_NEW</code> returns TRUE value)
APPEND ...	Substitutes for ABAP statement APPEND (3 variants)
CLEAR_TABLE	Substitutes for ABAP statement CLEAR
COLLECT_LINE	Substitutes for ABAP statement COLLECT
DELETE ...	Substitutes for ABAP statement DELETE (2 variants)
INSERT ...	Substitutes for ABAP statement INSERT (3 variants)
MODIFY_LINE	Substitutes for ABAP statement MODIFY
MOVE_TO_INSERT_POSITION	Marks a line for moving to an insert position

For a detailed description of the Stable Line Modes see 'Guideline for Edit Scenarios for New List ATS UIBB' (See Appendix).

Actions

General

Actions may be set in three different places in the List ATS UIBB: the toolbar, inside a normal cell, and inside the action column (known as one-click actions). An action in this sense is a UI element, such as a button, a link to action or an input field, that is capable of raising an FPM event.

Actions in the Toolbar

Firstly, the actions must be defined by the feeder. This is done in the feeder method `get_definition`, using parameter `et_action_definition`. Additionally, the actions must be added to the *Toolbar Schema* in the FPM configuration editor to ensure that they appear at runtime. At runtime, the actions will raise the FPM event ID that has been specified using the attribute ID in parameter `et_action_definition`. It is also possible that the action is not rendered in the toolbar of the List ATS UIBB but in the toolbar of the OVP

assignment block where the list is embedded. This is done by setting the flag *Exposable* of parameter `et_action_definition` to `TRUE`. It is possible to change properties such as *Visibility* or *Enabled* of the toolbar elements in the feeder method `get_data` using parameter `ct_action_usage`. It is possible to assign a particular action (corresponding to one entry in `et_action_definition`) to more than one UI element in the toolbar. However, when doing so, be aware of the following restriction: In feeder method `GET_DATA` it is possible to change properties of the UI elements in the toolbar via parameter `CT_ACTION_USAGE`. For actions that have been used more than once, it is only possible to change the *Visibility* and *Enable/Disable* properties.

Actions inside Cells

Actions can also be set inside cells. There are several display types capable of raising an event, such as input field, button, checkbox, and link to action. This is done in the configuration editor.

The standard behavior of all cell actions is to raise the same event, that is, `IF_FPM_GUIBB_LIST=>GC_GUIBB_LIST_ON_CELL_ACTION`. Using the event parameter `IF_FPM_GUIBB_LIST=>GC_EVENT_PAR_COLUMN_NAME`, you can determine which cell has raised which action. However, the standard behavior can be changed in the FPM configuration editor. In addition to the standard behavior there is the possibility to un-assign an action (no action at all) or to assign a specific action (in which case the action must be specified by the feeder class in method `get_definition`, parameter `et_action_definition`).

One-Click Actions in the Action Column

One-click actions are actions that are specific to a certain table row and are executed with a single mouse click, such as deleting a row, editing a row or setting the status of a row. One-click actions are of display type *Link to Action* and are rendered in one single column known as the action column. In accordance with current UI guidelines, there is only one action column and it is the first column in the table.

To use one-click actions, complete the following steps:

1. Define actions in the feeder class in the feeder method `GET_DEFINITION` using parameter `ET_ROW_ACTIONS`.
2. Edit the configuration in the FPM configuration editor. If the feeder class has defined one-click actions, the *Repositories* panel displays a special column, `FPM_ROW_ACTIONS_COLUMN` which you can add to the object schema.
3. Add one-click actions to the action column. To do this, select the action column in the object schema and then add single actions using the *Attributes* section of the FPM configuration editor.

At runtime it is possible to disable or hide single one-click actions per row. You need to define a new technical column of type *BOOLEAN* in the feeder class that controls the enablement or visibility of the one-click action. To do this, add the column in feeder class method `get_definition`, parameter `EO_FIELD_CATALOG` and mark it as *Technical* using parameter `ET_FIELD_DESCRIPTION` and attribute `technical_field`. The ID of that column must be put into the one-click action definition (feeder method `get_definition`, parameter `ET_ROW_ACTIONS`, attribute `visible_ref` or `enabled_ref`). Fill the content of that technical column according to your needs in feeder method `GET_DATA`.

Features

Filtering

The feeder class must indicate whether a column can be filtered or not. This is done using method `get_definition`, parameter `et_field_description`, attribute `allow_filter`. If the feeder class allows filtering on columns, it is possible to switch the filtering on or off in the configuration editor.

Sorting

The feeder class must indicate whether a column can be sorted or not. This is done using method `get_definition`, parameter `et_field_description`, attribute `allow_sort`. If the feeder class allows sorting on columns, it is possible to switch the sorting on or off in the configuration editor.

Grouping

This feature allows the user to group entries of columns by header. It is not controlled by the feeder class. It can be switched on or off only in the FPM configuration editor.

F4/Input Help

The List ATS UIBB supports the following types of F4 help:

- **DDIC**
If a column uses a data element that has a DDIC F4 help assigned to it, the F4 help is used automatically. It is possible to overwrite this and to use another DDIC F4 help. This is done in feeder class method `get_definition`, parameter `et_field_description`, attribute `DDIC_SHLP_NAME`.

- **OVS**
To use this type of F4 help you must implement the interface `IF_FPM_GUIBB_OVS`. This can be done in the feeder class itself or in another class. If you implement it in another class, the List ATS UIBB instantiates the class at runtime. It is also necessary to put the name of the class that implements the interface into the attribute `OVS_NAME` of parameter `et_field_description` in feeder class method `get_definition`.

The methods of the interface `if_fpm_gui_bb_ovs` must be implemented according to the documentation for the online value help (OVS) concept. You can pass an FPM event (`eo_fpm_event`) in method `HANDLE_PHASE_3`. This is optional. If it is passed, this event is raised directly after the end of the value help processing. If it is not passed, no roundtrip after the value help processing takes place.

- **Freestyle WD ABAP**
To use this type of F4 help, you need to create a Web Dynpro component that implements the Web Dynpro interface `IWD_VALUE_HELP` (for details, refer to the corresponding Web Dynpro ABAP documentation). You must put the name of that component into the attribute `WD_VALUE_HELP` of parameter `et_field_description` in feeder class method `get_definition`.

Export to Spreadsheet

At runtime the user has the possibility to export the content of the list into a spreadsheet. This feature is activated in the FPM configuration editor under *General Settings*.

Drag and Drop

You can enable drag-and-drop of rows during runtime. To do this, the feeder class must define the corresponding attributes. This is done in method `get_definition`, parameter `ET_DND_DEFINITION`. There, you can determine whether a list is to be the drag source, drop target or both (attribute *Type*), whether drag-and-drop occurs only within the same list or between several lists (attribute *Scope*). If drag-and-drop between several lists is defined, it is possible to control which lists are used as a drag source and which lists are used as a drop target (attribute *Tags*). When two lists have the same tags, you can drag and drop between them. You can assign more than one tag to a list in the configuration editor by separating them with semi colons. Drag-and-drop attributes can be changed at runtime (feeder class method `get_data`, parameter `CT_DND_ATTRIBUTES` and `EV_DND_ATTR_CHANGED`).

When drag-and-drop actually occurs at runtime, the following FPM event is raised after a row(s) has /have been dropped: `IF_FPM_GUIBB_LIST=>GC_GUIBB_LIST_ON_DROP` (the event is only raised by the List ATS UIBB when the user has dropped the rows).

You can check this using parameter `IV_RAISED_BY_OWN_UI` in feeder class methods `get_data` or `process_event`. Data regarding the row that has been dragged is passed as an event parameter of the event `IF_FPM_GUIBB_LIST=>GC_GUIBB_LIST_ON_DROP`. The row that has been dragged is not automatically inserted into the target list. That must be done by the feeder class itself. To comply with current UX guidelines, you must use a special method to insert the data. This method, `move_to_insert_position`, is part of the change log API. Refer to the heading *Data Exchange* for further details.

When grouping is enabled, it might happen that the drop position lies between two group header rows (for example, if at least one group is collapsed). In this case, the insert position in the drop event is initialized. The application must decide if the drop should still be executed. Inserted rows will be appended to the end of the table if no other insert position has been explicitly specified in `PROCESS_EVENT`.

Personalization

The List ATS UIBB has a UI personalization dialog box. The user accesses it by a standard toolbar button. In the personalization dialog box, the following features are available:

- Hide/unhide columns
- Change attributes of columns
- Create and change views

Advanced Features

Influencing the Row Order at Runtime

Usually, the row order at runtime is determined by sort settings which are personalized by the user. But even in the absence of such sort rules, the List ATS attempts to keep the row order stable. While this is the

desired behavior in most use-cases, an application is sometimes required to have more control over the row order at runtime.

If you need to influence the row order which is displayed to the user, you can choose from the alternatives listed below. However, keep in mind that personalized sort and filter rules may have an impact on any attempt to manipulate the row order at runtime.

- Disable sorting and grouping by configuration or feeder class definition. This prevents the user from changing the row order without the feeder class noticing it. However, you must be aware that newly inserted rows are inserted at the point of selection or appended to the end as long as the feeder provides a change log in `GET_DATA`. If you want to avoid this, do not provide such a change log (or set the change log report to „data is new“).
- Clear the sort and grouping rules immediately before setting new data as described in the section *Clearing Sorting and Grouping Rules from Personalization*. Please consider very carefully whether this is a good choice for your application. It is the application developer's responsibility to ensure that the UI remains intuitive and usable when making use of this feature.
- Move individual rows to a given position – retaining their front-end order. In method `PROCESS_EVENT`, you define the insert position as described in the section *Defining the Insert Position of New Rows*. In method `GET_DATA`, you supply a change log which provides the rows to be moved to the insert position in parameter `ET_MOVE_TO_INSERT_POSITION` of method `GET_INDEX_MAP`. Note that this procedure *retains the sequence* of the moved rows as they were previously shown on the front-end. Any personalized filter will *not* be applied to these rows any more.
- Move individual rows to a given position – displaying them in their back-end order. In method `PROCESS_EVENT`, you define the insert position as described in the section *Defining the Insert Position of New Rows*. In method `GET_DATA`, you supply a change log which lists the rows as both „deleted“ and „inserted“. As a result, the inserted rows will be shown at the current insert position in the *order they appear in the original data table*. Any personalized filter will *not* be applied to these lines.

Reading the Row Order Displayed at Runtime

It is possible to retrieve the order of the data rows as currently displayed on the front-end. To do so, you use the instance of `IF_FPM_LIST_ATS_UI_INFO` that is passed as an import parameter `IO_UI_INFO` to `PROCESS_EVENT`. For more details, refer to the ABAP documentation of method `GET_LINE_ORDER` of that interface (accessible with the F9-key in the ABAP workbench).

This information is particularly useful if you want to provide functionality to move individual rows up or down. In this case, make sure you consider the situations that users have personalized any filter conditions or that users might have defined any grouping rules and some groups may be collapsed. In such cases, `GET_LINE_ORDER` indicates hidden rows.

Defining the Insert Position of New Rows

You can define the insert position for new rows or moved rows in method `PROCESS_EVENT`. In order to do so, use the instance of `IF_FPM_LIST_ATS_UI_INFO` that is passed as an importing parameter `IO_UI_INFO` to `PROCESS_EVENT`. For more details, refer to the ABAP documentation of method `SET_INSERT_POSITION` of that interface (accessible with the F9-key in the ABAP workbench).

You specify the insert position as a table index referring to the importing parameter `IT_DATA` of method `FLUSH`. You can also specify whether the index position is before or after that data row. Always keep in mind

that the row order of the front-end may be different from the sequence in `IT_DATA`. Therefore, specifying the insert position as „before 2“ is not the same as „after 1“.

If users have personalized a view with active grouping rules, new rows will be added to the group in which the row at the insert position belongs to.

Clearing Sorting and Grouping Rules from Personalization

If the feeder class provides no change log in `GET_DATA` and sets `EV_DATA_CHANGED` to `TRUE`, the data table will be subjected to any personalized sorting rules. If you want new data to be shown in exactly the same order provided by the feeder class, you must clear the personalized sorting rules, first. The feeder class can achieve this by raising the FPM event

`IF_FPM_GUIBB_LIST=>GC_EVENT_CLEAR_SORT_RULES_ATS` (make sure to attach the instance key to the event). The sorting rules will be cleared after the phase `PROCESS_EVENT` has been finished for this event. So, if the feeder class provides new data in method call `GET_DATA` for this event, this data will be displayed in the sequence provided by the feeder class.

Note that clearing sorting rules also results in clearing grouping rules. The event does *not* clear any filter rules.

Sorting and filtering of icons (columns that have the display type image)

List ATS provides the possibility to let the user sort and filter images. In order to enable this, some meta data needs to be attached to the images. This is done in feeder class method `get_definition`, parameter `et_field_description`, attribute `ENUMERATION`. This attribute is a table consisting of two columns: values and their descriptions. The column values should contain the technical names that appear at runtime in the table itself. For example, let's assume there is a table column `image`, having to two entries: „~icon/red“ and „~icon/green“. The table should be filled like this:

Value	Text (description)
~icon/red	Declined
~icon/yellow	Undecided
~icon/green	Approved

The order of the entries in the enumeration table is important. At runtime, when a user chooses the sort action, the column is sorted in the order defined in the enumeration table, that means in this example, red icons are displayed on top, followed by yellow icons and then green icons. It is also possible to change the content of the enumeration table at each round trip. This is done in feeder class method `get_data`, via parameter `IO_EXTENDED_CTRL`.

Changes to Elements from 'Old' List Component (GUIBB List)

Note the following changes regarding elements that were available in the previous version of the list component:

- Data changes in feeder class method `FLUSH`
Method `FLUSH` contains the parameter `it_data`. This parameter holds a reference to the internal table which contains the table data. Therefore, it is theoretically possible to change the table data via this parameter, though it was never the intention to allow the feeder class to do so. The correct place to change the data is in feeder method `get_data`. The List ATS does not allow you to change data in method `FLUSH` any longer.
- Style of rendering
The old list can be adjusted to render itself in the following different styles:
 - Normal list rendering
 - Row-repeater rendering
 - ALV rendering

This feature no longer exists in the List ATS (the features in ALV rendering are already included in the List ATS, for example, personalization, extended sorting, filtering, and so on).

List Component (GUIBB LIST)

Note that this component has been superseded by a new List ATS component, `FPM_LIST_UIBB_ATS`.

This is 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 configuration editor for Floorplan Manager, FLUID.

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
- GET_DATA

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).

Parameter	Description
EO_FIELD_CATALOG	<p>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</p> <pre>eo_field_catalog ?= CL_ABAP_STRUCTDESCR=>describe_by_name (<name>)</pre> <p> The list GUIBB supports only flat structures. When using deep structures, only the highest level fields are available.</p>


ET_FIELD_DESCRIPTION	<p>Optional only.</p> <p>Is used to add additional properties for the columns:</p> <ul style="list-style-type: none"> • Attribute <code>label_by_ddic</code> Indicates whether column header text should be taken from DDIC or not • Search helps DDIC, OVS, and freestyle search help • Attribute <code>header_text_wrapping</code> 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. • <code>CELL_DESIGN_REF</code> Points to a technical column which contains data to change the background color of cells (data type <code>WDUI_TABLE_CELL_DESIGN</code>). Use this to change the background color of single cells. 		
ET_ACTION_DEFINITION	<p>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 <code>DISABLE WHEN NO LEAD SEL</code> to True.</p>		
ET_SPECIAL_GROUPS	<p>Here you have the same options that you have in the ABAP ALV (see function module <code>REUSE_ALV_GRID_DISPLAY</code>) to group the fields within your field catalogue. You must enter the special group for each field in the field description table in field <code>SP_GROUP</code>. 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.</p>		
<p>GET_PARAMETER_LIST:</p> <p>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.</p>			
<table border="1"> <thead> <tr> <th data-bbox="162 1293 500 1337">Parameter</th> <th data-bbox="506 1293 1448 1337">Description</th> </tr> </thead> </table>	Parameter	Description	
Parameter	Description		
RT_PARAMETER_DESCR	<p>Is returned from this method. It describes which parameter is possible. In Field <code>TYPE</code>, the DDIC type needs to be entered.</p>		

<p>INITIALIZE:</p> <p>Called at runtime when the list is created. It is the first feeder method which is called from FPM.</p>			
<table border="1"> <thead> <tr> <th data-bbox="162 1543 384 1587">Parameter</th> <th data-bbox="391 1543 1448 1587">Description</th> </tr> </thead> </table>	Parameter	Description	
Parameter	Description		
IT_PARAMETER	<p>Contains a list of the feeder parameters and the values for them specified in the configuration.</p>		
<p>FLUSH:</p> <p>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.</p>			
<table border="1"> <thead> <tr> <th data-bbox="162 1841 524 1885">Parameter</th> <th data-bbox="531 1841 1448 1885">Description</th> </tr> </thead> </table>	Parameter	Description	
Parameter	Description		

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.	
Parameter	Description
IT_CHANGE_LOG	Lists all changes made by the user.
IT_DATA	Is a table containing the data.
IV_OLD_LEAD_SEL	Previous lead selection.
IV_NEW_LEAD_SEL	Current lead selection.

PROCESS_EVENT: Called within the FPM event loop and forwards the FPM <code>PROCESS_EVENT</code> to the feeder class. Here the event processing can take place and this is where the event can be canceled or deferred.	
Parameter	Description
IO_EVENT	The FPM event which is to be processed.
EV_RESULT	The result of the event processing. There are 3 possible values: <pre> 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 </pre>
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 <code>PROCESS_BEFORE_OUTPUT</code> event to the feeder class. Here you specify the list data after the event has been processed.	
Parameter	Description
IO_EVENT (IV_EVENTID)	The FPM event which is to be processed
IT_SELECTED_FIELDS	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.
ET_MESSAGES	A list of messages which shall be displayed in the message area.
EV_DATA_CHANGED	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.
EV_FIELD_USAGE_CHANGED	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.
EV_ACTION_USAGE_CHANGED	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.
CT_DATA	The list data to be changed.

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 <code>FIXED_VALUES_CHANGED</code> for this field.
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.
CV_FIRST_VISIBLE_ROW	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.



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.

Parameter	Description
IO_LAYOUT_CONFIG	Of type <code>IF_FPM_GUIBB_LIST_CONFIG</code> : This object 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.

Parameter	Description
IO_LAYOUT_CONFIG	Of type IF_FPM_GUIBB_LIST_CONFIG: This object provides the API to read the configuration to be saved.
ET_MESSAGES	A list of messages which shall be displayed in the message region.

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 usage 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.

Parameter	Description
CV_PAGING_ACTIVE	Tells the list UIBB whether the paging feature is active or not.
CV_BUFFERING_ACTIVE	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.

GET_ABSOLUTE_AMOUNT_OF_ROWS:

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

Parameter	Description
EV_AMOUNT_OF_ROWS	Tells the list UIBB how many rows the list has in total.
CV_FIRST_VISIBLE_ROW	Tells the list UIBB which should be the first visual row (all indexes are absolute)..

FLUSH:

Is called at the beginning of each FPM event. This method replaces the method

IF_FPM_GUIBB_LIST=>FLUSH.

Parameter	Description
IT_CHANGE_LOG	Contains the change log (all indexes are absolute).
IV_NEW_LEAD_SEL	Contains the new lead selection index (all indexes are absolute)
IV_OLD_LEAD_SEL	Contains the old lead selection index (all indexes are absolute)

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.

Parameter	Description
IV_START_ABSOLUTE	Contains the start index
IV_AMOUNT_ROWS	Contains the required amount of rows.
IT_SELECTED_FIELDS	Tells what columns are configured
CT_DATA	This parameter is used for the table data.

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.

Parameter	Description
IO_EVENT	The FPM event which is to be processed.
IV_LEAD_INDEX	Contains the lead index (all indexes are absolute).
IT_SELECTED_LINES	Tells which rows are selected.

IV_RAISED_BY_OWN_UI	Tells whether the event was raised by the actual search UIBB or by some other UIBB.
ET_MESSAGES	The feeder class may return messages via this parameter. As usual the messages will be displayed within the FPM message area.
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.	

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
- **Fixed Columns:** It is possible to set the initial column in a list as a fixed column. As a result, whenever a horizontal scrollbar is used, these fixed columns cannot be scrolled. There is an attribute in the *General Settings* for the GUIBB, where you can set how many columns need to be fixed. The following points are to be noted:
 - It is only the initial column that can be fixed; columns with index 1,2,3... It is not possible to fix columns at random indices.
 - The number of visible columns is independent of the number of fixed columns. For example, if the *Fixed Columns* is set as 3 and the total columns as 5, then the actual number of columns visible on the UI would be 8. Here, the first 3 columns would

always remain fixed and the count for the visible column starts from 1 after the last fixed column.

- **Initial Lead Selection:** If this field is selected then, at runtime if the list contains at least 1 record, the first record is lead selected.

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` and
`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 GUIBB 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)

This is 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. It is also possible to get an event for the closing of a particular node in the master column. FPM provides an event called ON_LOAD_CHILDREN only when a tree node is opened. However, whenever a node is collapsed, another FPM event MASTER_COLUMN_NODE_COLLAPSED is also triggered. This can be handled in the feeder class just like other FPM events. The event data specific to the index is available as event parameters.

- **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 can be 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 configuration editor for Floorplan Manager, FLUID.

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

<p>GET_DEFINITION:</p> <p>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).</p>	
Parameter	Description
EO_FIELD_CATALOG	<p>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</p> <pre>eo_field_catalog ?= CL_ABAP_TABLEDESCR=>describe_by_name(<name>)</pre> <p> Note</p> <p>The tree GUIBB supports only flat structures. When using deep structures, only the highest level fields are available.</p>
ET_FIELD_DESCRIPTION	<p>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 achieved with the help of the field COLUMN_TYPE (see note and table below).</p> <p>The ET_FIELD_DESCRIPTION parameter can also be used to provide additional properties for various columns:</p> <ul style="list-style-type: none"> • LABEL BY DDIC: Indicates whether a


	<p>column header text should be taken from DDIC or not</p> <ul style="list-style-type: none"> • Search helps: DDIC, OVS, and freestyle search help • <code>HEADER_TEXT_WRAPPING</code>: Determines whether the column header text shall be wrapped or not • <i>Attributes</i> for read-only, mandatory, enabled and visibility • <i>Attributes</i> for enabling filtering and sorting, for formatting amongst others
ET_ACTION_DEFINITION	A list of all actions (which will be transformed to FPM events at runtime) that you can assign to the tree elements.
ET_SPECIAL_GROUPS	This provides you with the same options that you have in the ABAP ALV (see function module <code>REUSE_ALV_GRID_DISPLAY</code>) to group the fields within your field catalogue. You must enter the special group for each field in the field description table in field <code>SP_GROUP</code> . At design-time, the FPM Configuration Editor groups the fields together, providing you with an easier method for finding fields.



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

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.	
Parameter	Description
RT_PARAMETER_DESCR	Is returned from this method. It describes which parameter is possible. In Field <i>TYPE</i> , the DDIC type needs to be entered.
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.	
IT_CHANGE_LOG	Lists all changes made by the user.
IS_DATA	A structure containing the changed data.
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.	
IO_EVENT	The FPM event which is to be processed.
EV_RESULT	The result of the event processing. There are 3

	<p>possible values:</p> <ul style="list-style-type: none"> • 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.
<p>GET_DATA:</p> <p>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.</p>	
IO_EVENT	The FPM event which is to be processed.
IT_SELECTED_FIELDS	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.
ET_MESSAGES	A list of messages which shall be displayed in the message area.
EV_DATA_CHANGED	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.
EV_FIELD_USAGE_CHANGED	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.
EV_ACTION_USAGE_CHANGED	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.
CT_DATA	The tree data to be changed. This is the actual data that gets rendered on the screen.
CT_FIELD_USAGE	<p>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.</p> <p> If you change the fixed values of a field, set the flag <code>FIXED_VALUES_CHANGED</code> for this field.</p> <p>See also the note below on assigning attributes to single cells or whole columns.</p>

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

Additional Information on the Hierarchical List Component

- *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.
- *Master Column Display Types*: It is possible to configure different display types for the master column in a Tree UIBB using FPM's configuration editor. To pass the data or to maintain the cell level properties of the master column (for example, *Tooltip, Enabled*), you must specify the particular ...*REF* column in the field description. This *REF* column must be associated with the field in the catalog which is used as the ROW KEY for the tree. With the *Checkbox* display type, it is possible to associate text with the column. To do this, use the TEXT_REF field in the field description to handle the cell level text content in columns where the display type is *CheckBox*.
- *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.

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).

- Sorting Event:

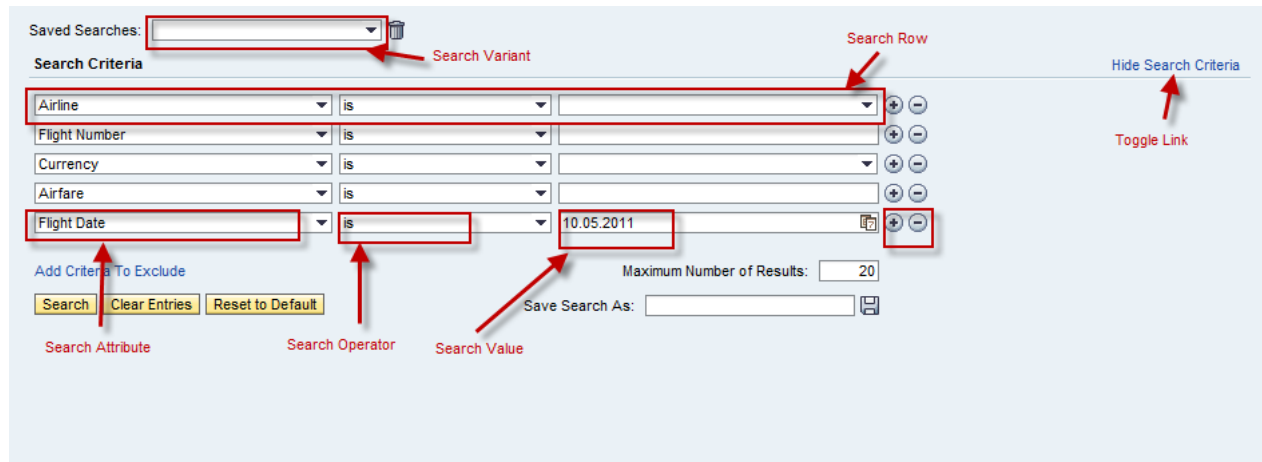
It is possible to sort the Hierarchical List/Tree UIBB. However, the FPM framework provides only the handle for the SORT event triggered on the UI by a user. As a result, an FPM event is triggered which can then be handled in the respective feeder class. Information such as the feeder class, the direction of the sort and the column which is sorted shall be available as event parameters. The actual sorting of the data needs to be handled in the feeder class itself. Note that the master column of the Tree UIBB shall not be available for sorting. Sorting can be activated just for the other columns in the Tree. Also the sorting needs to be activated via the feeder class, the same way it is for the List UIBB in the `GET_DEFINITION` method. The FPM event ID which is raised for sorting is `FPM_SORT_TREE_COLUMN`.

Search Component (GUIBB SEARCH)

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

You can determine the search query by configuring the Web Dynpro component `FPM_SEARCH_UIBB`.

The following screenshot details the search component elements:



Structure

A search component is comprised of the following objects:

- **Search Attributes**
 These 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 particular meta type is derived. The existing predefined meta types are: *text*, *alpha numeric*, *numeric*, *date*, *enumeration*, and *Boolean*.

The table below shows the mapping in the standard delivery.

ABAP Data type	Meta Type
Character or string	text
Char or string with 2 values in domain	Boolean
Char or string with more than 2 values in domain	enumeration
Numeric, integer, packed, float, hexadecimal	numeric
Date, time	date
Others	text

It is possible to overwrite the standard mapping by using the field description in the feeder class (see later in this section).

- **Search Operators**
 These 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.

The set of search operators can be modified in the feeder class (see later in this section):

Meta Type/ Property	Search Operators
Text (TE)	Is (01), is not (02), is empty (03), contains (05), starts with (04)
Alpha numeric (AN)	Is (06), is not (07), is empty (03), contains (05), is between (10), is greater than (08), is less than (09), is greater than or equal to (19), is less than or equal to (20)
Numeric (NU)	Is (06), is not (07), is empty (03), is between (10), is greater than (08), is less than (09), is greater than or equal to (19), is less than or equal to (20)
Date (TD)	Is (01), is not (02), is between (10), is earlier than (11), is later than (12), is earlier than or on (21), is later than or on (22)
Enumeration (EN)	Is (01), is not (02), is empty (03)
Boolean (BO)	Is (01)

- **Search Values**
These are the values that are used for the search. There is one input field where the user can enter a value. However, if the user chooses the search operator “is between” two input fields are displayed in order to build up a range. For the operator “is empty” no input field is displayed.
- **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 configuration editor for Floorplan Manager, FLUID.

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.



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.	
Parameter	Description
EO_FIELD_CATALOG_ATTR	<p>Is of type CL_ABAP_STRUCTDESCR. The components of this object are the available search attributes. The simplest way to provide a field catalog is to create a flat DDIC structure containing all search attributes and then get the field catalog via</p> <pre>eo_field_catalog_attr ?= CL_ABAP_STRUCTDESCR=>describe_by_name(<name of DDIC structure>)</pre> <p> The search GUIBB supports only flat structures. When using deep structures, only the highest level fields are available.</p>
ET_FIELD_DESCRIPTION_ATTR	Optional: 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. For details see below.
EO_FIELD_CATALOG_RESULT	Optional: Contains the columns that are possible to choose from during configuration of the result table.
ET_FIELD_DESCRIPTION_RESULT	Optional: Here you can provide additional information for the columns of the result table, e. g. the column text.
EV_RESULT_TABLE_SELECTION_MODE	This parameter determines the selection mode of the result table, such single line selection or multiple line selection.
EV_MESSAGE	Messages that will be displayed on the error page.
EV_ADDITIONAL_ERROR_INFO	Additional information about error messages.
ES_OPTIONS	Optional: Here you can adjust properties for the Search UIBB, e.g. Modify standard texts (details see below).
ET_ACTION_DEFINITION	Not used.
ET_SPECIAL_GROUPS	Not used.

Details for ET_FIELD_DESCRIPTION_ATTR: Optional. In this table you can provide additional information for each search attribute

Attribute	Description
NAME	Name of the search attribute.
IS_OF_TYPE	Overwrite the standard meta type.
TEXT	Text for the search criteria that is displayed.
INCLUDE_OPERATORS	Add operators to the standard set of operators.
EXCLUDE_OPERATORS	Exclude operators from the standard set of operators.
ENUMERATION	<p>Value set for an enumeration search criteria. The dropdown is always rendered as 'nullable', this means an empty entry in the value set is created in addition.</p> <p>It is possible to use an initial value as key but it is recommended not to do this. It is not allowed to use the operator IS_BETWEEN if the value set contains the initial value as key. In this case the NULL and the initial value for the low and high field cannot be handled properly.</p>
DDIC_SHLP_NAME	DDIC F4 help.
OVS_NAME	OVS name.
NULL_AS_BLANK	Displays zeros as blanks.
SIGN_POS	Sign left or right.
CONDENSE	Compress input.
DATE_FORMAT	Date format.
SHORT_TIME	Do not display seconds.
MULTI_VALUE_STRUCT	Structure of a multi-value field (details for multi-value fields see Appendix Multi-Value Fields).
EXCLUDE_CRITERIA	Not used at the moment.
FREE_TEXT_SEARCH	This attribute is a free text search.
WD_VALUE_HELP	WD value help.
MAX_1_VALUE	This criteria is a singleton, i.e. it can be used just once in the selection
DEFAULT_OP	Default operator for the search criteria. If there is one defined in the configuration, the default operator from the configuration is used.
DEACTIVATE_VALUE_HELP	No value help is displayed.

Details for ES_OPTIONS: Optional. Here you can adjust and overwrite standard properties of the Search UIBB.

Attribute	Description
TEXT_REPLACEMENT_FOR_HEADER	New text for header.
TEXT_REPLACE_HIDE_SEARCH_CRIT	New text for toggle link "Hide search criteria".

TEXT_REPLACE_SHOW_SEARCH_CRIT	New text for toggle link "Show search criteria".
LABEL_FOR_SAVED_SEARCHES	New label for "Saved Searches".
LABEL_FOR_SAVED_SEARCH_AS	New label for "Save Search as".
TEXT_SEARCH_BUTTON	Text for search button.
TEXT_CLEAR_BUTTON	Text for clear button.
TEXT_RESET_BUTTON	Text for reset button.
TTIP_DELETE_SEARCH_BUTTON	Tooltip for delete button.
TTIP_SAVE_SEARCH_BUTTON	Tooltip for save search button.
TTIP_SEARCH_BUTTON	Tooltip for search button.
TTIP_CLEAR_BUTTON	Tooltip for clear button.
TTIP_RESET_BUTTON	Tooltip for reset button.
APP_KEY_FOR_SAVING_SEARCHES	Configuration-Id for saving a search.
HIDE_MAX_NUM_RESULT_FIELD	Field "Max. Number of Results" is not displayed.
FREE_TEXT_SEARCH_ALLOWED	Not used
RAISE_EVENT_ON_ATTR_CHANGE	Raise event FPM_SEARCH_ATTR_CHANGED
RAISE_EVENT_ON_RESULT_LEAD_SEL	The FPM event FPM_RESULT_SEL (constant IF_FPM_GUIBB_SEARCH => FPM_RESULT_SELECTION) will be raised.
FIXED_WIDTH_FOR_USE_IN_DIALOG	If the search UIBB is used in a popup it should have a fixed width.
ALLOW_EXCLUDE_CRITERIAS	All search criteria are allowed as exclude criteria. It is not possible to choose just some search criteria as exclude criteria.
SET_MULTI_ATTR_TE_READ_ONLY	Only for Multi-value fields: The search value of a multi value field is set to read only. For details, see Appendix Multi-Value Fields.
MULTI_ATTR_SEPERATOR	Only for Multi-value fields: The separator between two entries in the formatted string of a multi-value field. For details, see Appendix Multi-Value Fields.
MULTI_ATTR_NEW_LINE	Only for Multi-value fields: The delimiter between two search attributes in the formatted string of a multi-value field. For details, see Appendix Multi-Value Fields.
PASS_INITIAL_SEARCH_ROWS	All search criteria will be passed to the user in the feeder class, i.e. also search criteria with initial input. The feeder method GET_DATA always gets all search criteria in parameter CT_FPM_SEARCH_CRITERIA passed.
USE_STD_DIALOG_MULTI_EDIT	Only for Multi-value fields: The standard dialog id 'FPM_SEARCH_STD_DIALOG' (constant IF_FPM_GUIBB_SEARCH=> FPM_SEARCH_STD_DIALOG) is used for a multi-value field pop up. For details, see Appendix Multi-Value Fields.
USE_EXTERNAL_FORMAT	If this parameter is ABAP_FALSE, any range conversion of search attributes is ignored. All attributes are typed and transferred in internal format.

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.

Parameter	Description
RT_PARAMETER_DESCR	Is returned from this method. It describes which parameters are possible. In Field TYPE, the DDIC type needs to be entered.

INITIALIZE:

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

Parameter	Description
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 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.

Parameter	Description
IT_FPM_SEARCH_CRITERIA	Contains the actual search criteria.
IV_MAX_NUM_RESULTS	Contains the amount of maximum number of result objects.
IT_SELECTED_LINES_OF_RESULT	Contains which lines within the result table are currently selected.

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 conduction 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.

Parameter	Description
IO_EVENT	The FPM event which is to be processed.
IT_FPM_SEARCH_CRITERIA	The current search criteria. The parameter ES_OPTIONS=>PASS_INITIAL_SEARCH_ROWS is evaluated. For details of this table see below.
IV_RAISED_BY_OWN_UI	Tells whether the event was raised by the actual search UIBB or by some other UIBB.
IV_MAX_NUM_RESULTS	Tells the maximum number of found objects to be displayed in the result table. Zero is allowed and should display all search results.
ET_MESSAGES	The feeder class may return messages via this parameter. As usual the messages will be displayed within the FPM message area.
EV_RESULT	The result of the event processing. There are 3 possible values:

	<p>ev_result = if_fpm_constants=>gc_event_result-OK</p> <p>ev_result = if_fpm_constants=>gc_event_result-FAILED.</p> <p>ev_result = if_fpm_constants=>gc_event_result-DEFER</p>
<p>Details for IT_FPM_SEARCH_CRITERIA (structure FPMGB_S_SEARCH_CRITERIA) : For each search row, the following attributes are passed. If the parameter es_options- PASS_INITIAL_SEARCH_ROWS = true in the method get_definition, then all search rows will be in this table. Otherwise, only the search rows where there is an input in the search value field are included.</p>	
SEARCH_ATTRIBUTE	Name of search attribute.
OPERATOR	Current search operator.
LOW	Search value in low field.
HIGH	Search value in high field, this is just used for operator "is between".
MULTI_VALUE_ATTR	Multi value (see Appendix Multi-Value Fields).
SIGN	Include (I) or exclude (E) search criteria.
FREE_TEXT_SEARCH	This attribute is a free text search.
IS_INITIAL_VALUE_EVALUATED	This value is ABAP_TRUE if something was entered in the LOW field. If the search criteria is Drop down list box and a value with an initial key was selected then IS_INITIAL_VALUE_EVALUATED = ABAP_TRUE and LOW is initial.
APPL_FORMATTED_MV_STRING_TABLE	Only for Multi-value fields: This parameter is used when the user formats the multi-value data. The passed string table will be displayed in the search row. (For details, see Appendix Multi-Value Fields).
<p>GET_DATA:</p> <p>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.</p> <p>With this method the user can also set initial data in the search UIBB. The user has to check for the event IDs cl_fpm_event=>gc_event_start and if_fpm_guihb_search=>fpm_reset_search.</p> <p>Check for event ID if_fpm_guihb_search=>fpm_execute_search and pass the search results from the application to the search UIBB.</p>	
Parameter	Description
IO_EVENT	The FPM event which is to be processed.
IV_RAISED_BY_OWN_UI	Tells whether the event was raised by the actual search UIBB or by some other UIBB.
IT_VISIBLE_ATTRIBUTES	Tells which search attributes are currently visible on the UI.
IT_SELECTED_COLUMNS_OF_RESULT	Tells the columns that have been configured for the result table.
IT_SELECTED_SEARCH_ATTRIBUTES	Tells the search attributes that have been configured.
ET_MESSAGES	The feeder class may return messages via this parameter. As usual the messages will be displayed within the FPM message area.

EV_SEARCH_CRITERIA_CHANGED	Tells the search UIBB whether the search criteria have been changed. If so then the search criteria on the screen will be updated accordingly.
ET_RESULT_LIST	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=>FPM_EXECUTE_SEARCH).
CT_FPM_SEARCH_CRITERIA	Contains the current search criteria. All search criteria will be passed from FPM to the user, i.e. also search criteria with initial search values.

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.	
Parameter	Description
IO_LAYOUT_CONFIG	Of type IF_FPM_GUIBB_SEARCH_CONFIG: This object 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.	
Parameter	Description
IO_LAYOUT_CONFIG	Of type IF_FPM_GUIBB_SEARCH_CONFIG: This object provides the API to read the configuration to be saved.
ET_MESSAGES	A list of messages which shall be displayed in the message region.

Enter, Reset, and Clear Buttons

Enter: When the cursor is positioned in a search value the search will start when pressing enter.

Clear: The *Clear* button clears all search fields, but does not reset the result list or the search statements (that is, attributes and operators remain unchanged, and lines are not removed). The *Clear* button also clears the *Save Search As* field and *Saved Searches* field, but the *Maximum Number of Results* field remains unchanged.

Reset: The *Reset* button rolls all search criteria back to the predefined default state (that is, the predefined combination of search criteria that was configured for the search). *Reset Page* clears all search fields as well as the *Save Search As* field. *Reset Page* also clears the search result list if the standard result list is used.

Result List

There are two alternatives when defining a result list:

- Use the standard result list from the Search UIBB

This list is very simple; it has no toolbar. If you want to use this list, define the parameters `EO_FIELD_CATALOG_RESULT` and `ET_FIELD_DESCRIPTION_RESULT` in the method `GET_DEFINITION`.

the connection id	currency	the flight date	Booking total
0555	EUR	27.10.2010	48.440,40
0555	EUR	24.11.2010	49.372,80
0555	EUR	22.12.2010	49.078,85
0555	EUR	19.01.2011	48.290,55
0555	EUR	16.02.2011	48.716,05
0555	EUR	16.03.2011	47.354,45
0555	EUR	13.04.2011	47.034,40
0555	EUR	11.05.2011	47.966,80
0555	EUR	08.06.2011	48.223,95
0555	EUR	06.07.2011	28.009,00

The reset functionality works automatically.

- Use a separate List UIBB

You can either create a separate list in the floorplan or you can use the Composite Search. When you use a List UIBB, you must implement the reset functionality and you are also responsible for the communication between the Search UIBB and the result list. You can use wiring, for example, for this communication.

If you use the composite search, you can add and configure a List UIBB directly in the configuration editor of the Search UIBB (see below):

Element	Default Operator	Display Type	Parameter	Component name	Text
Search Criteria				CARRID	
Search Criteria				CONNID	
Search Criteria				CURRENCY	Currency
Search Criteria				FLDATE	Flight Date
Search Criteria				PRICE	Airfare
Search Criteria				IS_INTERCONTINENTAL	

In the *General Settings* of the List UIBB there is a flag “*Table Is Search Result List*”. If this flag is selected, then the visible row count is changed to 10. This flag is automatically selected if you create the List UIBB in the configuration editor of the Search UIBB.

Additional Settings	
Title:	Result List
Accessibility Description:	
Visible Row Count:	10
Export to Spreadsheet:	Disabled
Scroll Mode:	
Selection Mode:	
Table Height:	Fit to Row Count (Resizing Possible)
Sorting:	Enabled (Changes not Saved)
Filtering:	Enabled (Changes Not Saved)
Calculation:	Calculations Disabled
Grouping:	Disabled
Enable Event on All Selections:	Selection Raises No FPM Event
Check Mandatory:	<input type="checkbox"/>
Hide Empty List:	<input type="checkbox"/>
Enable Creation of New Views:	<input type="checkbox"/>
Personalization:	Enabled (Button in List Toolbar)
Row Height:	0
Table Is Search Result List:	<input checked="" type="checkbox"/>
Fit Columns to Table Width:	<input checked="" type="checkbox"/>

Exclude Criteria

This relates to the definition of search statements that are used to reduce the search result list.

Firstly, the feeder class must activate this feature using method `GET_DEFINITION`, parameter `ES_OPTIONS`, and attribute `ALLOW_EXCLUDE_CRITERIAS`.

Secondly, in the FPM configuration editor, under *General Settings*, you enable or disable the *Exclude Criteria* field.

If *Exclude Criteria* is enabled, the *Add Criteria to Exclude* link appears on the search screen at runtime. When the link is pressed, a new *Exclude Criteria* area appears. It may contain all search criteria that have been defined in the normal search.

When the search is executed, all results found by the 'exclude area' will be removed from the overall search result before the search result is displayed.

Dependent Searches

This feature relates to the OVS value help in GUIBB search. There is now the possibility to influence the content of the F4 value help dialog box based on what is currently selected for the search attributes on the screen.

Example: A user selects *France* from the dropdown list in the search criterion *Country* at runtime; accordingly, the search criterion *City* should only display cities in France in its dropdown list. To activate this feature, you must implement the OVS interface `IF_FPM_GUIBB_OVS_SEARCH`.

This interface contains the method `SET_CURRENT_SEARCH_CRITERIA`. It indicates what is currently selected in all search criteria rows before an OVS roundtrip is started.

Launchpad Component (GUIBB LAUNCHPAD)

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

Structure

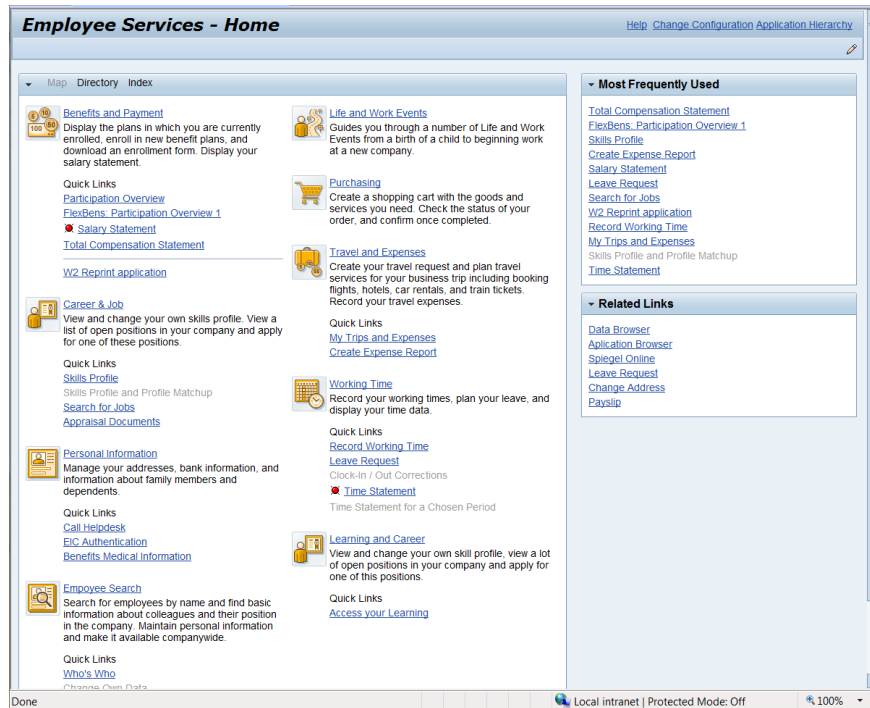
A navigation block is comprised of the following components:

- **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



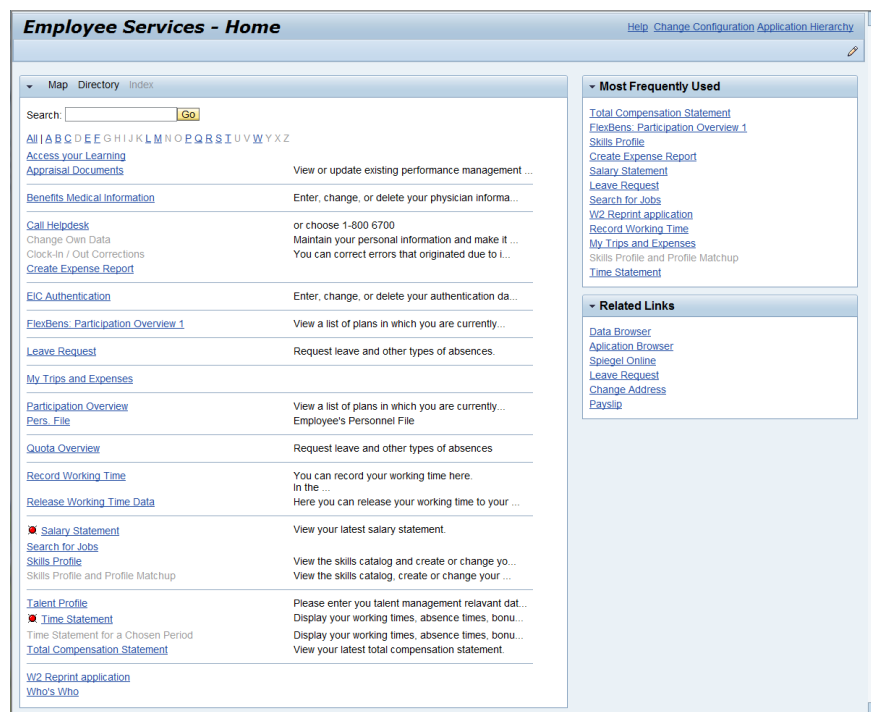
The *Directory* view displays all entries of the launchpad, also grouped in the same way as in the *Map* view, but subgroups are grouped by a *LinkChoice* UI element.

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

OVERWRITE_LAUNCHPAD_KEY:

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.

Parameter	Description
IV_ROLE	Role of a launchpad.
IV_INSTANCE	Instance of a launchpad.
IR_EVENT	FPM event
EV_ALSO_CALLED_IN_PBO	Flag if this method is processed also in Process Before Output
EV_ANYTHING_CHANGED_IN_PBO	Must be set to abap_true if anything was changed in Process Before Output

MODIFY:

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.

Parameter	Description
IR_NAVIGATION	A reference to the IF_FPM_NAVIGATION interface.
IR_EVENT	FPM event
EV_ALSO_CALLED_IN_PBO	Flag if this method is processed also in Process Before Output
EV_ANYTHING_CHANGED_IN_PBO	Must be set to abap_true if anything was changed in Process Before Output

OVERWRITE_REL_LINKS_LPD_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*.

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

MODIFY_RELATED_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.

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

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.

Parameter	Description
<code>IO_EVENT</code>	The FPM navigation event which is to be processed.
<code>EV_RESULT</code>	The result of the event processing. There are 3 possible values: <ul style="list-style-type: none"> <code>ev_result = if_fpm_constants=>gc_event_result-OK</code> <code>ev_result = if_fpm_constants=>gc_event_result-FAILED</code> <code>ev_result = if_fpm_constants=>gc_event_result-DEFER</code>

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.

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:



```
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'
                                   iv_value = lv_tab_name ).
lo_fpm->raise_event( io_event = lo_event ).
```

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



```
CASE io_tabbed->mo_event->mv_event_id.
WHEN 'CHANGE_TAB_NAME'.
DATA lv_name TYPE string.
DATA lv_id TYPE string.
```

```

io_tabbed->mo_event->mo_event_data->get_value( EXPORTING iv_key   = 'ID'
IMPORTING ev_value = lv_id ).
io_tabbed->mo_event->mo_event_data->get_value( EXPORTING iv_key   = 'NAME'
IMPORTING ev_value = lv_name ).
io_tabbed->rename_tab( iv_tab_id   = lv_id
iv_new_name = lv_name ).

```

It is also possible to hide the master and the detail UIBBs at runtime. Methods have been introduced in the IF_FPM_TABBED interface which can be used for this purpose. They are described below:

- **SET_MASTER_UIBB_HIDDEN** - To hide or unhide the master UIBB in the Tabbed component. The application must pass the UIBB information (*Component, Interface View and Configuration Name*) to FPM along with information about whether the UIBB should be set to visible or invisible.
- **SET_TAB_UIBB_HIDDEN** - To hide the detailed (tabbed) UIBB in the Tabbed component, similar to the master UIBB. Along with the UIBB details, the flag to set it as visible or invisible should be sent to FPM.

Note that even though all master UIBBs can be hidden or removed, the last detailed UIBB on a tab cannot be hidden or removed.

POWL Component (GUIBB POWL)

From NetWeaver 7.03 release onwards, Floorplan Manager (FPM) supports integration of the Standard POWL application by providing a new POWL component, FPM_POWL_UIBB.

The most important features of the POWL component are as follows:

- The POWL UIBB can be configured as a master list within any floorplan
- All FPM UIBBs/GUIBBs can be configured as a detail view
- The detail view participates seamlessly in the FPM event loop

Assumptions

Users are familiar with the POWL concept and maintenance of the POWL Cockpit.

For more information, see the POWL wiki:

<https://wiki.wdf.sap.corp/wiki/display/ERPOPSBNG/POWL+Framework>.

Pre-requisites

This document focuses on the POWL component within FPM, and not specifically on POWL configuration. Therefore, it is required that all POWL application-related entries such as *Application_ID*, *POWL_type*, *feeder implementation* and *POWL_query* are already created with corresponding associations to users and roles.

The POWL Component in FPM

With FPM's new configuration editor, FLUID, you can configure a POWL component as a UIBB. Depending on the floorplan, you can configure a POWL component as a master view in the following ways:

- in a subview of an Object Instance (OIF) floorplan
- in a main step or a substep of a Guided Activity (GAF) floorplan
- in a section of an Overview Page (OVP) floorplan
- in a tabbed component in a master UIBB
- in a tabbed component in a tab UIBB
- in a dialog box

Configuring a POWL Component in FPM

To configure a POWL component, complete the following steps:

1. Launch FLUID, the FPM configuration editor
 - a. Select a Web Dynpro application configuration in the *Object Navigator* of the *ABAP Workbench*.
 - b. 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.
 - c. In this window, go to the application's identification region and choose the *Configure Page* button.
 - d. Choose *OK* to create a Customizing record or *Cancel*.
 - e. Choose the *Edit* button.
2. Add the POWL UIBB
 - a. On the object schema tab, select the place where a new POWL UIBB is to be added.
 - b. Choose the *Add UIBB* button, and select *POWL Component*. The following values are inserted automatically by the system to identify it as a POWL component:
 - *Component*: FPM_POWL_UIBB
 - *Window Name*: POWL_WINDOW
 - c. Enter a configuration name. This is a mandatory field if you want to be able to configure the POWL UIBB. It is not necessary to enter an existing configuration; you can create a new configuration in the next steps.
 - d. (Optional) Choose the *Attributes* button on the main toolbar to display the attributes of the UIBB. Edit the remaining attributes such as *Location*, *Container Stretching*, *Sequence Index* and *Instance ID*.
 - e. Save your entries. If you are not using an existing POWL configuration, an error message appears to say your configuration does not exist. You will create this in the next steps.

Note that you can also drag and drop an existing POWL component configuration from the *Repositories* panel onto the object schema tab.

3. Configure POWL UIBB

- a. In the object schema tab, choose the inserted POWL UIBB and choose the *Configure UIBB* button.
- b. In the *Editor for the Web Dynpro ABAP Component Configuration* screen, choose *New* to create your new POWL configuration and enter the description and package details in the dialog boxes that follow. The system opens the FPM configuration editor with the name of the new configuration displayed in the header.
- c. Enter an existing *Application ID*. The input help selects existing configurations from the POWL Cockpit. When you enter an *Application ID*, the system automatically updates the field *Configuration Name* with `FPM_POWL_CONFIG`, the default configuration provided by FPM.

Note: the default configuration `FPM_POWL_CONFIG` has the following attributes enabled to ensure that the FPM configuration is compliant with the most recent guidelines:

- Enable FPM
- Enable New UI
- Display ALV Dialog as Popup

You can choose your own POWL configuration and enable these attributes yourself. The remaining attributes are features that POWL itself offers.

- d. Choose *Configure*. In the *Editor for the Web Dynpro ABAP Component Configuration* screen, choose *Continue in Change Mode*. The Web Dynpro default editor is opened.
- e. In the *Component-Defined* assignment block, choose the *confData* node to display the POWL attributes.
- f. Save your changes.

Attributes of the POWL Component

FIELD NAME	DESCRIPTION
<i>Query ID</i>	The query which is activated on load of the application. Data is selected from existing active queries in the POWL backend.
<i>Application</i>	This is the POWL Application ID that is maintained in the POWL Cockpit. It is a mandatory parameter.
<i>forAllq</i>	This parameter sets all the query selection parameters of all queries with the value of <i>querySelPara</i> .
<i>querySelPara</i>	This is a query selection parameter for all active POWL queries. If the checkbox for the field <i>forAllq</i> is selected, the value in <i>querySelPara</i> is applicable for all queries; otherwise, the value in <i>querySelPara</i> is applicable to the default query.
<i>POWLDeltaRendering</i>	This parameter enhances the browser performance during rendering.
<i>refreshAllq</i>	This parameter shall refresh all the active POWL queries registered for this application on every load of the FPM application.
<i>refreshOnLoad</i>	This parameter shall refresh the default active query on load of the application.
<i>Logon/server group</i>	These parameters can be used to set the details of the server group.
<i>POWL Wfk Theme ty</i>	This parameter allows you to select the theme supported by POWL. The available options are <i>Object worklist</i> and <i>Workflow Worklist</i> .
<i>Configuration Name</i>	This is the default POWL configuration provided by FPM. You have an option to change the configuration name. In this case, the custom configuration should set the required parameters.* * If you decide to have your own POWL configuration, you should select the following parameters to comply with SAP UI guidelines: <ul style="list-style-type: none"> • EnableFPM • EnableNewUI • DisplayAlvDialogAsPopup
<i>Component</i>	This is the standard POWL component (<i>POWL_UI_COMP</i>) which is used by FPM POWL UIBB.

The POWL Component at Runtime

The POWL UIBB provides you with FPM events which can be handled in the following ways:

- In the detail UIBB
Events are handled in the feeder class of the GUIBBs or in the Web Dynpro component which implements the `IF_FPM_UI_BUILDING_BLOCK` interface.
- In FPM's `CONF_EXIT` methods
Events are handled by the applications themselves as required.

The following events are available:

EVENT NAME	DESCRIPTION
FPM_GUIBB_POWL_ON_LEAD_SELECT	<p>This event is triggered by FPM when a lead selection event occurs in the POWL worklist. The event parameters supplied with this event are:</p> <ul style="list-style-type: none"> • <code>POWL_SELECTED_LINE_INDEX</code> (type <code>integer</code>) The value is the selected index in the POWL worklist. • <code>POWL_SELECTED_LINE_DATA</code> (type <code>ref to data</code>) The data object refers to the data in the line which is lead-selected in the POWL worklist. • <code>POWL_CURRENT_TYPE</code> (type <code>POWL_type_ty</code>) The type associated with the query on which the lead selection event has occurred.
FPM_GUIBB_POWL_ON_QUERY_SWITCH	<p>This event is triggered by FPM when there is a query switch operation performed by the user. The event parameters supplied with this event are:</p> <ul style="list-style-type: none"> • <code>POWL_SELECTED_QUERY_DATA</code> (type <code>POWL_query_sty</code>) • The value is the details about the newly selected query. • <code>POWL_CURRENT_TYPE</code> (type <code>POWL_type_ty</code>) • The type associated with the newly selected query. To handle the functionality in the detail UIBB, you can use <i>just</i> the information about the type of the newly selected query

	or <i>all</i> the details of the newly selected query
FPM_GUIBB_POWL_ON_NEW_QUERY	This event is raised when you click the <i>New Worklist</i> button on the POWL component toolbar.
FPM_GUIBB_POWL_ON_CHANGE_QUERY	This event is raised when you click the <i>Change Worklist</i> button on the POWL component toolbar.

Actions from Detail UIBB

The following actions can be triggered by the detail UIBB:

- Trigger a refresh of the POWL worklist
This is achieved by raising the FPM event `POWL_REFRESH`. The FPM framework informs the POWL to refresh the worklist.
- Request a follow-up from POWL
This is achieved by raising the FPM event `POWL_FOLLOW_UP` with the following event parameters:
 - `ADD_EVENT_DATA` (type `ref to data`)
 - `EVENT_PARAMETERS` (type `POWL_NAMEVALUE_TTY`)

Note: It is not necessary to provide information about the detail component in the feeder class of the POWL (method `IF_POWL_FEEDER~GET_DETAIL_COMP`) as the UIBB, configured in FPM, acts as a detail for the POWL by reacting to the above FPM events.

Note: At runtime, the actions *Query Switch* and *Refresh List* result in the removal of any previous lead selection in the table.

Navigation to Error Page

If no POWL *Application ID* is provided in the configuration of the POWL component, the FPM application is not launched; instead, the system navigates to the standard error page as shown below.



Composite Component (GUIBB Composite)

A composite component is a generic design template allowing you to group application-specific views (UIBBs) within a single UIBB.

It is implemented using the Web Dynpro component `FPM_COMPOSITE_UIBB`.

The information displayed in a composite component at runtime is determined by the feeder class assigned to the configurations of the individual Web Dynpro components within the composite component `FPM_COMPOSITE_UIBB`.

The main use case is the Overview Page Floorplan (OVP) where composite UIBBs can be used to arrange several UIBBs in an assignment block. However, Composite UIBBs can be used in any floorplan where several UIBBs need to be grouped together.

You use this design template when you want to display data in different ways within a single UIBB. For example, you can display a form, a table and a search component all within one UIBB. You can determine the concrete display of the data in a composite component when configuring the Web Dynpro component `FPM_COMPOSITE_UIBB`.

Composite UIBBs explicitly do **not** serve Master/Detail purposes.

Structure

A Composite UIBB has the following possible layouts:

- One Column Layout (Standard)
The maximum number of UIBBs is 3 and these are arranged below each other.
- Two Column Layout (50:50)
The maximum number of UIBBs is 6 and these are arranged with a maximum of 3 rows and 2 columns.

A UIBB can be stretched over several rows or columns in both layouts.

Editing the Composite Component

You edit this component using the FPM configuration editor, FLUID.

Changing the Composite UIBB dynamically at Runtime

You may add, remove and get UIBBs from your composite UIBB component during runtime.

To do this, proceed as follows:

1. Choose an application-specific Web Dynpro component and add the Web Dynpro interface `IF_FPM_CMPST_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.
3. In the component controller, implement the method `OVERRIDE_CONFIG_COMPOSITE`.

Alternatively, you can choose to use a class-interface instead of a Web Dynpro interface. The class interface is `IF_FPM_COMPOSITE_CONF_EXIT`.

Analytical Components

You can use analytical components to embed analytical and planning content into transactional applications in Floorplan Manager without development effort.

You can either use the standard application *List Report on Analytic Query* (`WDA_BS_ANLY_LIST_OVP`) or build your own applications for list reporting and planning purposes or modify an existing application.

In addition to the user-interface building blocks (UIBBs) provided by the Floorplan Manager, you can add analytical UIBBs to FPM applications without development effort.

The following analytical UIBBs/components exist:

- **Analytics List Component** (`WDC_BS_ANLY_LIST_ALV`)
You can display selected data in a list. If you use a query that is ready for input, you can also use a list component for planning purposes.
- **Hierarchical List (Tree) Component with Analytics Feeder Class**
You can use the Web Dynpro component `FPM_TREE_UIBB` of the Floorplan Manager with an analytics feeder class `CL_BS_ANLY_TREE_FEEDER` to display selected data in a hierarchical list.
- **Analytics Search Component with Analytics Feeder Class**
You can use the Web Dynpro component `FPM_SEARCH_UIBB` of the Floorplan Manager with the analytics feeder class `CL_BS_ANLY_LIST_SEARCH_FEEDER` to restrict the amount of data selected.
- **Chart Component** (`BS_ANLY_CHART_UIBB`)
You can display selected data in a chart. Feeder class is `CL_BS_ANLY_CHART_FEEDER`.
- **Generic Analytic Application Controller** (`WDC_BS_ANLY_APPCONTROLLER`)

This is a central module that controls the interaction between the different components and events.

You can use these components to display the results of an analytic query in a list or chart, and to restrict the results in a selection screen with query variables.



These analytical components are part of the Business Suite Foundation layer (Software Component `SAP_BS_FND`) and, as such, are not available to all applications.

Analytics List Component

The analytics list component `WDC_BS_ANLY_LIST_ALV` uses the SAP List Viewer (ALV) to display the data of an analytic query (BI query) as a table or with Crystal Reports.

You can sort and filter data, define your own view, print and export lists to Microsoft Excel.

You can change the *List Drilldown* by choosing the *Drilldown Settings* pushbutton at runtime.

If you choose Crystal Reports to display the data, you can change the formatting to meet your requirements and you can export the data from the Crystal Report view to files with the different formats.

You can navigate from the list to predefined destinations, such as master data reports or transactions (for example determine navigation targets through another launchpad or using *Sender/Receiver Assignments* defined in transaction `RSBBS`).

If you have defined any destinations, you can choose the *Goto* pushbutton at runtime to display all the destinations available.

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.

Component Configuration

The configuration editor consists of the following areas:

- General Settings, e.g. query name or drilldown settings
- List Settings, e.g. info allowed or export allowed
- Navigation Settings, e.g. Launchpad settings

▼ Component-Defined

General Settings

Query:

Title:

Title Visible

Drilldown Settings:

List Settings

Filter Allowed:

Settings Dialog Visible:

Column Selection Allowed

Calculation Allowed

Saving Views Allowed:

Sorting Allowed:

Changing Display Type Allowed:

Export Allowed:

Printing Allowed:

Info Allowed:

Number of Rows:

Number of Fixed Columns:

Number of Non-Fixed Columns:

Display All Columns

Selection Type:

Navigation Settings

Goto Allowed:

RR: Runtime Environment for Report Type Query:

Launchpad Role:

Launchpad Instance:

Tree Component with Analytics Feeder Class

The tree component `FPM_TREE_UIBB` with the analytics feeder class `CL_BS_ANLY_TREE_FEEDER` allows you to display an analytics query in a hierarchical display.

▼ Result		
Material	Material	JAN 2009/Euro/Net value
▼ Classic	CLASSIC	2.254,24
▪ Il Trionfo Del Tempo	180000000000000001	151,92
▪ Violin Concerto No.1	180000000000000004	575,68
▪ BWV 1060	180000000000000007	727,44
▪ Symphoniae Sacrae	180000000000000010	799,20
▼ Jazz	JAZZ	965,22
▪ Sketches of Spain	180000000000000025	389,70
▪ Blues Train	180000000000000028	575,52
▼ Pop	POP	1.486,80
▪ Modus Operandi	180000000000000013	323,82
▪ Decksdrumsandrockan	180000000000000016	467,64
▪ Closer	180000000000000019	539,46
▪ Kaleidoscope	180000000000000022	155,88

However, the query has to fulfill one of the following conditions so that it can be displayed in the tree component:

- Either the query contains exactly one characteristic in the row drill-down and has a hierarchy on this characteristic
- Or the query contains several characteristics in the row drill down and has the feature “Display Rows as hierarchy” switched on

The list component supports the following functions:

- You can use analytic queries that are ready for input.
- You can display multiple attributes.
- You can trigger planning functions.

You can use one of the following feeder actions:

- For planning functions: Feeder actions `BSA_PLFUNC_01` to `BSA_PLFUNC_10`
- For planning sequences: Feeder actions `BSA_PLSEQU_01` to `BSA_PLSEQU_10`

You can use the following event parameters:

- For the feeder action BSA_PLFUNC_01 to BSA_PLFUNC_10: Select the planning function and filter.
- For the feeder action BSA_PLSEQU_01 to BSA_PLSEQU_10: Select the planning sequence.

Search Component with Analytics Feeder Class

The search component `FPM_SEARCH_UIBB` with the analytics feeder class `CL_BS_ANLY_LIST_SEARCH_FEEDER` 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 screenshot shows the 'Selection' dialog box in SAP. At the top, there is a 'Variant' dropdown set to 'My Report Selection' with a trash icon. Below this is the 'Selection Criteria' section, which has a 'Hide Selection Criteria' link. The criteria are listed in a table:

Characteristic	Operator	Value	Copy	Plus	Minus
Material (SingleValue;Optional)	is				
Country	is	DE			
Country (MultiSingleValue; Optional)	is				
Sales Docs (Interval;Optional)	is				
Cal. Month (SelOpt;optional)	is	09.2010			

Below the table, there is a link 'Add Criteria To Exclude', two buttons 'Execute' and 'Clear Entries', and a 'Save Variant As' section with a dropdown set to 'My Report Selection' and a save icon.



The following restriction applies:

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, `OCALMONTH` with conversion routine `PERI6`, you cannot enter a number sign.

Chart Component with Analytics Feeder Class

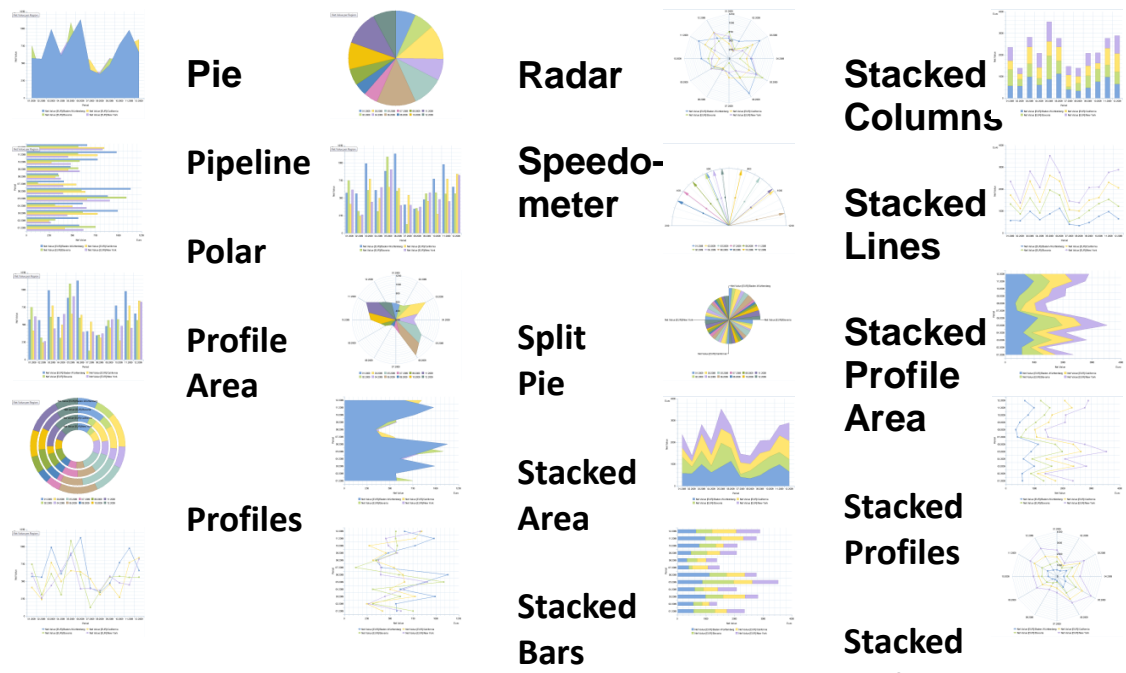
The chart component `BS_ANLY_CHART_UIBB` with the analytics feeder class `CL_BS_ANLY_CHART_FEEDER` allows you to display an analytic query in a graphical form.

The chart component can be used to visualize data series as a chart. For example vertical bar charts (columns):



You can determine the concrete display of the data in a chart when configuring the Web Dynpro component `BS_ANLY_CHART_UIBB`.

The following chart types are available:



The following chart types are not supported:

- Gantt
- Milestone trend analysis (MTA)
- Heatmap
- Time bar
- Stacked time bar
- Time column
- Stacked time column

Structure

A chart consists of the following areas:

- Toolbar
- Graphic
- Header

The chart can display/use one or several data series. Data series consist of points. The points of the different series can be grouped into categories.

For more information, see BW documentation under [Editing Charts](#) and [BusinessGraphics](#).

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

- The chart type or XML chart configuration file.
- The tooltip, width and height of the chart UIBB.
- The possibility to propagate chart selection events to the feeder class.

- The possibility to export the chart as GIF file.
- The header of the chart UIBB.
- The toolbar of chart UIBB.
- The title and units of the chart axes.



The feeder class is responsible to return the definition of the series, data points and categories. The feeder class `CL_BS_ANLY_CHART_FEEDER`, which displays BI queries, defines data points, series and categories as follows:

- Each key figure value of a BI query in a data column and a certain row represents a data point (a single cell, which displays a key figure).
- Each data column (column with key figure values) of a BI query becomes a series.
- Each row of the BI query becomes a category. The text of a category is derived from the characteristic value combination of the row.

Integration

You can configure a chart component using the Chart Configuration Editor of the Floorplan Manager.

IF_BS_ANLY_GUIBB_CHART Interface

The feeder class of the CHART UIBB must implement this interface. The following tables describe the methods (and the attributes) of the `IF_BS_ANLY_GUIBB_CHART` interface.



(Optional) You can use the additional interface `IF_BS_ANLY_GUIBB_CHART_EXT` and implement it in your feeder class.

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 of the interface `IF_BS_ANLY_GUIBB_CHART`:

- `GET_DEFINITION`
- `GET_DATA`

Methods

GET_DEFINITION:	
Allows the feeder to provide all necessary information for configuring a chart: the definition of the chart categories and chart data series; the list of toolbar actions (FPM events).	
Parameter	Description
<code>IS_CHART_TYPE</code>	Contains information about the chart type: Does it have a category, time and / or value axis?
<code>ET_CATEGORY</code>	Definition of the chart categories (this is only required if the chart type is category based)

GET_DEFINITION:

Allows the feeder to provide all necessary information for configuring a chart: the definition of the chart categories and chart data series; the list of toolbar actions (FPM events).

Parameter	Description
ET_SERIES	Definition of the chart series
ET_TOOLBAR	Definition of the chart toolbar
ES_MESSAGE	Error message
EV_ADDITIONAL_ERROR_INFO	Additional error information
ES_SELECTION_BEHAVIOR	Defines, whether single or multiple selection (of points, categories or series) in the chart during runtime will be supported in principal.
EF_STORE_XML_IN_CONTEXT	EF_STORE_XML_IN_CONTEXT = space: The chart configuration file is stored in the MIME Repository; at runtime the configuration file can be changed. EF_STORE_XML_IN_CONTEXT = 'X': The configuration file is stored in the Web Dynpro context and cannot be changed at runtime.

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.

Parameter	Description
RT_PARAMETER_DESCR	Is returned from this method. It describes the parameters. In Field TYPE, the DDIC type needs to be entered.

INITIALIZE:

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

Parameter	Description
IT_PARAMETER	Contains a list of the feeder parameters and the values for them specified in the configuration.
IO_APP_PARAMETER	Contains a reference to an interface for reading Web Dynpro / FPM application parameter
IV_COMPONENT_NAME	ID of the Web Dynpro component
IS_CONFIG_KEY	Configuration key of the Web Dynpro component
IV_INSTANCE_ID	FPM Instance ID

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 chart itself). Use it to forward changed chart data to other components in the same application.

Parameter	Description
IT_DATA	Chart Data

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 deferred.	
Parameter	Description
IO_EVENT	The FPM event which is to be processed. The parameters of the event are retrieved with the method call <code>io_event->mo_event_data->get_keys()</code> .
EV_RESULT	The result of the event processing. There are 3 possible values: <ul style="list-style-type: none"> • 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
IV_RAISED_BY_OWN_UI	Indicates whether the event was raised by the own instance of the chart UIBB or by any other UIBB
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.	
Parameter	Description
IO_EVENT	The FPM event which is to be processed.
IV_RAISED_BY_OWN_UI	Indicates whether the event was raised by the own instance of the chart UIBB or by any other UIBB
EV_DATA_CHANGED	For performance reasons, the chart UIBB 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.
EV_TOOLBAR_CHANGED	For performance reasons, the chart UIBB adjusts the toolbar of the chart only if the toolbar has been changed. To indicate this, set this flag whenever you change the toolbar within this feeder.
ET_MESSAGE	A list of messages which shall be displayed in the message area.
CT_DATA	Chart Data
CT_TOOLBAR	Chart Toolbar
CS_CHART_SETTINGS	Chart settings (various chart texts which can be changed at runtime; furthermore the chart UIBB can be set to invisible)

GET_DEFAULT_CONFIG:	
Called when creating a new configuration. Use it to provide pre-configured chart configurations when a user starts the FPM Configuration Editor.	
Parameter	Description
IO_LAYOUT_CONFIG	Of type IF_BS_ANLY_GUIBB_CHART_CONFIG: This object provides the API to create a default configuration.

CHECK_CONFIG:	
Implement this if you want to make your own application-specific checks on the configuration in the FPM Configuration Editor immediately before saving.	
Parameter	Description
IS_CHART_TYPE	Contains information about the chart type: Does it have a category, time and / or value axis?
IO_LAYOUT_CONFIG	Of type IF_BS_ANLY_GUIBB_CHART_CONFIG: This object provides the API to read the configuration to be saved.
ET_MESSAGES	A list of messages which shall be displayed in the message region.

Chart Configuration for the Floorplan Manager

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

Features

The Chart Configuration consists of the following work areas:

- **Toolbar**
For example, you can assign the analytics feeder class `CL_BS_ANLY_CHART_FEEDER`.
- **Preview**
In the preview, the chart in the current configuration is displayed so as to give you a picture of the layout of the chart.
- **Configuration Dialog**
Attributes of chart that can be changed are displayed in the configuration dialog.

The attributes comprise the following areas: *General Settings*, *Chart Appearance*, *Primary Axis Text*, and *Selection in Chart*.

▼ Component-Defined

Display Feeder
Display Parameters
Configure Toolbar
Define Final Flags
Export
Import

General Settings

Title:

Tooltip:

Height:

Width:

Export as Picture Allowed:

Chart Appearance

Define Chart Appearance By:

Chart Customizing File:

Chart Type:

Display Values:

Primary Axes Texts

Title of Category / Value Axis 1:

Unit of Category / Value Axis 1:

Title of Value Axis 2:

Unit of Value Axis 2:

Selection in Chart

Selection Type:



Chart Appearance

You can switch between the SAP-defined default settings and your own Customizing scheme that you created using the SAP Chart Designer.

Using the Chart Designer **Error! Reference source not found.** (or even a text editor) you can define many more settings than directly using the **Error! Reference source not found.**

To define the chart appearance (*Define Chart Appearance by*) you have the following options:

- *Default Settings*
The configuration of the chart is exclusively based on the Web Dynpro configuration settings. You can configure here e.g. the chart type and the titles and units of the chart axis.

As long as the user has not imported a Customizing scheme, this dropdown list box is disabled and *Default Settings* option is selected.

- *Customizing Scheme (Chart Designer)*
In addition to the parameters that are directly available in configuration dialog, you can influence the chart's appearance by defining a chart Customizing scheme.

The configuration of the chart is based on an XML chart Customizing file. This should be used for more complex charts definitions. For more information, see documentation of dataelement `BS_ANLY_CUST_TYPE`.

You can use the following tools:

- **A text editor**
This can be any text editor such as Notepad. However, this provides no syntax support and a low user experience.
- **SAP Chart Designer**
This is already implemented and can be downloaded from the SAP Developer Network.
You can make your changes in the Chart Designer or in the transaction `BS_ANLY_CD` (which calls the Chart Designer from SAP GUI). For more information see, [Chart Designer](#).

Chart Customizing File

This field is editable when you have chosen *Customizing Scheme (Chart Designer)*.

When using the parameter `EF_STORE_XML_IN_CONTEXT` in the feeder class method `GET_DEFINITION` you have the following options for configurations based on a chart Customizing file:

- `EF_STORE_XML_IN_CONTEXT = space`: This is the default setting. The chart configuration file is stored in the MIME Repository; at runtime the configuration file can be changed.
- `EF_STORE_XML_IN_CONTEXT = 'X'`: The configuration file is stored in the Web Dynpro context and cannot be changed at runtime.

The Chart Configuration launches from the configuration editor of the Floorplan Manager automatically when you start the configuration of an application-specific UIBB that uses the `BS_ANLY_CHART_UIBB` Web Dynpro component.

Adding a Chart Component

You add a Chart Component to an application configuration in the same way that you add components provided by FPM (see the section [Adding an Existing UIBB to an Application](#)). When you insert the Chart UIBB in the `<Floorplan Name> Schema` tab page, the following information is automatically inserted into the fields below:

- **Component:** `BS_ANLY_CHART_UIBB`
- **Window Name:** `CHART_WINDOW`

Use the field help to select a different window if required.

Use the field help to select the configuration name.

When configuring the Chart component, choose the *Feeder Class* button in the *General Settings* tab and select `CL_BS_ANLY_CHART_FEEDER` (or your own feeder class). The system lists all classes implementing the interface `IF_BS_ANLY_GUIBB_CHART`. In the *Edit Parameters* dialog box, enter an Analytic Query.

The following information is useful when configuring a Chart Component.

FPM Events and the Chart Component

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

- **Chart Selection Events**

FPM events are raised when selecting a series, category or point in the chart. However this does only happen if the flag *Propagate Selection Events* is set in the chart configuration.

The information about the point, category and / or series is added as event parameters to this FPM event:

- `IF_BS_ANLY_GUIBB_CHART => GC_EVENT_POINT_SELECTED`
- `IF_BS_ANLY_GUIBB_CHART => GC_EVENT_CATEGORY_SELECTED`
- `IF_BS_ANLY_GUIBB_CHART => GC_EVENT_SERIES_SELECTED`

- **Toolbar Events**

Each toolbar element raises 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.

- **New Graphic**

A new graphic, e.g. with a new chart type, a new chart configuration file or with new series, is displayed by the event `IF_BS_ANLY_GUIBB_CHART => GC_EVENT_NEW_GRAPHICS`.

For more information about charts, see BW documentation under [Editing Charts](#).

For more information about analytical UIBBs, see documentation on SAP Help Portal <http://help.sap.com> under *SAP Business Suite → Processes and Tools for Enterprise Applications → Analytics Infrastructure → Analytical Components in Floorplan Manager Applications (CA-EPT-ANL)* and on SDN <http://www.sdn.sap.com> under *BPX Community → Home → SAP Business Suite → SAP Business Suite Analytics*.

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 an Analytic 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 an Analytic Query
- `IF_BS_ANLY_SELECTION->GET_VARIABLES`
Reads metadata of variables
- `IF_BS_ANLY_SELECTION->READ`
Reads variable values

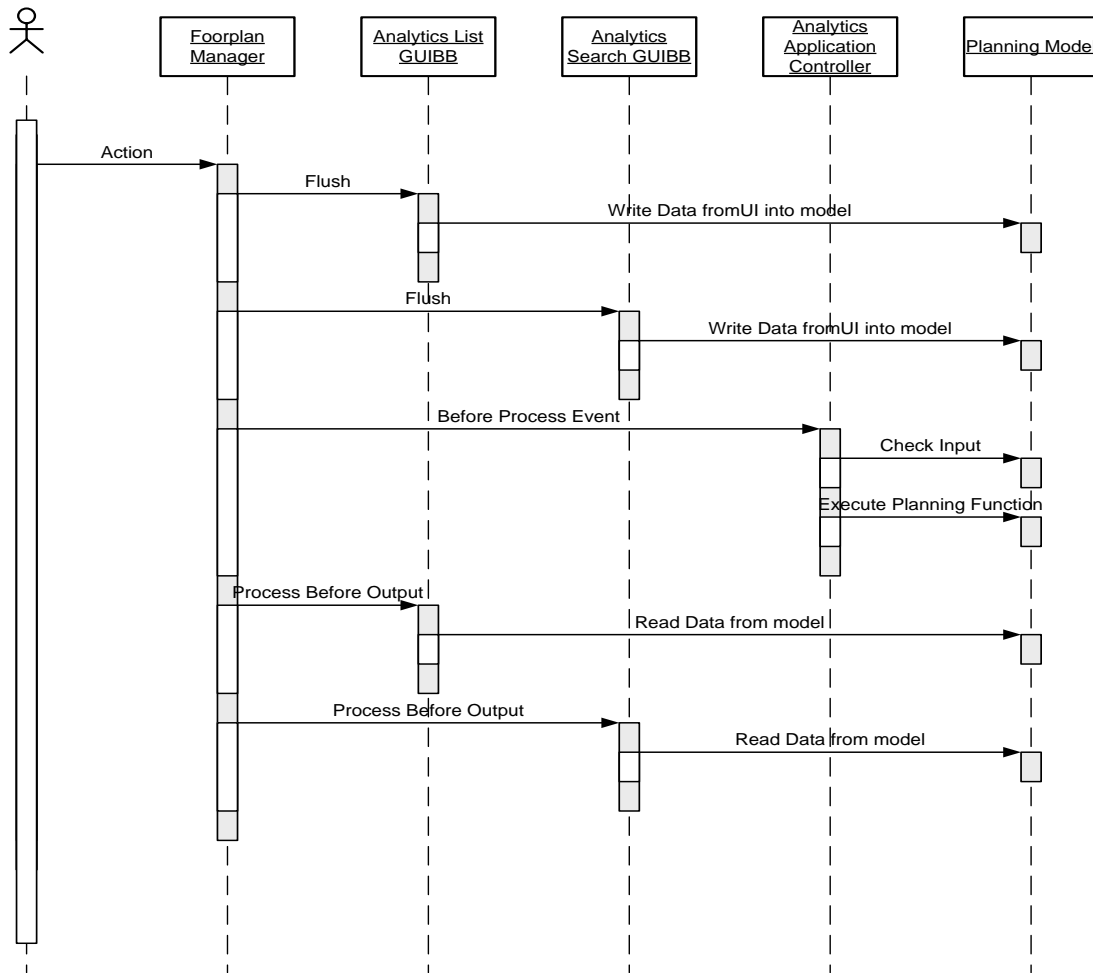
- `IF_BS_ANLY_SELECTION->WRITE`
Writes back changed variable values
- `IF_BS_ANLY_SELECTION->CHECK:`
Checks entered variable values. Note that `CHECK` has to be called after `WRITE` otherwise the changed variable values are not required.
- `IF_BS_ANLY_LIST_RRI~GET_RECEIVERS:`
Retrieves the Report-Report 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:

**Note:**

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.
 - Note:
 - In this phase, the data has not been written back to the planning buffer and the variables have not been submitted.
 - 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.

REUSE UIBB (RUIBB)

Reuse UIBBs (RUIBBs) are intended to be reused by other applications. You can add them to your application configurations and immediately start using their functionality without having to write lots more additional code.

The Attachment RUIBB and Notes RUIBB are typical examples of RUIBBs.

What is the Difference between a RUIBB and a GUIBB?

A Generic UIBB (GUIBB) is a pure UI pattern which itself does not support any business functionality. There is always the need to provide the business functionality.

A RUIBB is, more or less, a complete component offering common business logic and a user interface to go with it.

Attachment RUIBB

RUIBB Interface

FPM provides a marker interface, `IF_FPM_RUIBB`, which must be implemented by the WD components.

This interface can also be used by other applications, to turn a UIBB into a RUIBB.

For the Attachment RUIBB, the `IF_FPM_RUIBB` interface is implemented by the attachment 'wrapper' component `FPM_ATTACHMENT_WRAPPER`.

KPRO Configuration for Attachment RUIBB

KPRO provides the functionality to store the attachments. This is the basic KPRO configuration required for Attachment RUIBB. It does not cover all aspects of KPRO.

1. Create a *Content Table*

In transaction SE11, you can copy a row from table SDOKCONT1, for example FPM_T_ATTACHMENT.

2. Create a *Content Repository*

Open transaction OAC0, create a new content repository and specify the following details (note that the table below shows only sample details):

Field	Value
Content Rep.	FPM_ATTACHMENT
Description	FPM Attachment
Storage Type	SAP System Database
Rep. Sub-Type	Normal
Version No.	0046
Contents Table	FPM_T_ATTACHMENT

3. Create a Category

In transaction OACT, enter the category name and the content repository you created in the previous step.

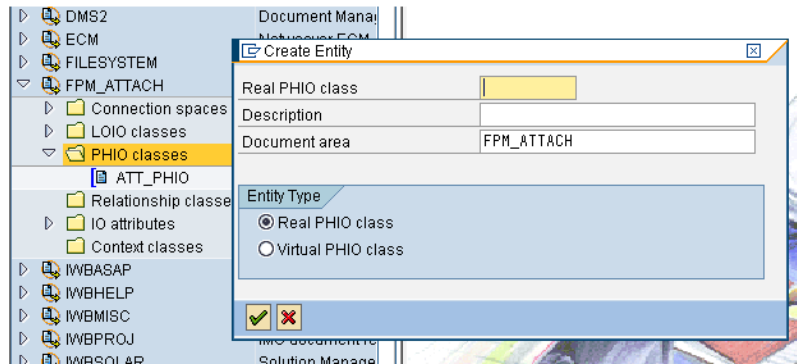
4. KPRO Modeling

(I) Create a Document Area

In transaction DMWB, choose *Create* to display the *Create Entity* dialog box. Select the *Document Area* radio button and specify the *Document area* and *Description*. Save and activate the Document area

(II) Create a PHIO Class

- In the *Document Modeling Workbench* hierarchy, expand the folder for the document area you have just created. In the context menu for *PHIO classes*, select the *Create* option.
- Select *Real PHIO Class* and enter the PHIO class and description.



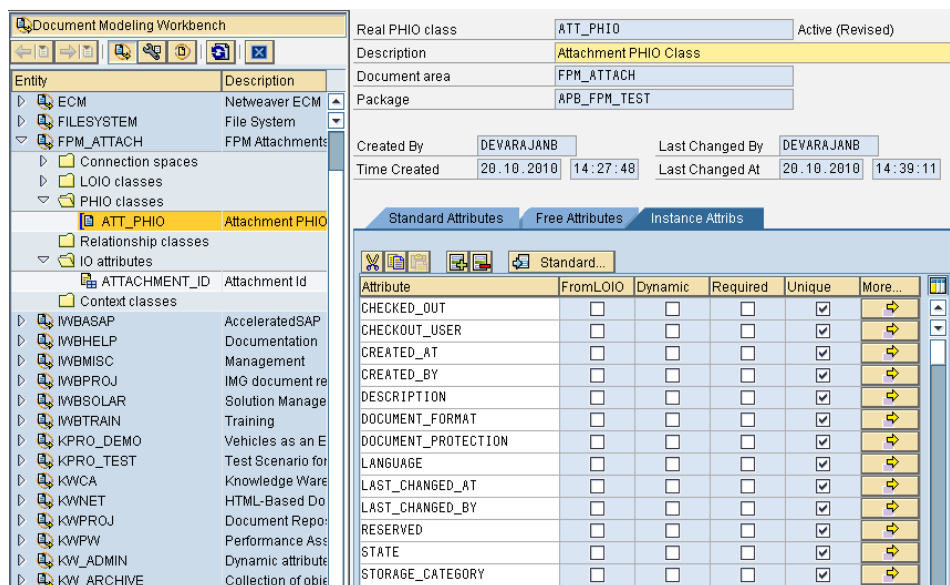
Creating a PHIO Class

- In the *Document Modeling Workbench* hierarchy, double-click the newly created PHIO class and, on the *Standard Attributes* tab page, specify the following attributes:

Attribute	Value
Auto_Index	X
Storage_category	Specify the category you created in the previous step and mark it as <i>Exposed</i>

PHIO PROPERTIES

- Choose the *Tabulation* button.
- Choose the *Copy Table Set* button.
- Specify the *Name Prefix for Copy Tables* and execute. Choose *OK* for all the information dialog boxes that appear.
- In the *Document Modeling Workbench* hierarchy, double-click on the PHIO class. On the *Instance Attribs* tab page, choose the *Standard* button and select all the attributes and choose *OK*.

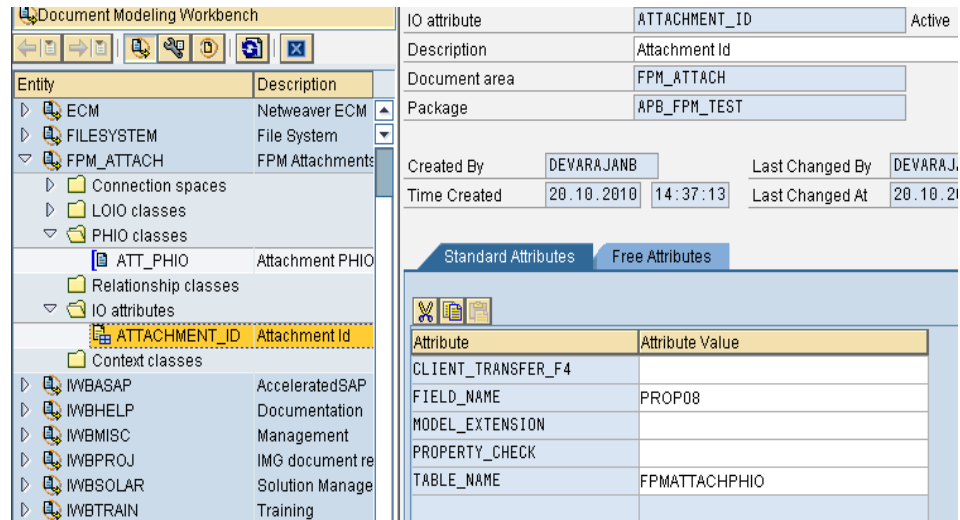


Instance Attributes for the PHIO

- Save and activate.

(III) Create IO Attribute

- In your document area inside the *Document Modeling Workbench* hierarchy, choose *Create* in the context menu for the *IO Attributes* folder.
- Enter the *IO Attribute* and *Description* and choose *OK*.
- Double-click on the newly created IO attribute and enter the attribute values for *Field_Name* and *Table_Name* (see screenshot below).



IO Attributes

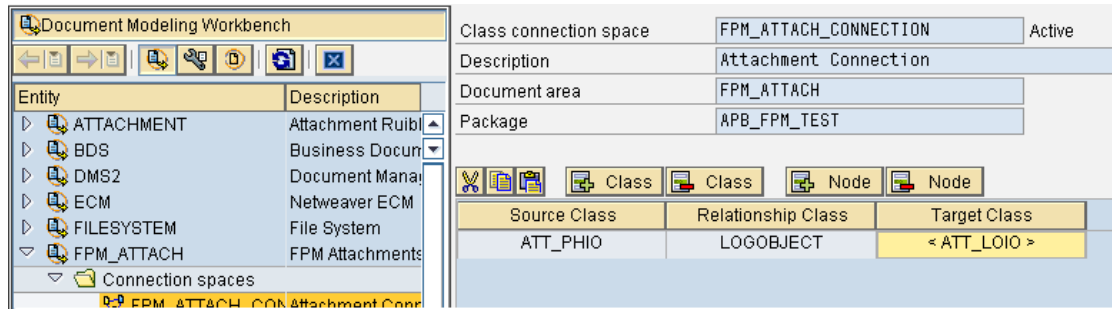
- Save and activate.

(IV) Create LOIO Class

- Repeat the steps to create a PHIO class as previously described.
- On the *Instance Attribs* tab page, after adding the standard attributes, choose the *Insert Row* icon and add the IO attribute which you created in Step III.
- Save and activate.

(V) Create a Connection Space

- In the *Document Modeling Workbench* hierarchy, right-click the *Connection Spaces* folder and choose *Create*. Enter a *Class Connection Space* and *Description*.
- Double-click the newly-created connection space, and add the following:
 - Select Insert Class and add PHIO class as Source class.
 - Select Insert Class and add LOGOBJECT as relationship class.
 - Select Insert Class and add LOIO class as Target class.



CONNECTION SPACE

Adding the Attachment RUIBB in FLUID

The Attachment RUIBB can be added to a Floorplan Manager application at any time. Depending on the floorplan and the application-specific views already embedded, you can position an attachment RUIBB component in one of the following ways:

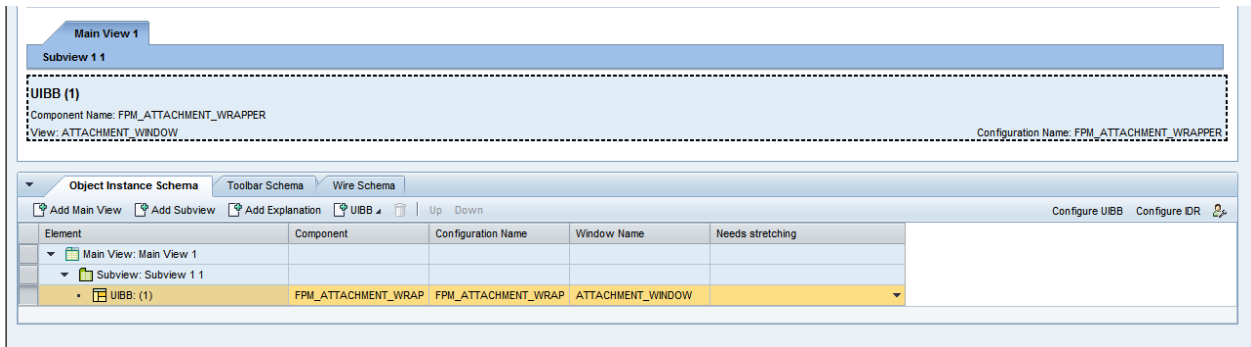
- in a subview of an object instance
- in a main step of a guided activity
- in a substep of a guided activity
- in a content area of an OVP
- in a tabbed component in a master UIBB
- in a tabbed component in a tab UIBB
- in a composite UIBB

Prerequisites

If the Attachment RUIBB needs to be configured, KPRO configuration should be complete (Refer to *KPRO Configuration for Attachment RUIBB*).

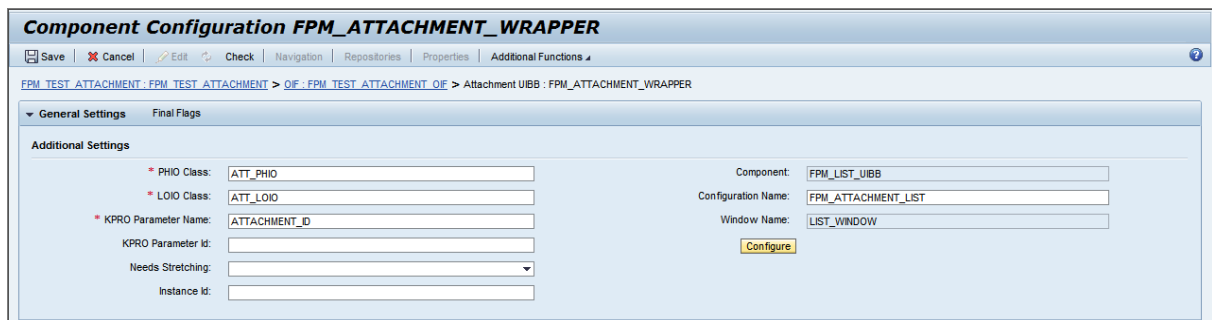
Procedure

1. In the *Preview* panel, select the place where you want to add the Attachment RUIBB.
2. If the attachment RUIBB configuration already exists, search for it in the *Repositories* panel and drag and drop it in the required position.
3. If a new UIBB needs to be added, choose *Add UIBB* in the floorplan schema tab.
4. Now specify the UIBB as an attachment component by entering the following values in the *Properties* panel or object instance schema:
 - For *Component*, enter **FPM_ATTACHMENT_WRAPPER**.
 - For *View*, enter **ATTACHMENT_WINDOW**.
5. Enter a configuration name.



CONFIGURING ATTACHMENT RUIBB

6. Choose *Configure UIBB*
7. Enter the following KPRO Parameters
 1. KPRO PHIO class - Mandatory
 2. KPRO LOIO class - Mandatory
 3. KPRO Parameter Name - Mandatory
 4. (Configured as IO attribute in KPRO DMWB)
 5. KPRO Parameter ID - Optional (It can be provided here or during runtime via FPM_SET_DOCUMENT_DETAIL event).
8. Default List Configuration(for Attachment RUIBB) - FPM_ATTACHMENT_LIST
 Default List feeder class (for Attachment RUIBB) - CL_FPMGB_ATTACHMENT



ATTACHMENT RUIBB CONFIGURATION

9. The default List configuration is FPM_ATTACHMENT_LIST. The standard functions provided by FPM are configured in FPM_ATTACHMENT_LIST. If the applications wants to add or delete features, then the application must specify its own list configuration.
10. If more than one attachment needs to be configured, enter the *Instance ID* in the List configuration.
11. If the application overrides the default list configuration(FPM_ATTACHMENT_LIST) with an application-specific configuration, the feeder class can be CL_FPMGB_ATTACHMENT or its subclass.
12. Applications can extend from the default feeder (CL_FPMGB_ATTACHMENT) and define their own feeder class to include application-specific functions.
13. Refer *Configure List GUIBB* in the FPM Cookbook for configuring application-specific List configuration.

Attachment RUIBB Features

The standard functions that are provided by FPM for the Attachment RUIBB are as follows:

- Add attachment
- Add URL/link
- Download an attachment
- Edit title of attachment/link
- Delete an attachment/link
- Replace an attachment/link
- Grouping based on *File Type, Created By, Changed By*

Note the following information:

- CL_FPMGB_ATTACHMENT is the standard feeder class for the Attachment RUIBB. This is a List feeder class which implements the IF_FPM_GUIBB_LIST interface.
- To set a unique Object ID (KPRO parameter ID), applications can raise the IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attributes-fpm_set_attachment_detail event, and set the ID in the FPM_DOCUMENT_ID parameter of the event.
- The attachments/links are stored in KPRO.
- Once the attachment/link is added, choose *OK* in the *Add Attachment/Link* dialog box. The attachment is saved to the KPRO.
- If the application requires a change in the standard behavior or needs to add or remove features, the application must configure a list component and specify an application-specific feeder class, which extends from the standard feeder class (CL_FPMGB_ATTACHMENT).
- The interaction between the Attachment RUIBB and the dialog box views is carried out by the use of events.
- The dialog box cannot be modified by the application. If the application needs to change the dialog boxes, it can define its own dialog boxes and handle them through its feeder class (which inherits the standard feeder class, CL_FPMGB_ATTACHMENT).
- By default, there are no authorization checks in the standard feeder. An application can implement them by extending the default feeder class, CL_FPMGB_ATTACHMENT.

The standard Attachment RUIBB is shown in the screenshot below:

Attachments								
Group By: None Attachment Link Download								
Actions	Name	File Name	File Size (KB)	File Type	Created By	Created On	Changed By	Changed On
	Driving	3.txt	1	Simple Text	DEVARAJANB	26.10.2010 10:08:40	DEVARAJANB	27.10.2010 08:41:44
	SAP AG	www.sap.com		Hypertext Markup Language (HTML)	DEVARAJANB	26.10.2010 10:13:54	KOENIGSTEIN	27.10.2010 11:54:31
	Image	image41815.jpg		Hypertext Markup Language (HTML)	DEVARAJANB	27.10.2010 08:37:25	DEVARAJANB	27.10.2010 08:37:25
	Ein Stern1	Ein Stern.M4A	1.231	audio/mp4	AMNRR	28.10.2010 08:39:11	AMNRR	28.10.2010 08:39:11
	www.google.com	HSDD_SupplierManagement-Takt2-V1.doc	3.080	Microsoft Word	AMNRR	28.10.2010 08:43:45	AMNRR	28.10.2010 08:43:45

Note the following points:

- The *File Type* and the icons in the *Name* column are based on the mime type of the attachment.
- *Edit Name, Delete Attachment/Link* and *Replace Attachment/Link* are available as row actions.
- The *Download* option is available only for the attachment and not for the link.
- *File Size* is not applicable for a link.
- If the *File Size* is less than 1 KB, it is rounded off to 1 KB and displayed as such.

User Actions at Runtime

You can perform the following actions at runtime on the Attachment RUIBB:

- **Add an Attachment/Link**

Choose *Add Attachments* to display the *New Attachment* or *New Link* dialog box. The dialog box cannot be modified by the application. If the application needs to change the dialog box, they can define their own dialog boxes and handle them through their feeder class (which inherits the standard feeder, CL_FPMGB_ATTACHMENT).

Note that KPRO has a file size restriction of 2GB per file.

Choose *OK* to add the attachment or link to the KPRO. When the attachment or link is added to the KPRO, the following attributes are stored as either PHIO (physical object) or LOIO (logical object) properties:

<i>Attribute</i>	<i>Type of Property</i>
Storage Category	PHIO
Language	PHIO
Description	PHIO
User-Specified Attribute (KPRO Attribute Name)	LOIO
Language	LOIO
Description	LOIO

Once the new attachment or link is added to the KPRO, it can be opened by choosing the link present in the *Name* column.

- **Edit Attachment/Link Name**

You edit the attachment/link name through the row action, *Edit*. The new name is specified in the *Edit Name* dialog box.

In the KPRO, the PHIO property description is modified according to the new name specified. The LOIO property *Description* will still have the original name specified during the time of the attachment/link creation.

- **Delete Attachment/Link**

You delete the attachment/link through the row action, *Delete*. In the confirmation dialog box, choose *OK* to delete the attachment/link.

In the KPRO, both the LOIO class and the PHIO class are deleted.

- **Replace Attachment/Link**

You replace the attachment/link through the row action, *Replace*. The *Replace Attachment* or *Replace Link* dialog box opens and you specify the new attachment/link name.

If the original file name and the replacement filename are different, a warning dialog box is displayed and the user must confirm that the attachment/link is to be replaced.

In the KPRO, the LOIO remains the same and the PHIO is deleted. A new PHIO is created for the replacement attachment/link.

- **Download**

The *Download* option is applicable for attachments only. Choose *Download* and the user has the option to open or save the attachment.

- **Group By**

Group By has the following options to group attachments:

- File Type
- Created By
- Changed By
- None (No Grouping)

- **Sorting and Filtering**

Sorting and filtering is available on all columns except the *Actions* column.

Events

The following table details events that are triggered during runtime:

Event Name	Constant	Description
FPM_SET_DOCUMENT_DETAIL	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_set_attachment_detail (The Parameter ID must be set in FPM_DOCUMENT_ID as an event parameter)	Event used for setting the unique object ID dynamically
ADD_ATTACHMENT	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_add_attachment	Event triggered on click of <i>New Attachment</i> button
OPEN_ATTACHMENT_POPUP	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_open_attachment	Event triggered for opening a <i>New Attachment</i> dialog box
FPM_NEW_ATTACHMENT	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_new_attachment	Event triggered for adding new attachment
ADD_LINK	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_add_link	Event triggered on click of <i>New Link</i> button
OPEN_LINK_POPUP	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_open_link	Event triggered for opening a <i>New Link</i> dialog box
FPM_NEW_LINK	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_new_link	Event triggered for adding new link
ROW_ACTION_EDIT_TITLE	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_row_edit_title	Event triggered on click of row action <i>Edit</i>
CHANGE_TITLE_POPUP	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_open_edit_title	Event triggered for opening <i>Change Title</i> dialog box
FPM_NEW_TITLE	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_new_title	Event triggered for editing the title
ROW_ACTION_REPLACE	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_row_replace	Event triggered on click of row action <i>Replace</i>

REPLACE_ATTACHMENT	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_replace_event	Event triggered for opening replace attachment/link dialog box
FPM_REPLACE_ATTACHMENT	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_replace_attachment	Event triggered for replacing the attachment/Link
REPLACE_CHECK_POPUP	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_replace_check	Event triggered for opening replacement confirmation dialog box
FPM_CONFIRM_REPLACE	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_confirm_replace	Event triggered for confirming the replacement of attachment/Link with a different file
ROW_ACTION_DELETE	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_row_delete	Event triggered on click of row action Delete
DOWNLOAD	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_attachment_download	Event triggered on click of <i>Download</i>
GROUP	IF_FPM_RUIBB_constants=>gc_ruibb_attachment_attri butes-fpm_attachment_group	Event triggered on click of <i>Group By</i>

Notes RUIBB

RUIBB Interface

FPM provides a marker interface, `IF_FPM_RUIBB`, which must be implemented by the WD components.

This interface can also be used by other applications, to turn a UIBB into a RUIBB.

For the Notes RUIBB, the `IF_FPM_RUIBB` interface is implemented by the notes 'wrapper' component `FPM_NOTES_WRAPPER`.

KPRO Configuration for Notes RUIBB

KPRO provides the functionality to store the Notes Contents. This is the basic KPRO configuration required for Notes RUIBB. It does not cover all aspects of KPRO.

5. Create a *Content Table*

In transaction SE11, you can copy a row from table SDOKCONT1, for example FPM_T_NOTES.

6. Create a *Content Repository*

Open transaction OAC0, create a new content repository and specify the following details (note that the table below shows only sample details):

Field	Value
Content Rep.	FPM_NOTES
Description	FPM Notes
Storage Type	SAP System Database
Rep. Sub-Type	Normal
Version No.	0046
Contents Table	FPM_T_NOTES

7. Create a Category

In transaction OACT, enter the category name and the content repository you created in the previous step.

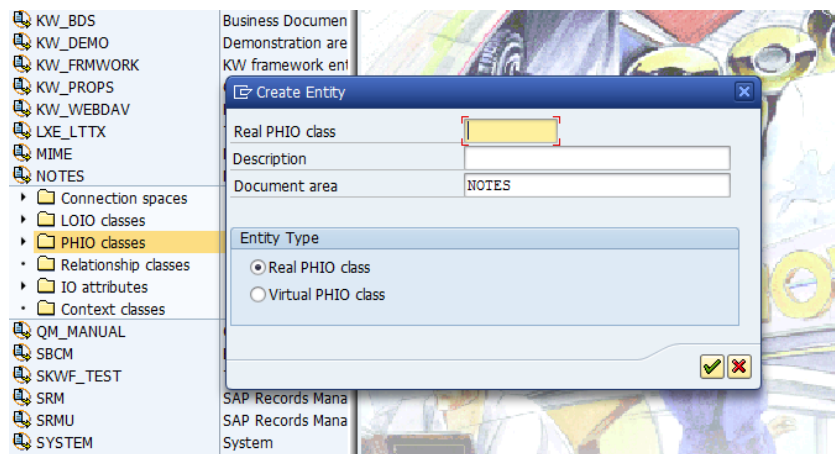
8. KPRO Modeling

(II) Create a Document Area

In transaction DMWB, choose *Create* to display the *Create Entity* dialog box. Select the *Document Area* radio button and specify the *Document area* and *Description*. Save and activate the Document area

(III) Create a PHIO Class

- In the *Document Modeling Workbench* hierarchy, expand the folder for the document area you have just created. In the context menu for *PHIO classes*, select the *Create* option.
- Select *Real PHIO Class* and enter the PHIO class and description.



Creating a PHIO Class

- In the *Document Modeling Workbench* hierarchy, double-click the newly created PHIO class and, on the *Standard Attributes* tab page, specify the following attributes:

Attribute	Value
Auto_Index	X

Storage_category	Specify the category you created in the previous step and mark it as <i>Exposed</i>
------------------	---

PHIO PROPERTIES

- Choose the *Tabulation* button.
- Choose the *Copy Table Set* button.
- Specify the *Name Prefix for Copy Tables* and execute. Choose *OK* for all the information dialog boxes that appear.
- In the *Document Modeling Workbench* hierarchy, double-click on the PHIO class. On the *Instance Attribs* tab page, choose the *Standard* button and select all the attributes and choose *OK*.

Real PHIO class: NOTES_PHIO Active

Description: Notes PHIO

Document area: NOTES

Package: APB_FPM_TEST

Created By: DEVARAJANB Last Changed By: RAJASS

Time Created: 08.09.2010 13:41:37 Last Changed At: 29.10.2010 12:13:53

Tabulation

Standard Attributes Free Attributes Instance Attribs

Attribute	Attribute Value	Exposed	Required	Hidden	Maint. ...
AUTHORITY_CHECK_FUNCTION		<input type="checkbox"/>			
AUTO_INDEX	X	<input type="checkbox"/>			
BUFFER_EXPIRATION		<input type="checkbox"/>			
DELETE_FUNCTION		<input type="checkbox"/>			
DOCUMENT_PROTECTION		<input type="checkbox"/>			
DYN_DELETE_FUNCTION		<input type="checkbox"/>			
DYN_GET_FUNCTION		<input type="checkbox"/>			
DYN_INIT_FUNCTION		<input type="checkbox"/>			
DYN_PURGE_FUNCTION		<input type="checkbox"/>			
DYN_QUERY_FUNCTION		<input type="checkbox"/>			
DYN_SET_FUNCTION		<input type="checkbox"/>			
EXPORT_FUNCTION		<input type="checkbox"/>			
IMPORT_FUNCTION		<input type="checkbox"/>			
NO_BUFFER		<input type="checkbox"/>			
PROPERTY_FUNCTION		<input type="checkbox"/>			
STORAGE_CATEGORY	NOTES	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
STORAGE_CATEGORY_FOR_URLS		<input type="checkbox"/>			
STORAGE_CATEGORY_MAINT		<input type="checkbox"/>			
TRANSLTN_CHECK_FUNCTION		<input type="checkbox"/>			
TRANSPORT_FUNCTION		<input type="checkbox"/>			
VERSION_TYPE		<input type="checkbox"/>			
VIEW_FUNCTION		<input type="checkbox"/>			

Instance Attributes for the PHIO

- Save and activate.

(III) Create IO Attribute

- In your document area inside the *Document Modeling Workbench* hierarchy, choose *Create* in the context menu for the *IO Attributes* folder.
- Enter the *IO Attribute* and *Description* and choose *OK*.
- Double-click on the newly created IO attribute and enter the attribute values for *Field_Name* and *Table_Name* (see screenshot below).

IO attribute	DOC_ID	Active
Description	Doc Id	
Document area	NOTES	
Package	APB_FPM_TEST	
Created By	DEVARAJAMB	Last Changed By
Time Created	08.09.2010 13:44:07	Last Changed At
		26.11.2010 09:35:35
		DDIC

Attribute	Attribute Value
CLIENT_TRANSFER_F4	
FIELD_NAME	PROP08
MODEL_EXTENSION	
PROPERTY_CHECK	
TABLE_NAME	FPMNOTESPHIO

IO Attributes

- Save and activate.

(VI) Create LOIO Class

- Repeat the steps to create a PHIO class as previously described.
- On the *Instance Attribs* tab page, after adding the standard attributes, choose the *Insert Row* icon and add the IO attribute which you created in Step III.
- Save and activate.

(VII) Create a Connection Space

- In the *Document Modeling Workbench* hierarchy, right-click the *Connection Spaces* folder and choose *Create*. Enter a *Class Connection Space* and *Description*.
- Double-click the newly-created connection space, and add the following:
 - Select *Insert Class* and add PHIO class as Source class.
 - Select *Insert Class* and add LOGOBJECT as relationship class.
 - Select *Insert Class* and add LOIO class as Target class.

Class connection space	NOTES_CONNECTION	Active
Description	Notes connection	
Document area	NOTES	
Package	APB_FPM_TEST	

Source Class	Relationship Class	Target Class
NOTES_PHIO	LOGOBJECT	< NOTES_LOIO >

CONNECTION SPACE

Adding the Notes RUIBB in FLUID

The Notes RUIBB can be added to a Floorplan Manager application at any time. Depending on the floorplan and the application-specific views already embedded, you can position a notes RUIBB 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 content area of an OVP
- in a tabbed component in a master UIBB
- in a tabbed component in a tab UIBB
- in a composite UIBB

Prerequisites

If the Notes RUIBB needs to be configured, KPRO configuration should be complete (Refer to *KPRO Configuration for Notes RUIBB*).

Procedure

14. In the *Preview* panel, select the place where you want to add the Notes RUIBB.
15. If the notes RUIBB configuration already exists, search for it in the *Repositories* panel and drag and drop it in the required position.
16. If a new UIBB needs to be added, choose *Add UIBB* in the floorplan schema tab.
17. Now specify the UIBB as a notes component by entering the following values in the *Properties* panel or object instance schema:
 - For *Component*, enter FPM_NOTES_WRAPPER.
 - For *View*, enter NOTES_WINDOW.
18. Enter a configuration name.

Element	Component	Configuration Name	Window Name	Container Stretching
Main View: Main View 1				
Subview: Subview 1 1				
UIBB: (1)	FPM_NOTES_WRAPPER	FPM_NOTES_WRAPPER	NOTES_WINDOW	

CONFIGURING NOTES RUIBB

19. Choose *Configure UIBB*
20. Enter the following KPRO Parameters
 1. KPRO PHIO class - Mandatory
 2. KPRO LOIO class - Mandatory

3. KPRO Parameter Name - Mandatory
 4. (Configured as IO attribute in KPRO DMWB)
 5. KPRO Parameter ID - Optional (It can be provided here or during runtime via FPM_SET_DOCUMENT_DETAIL event).
21. Default List Configuration(for Notes RUIBB) - FPM_NOTES_LIST
 Default List feeder class (for Notes RUIBB) - CL_FPMGB_NOTES

NOTES RUIBB CONFIGURATION

22. The default List configuration is FPM_NOTES_LIST. The standard functions provided by FPM are configured in FPM_NOTES_LIST. If the applications wants to add or delete features, then the application must specify its own list configuration.
23. If more than one notes uibb needs to be configured, enter the *Instance ID* in the List configuration.
24. If the application overrides the default list configuration(FPM_NOTES_LIST) with an application-specific configuration, the feeder class can be CL_FPMGB_NOTES or its subclass.
25. Applications can extend from the default feeder (CL_FPMGB_NOTES) and define their own feeder class to include application-specific functions.
26. Refer *Configure List GUIBB* in the FPM Cookbook for configuring application-specific List configuration.

Notes RUIBB Features

The standard functions that are provided by FPM for the Notes RUIBB are as follows:

- Add notes
- Display notes
- Edit notes
- Delete notes

Note the following information:

- CL_FPMGB_NOTES is the standard feeder class for the Notes RUIBB. This is a List feeder class which implements the `IF_FPM_GUIBB_LIST` interface.
- To set a unique Object ID (KPRO parameter ID), applications can raise the `IF_FPM_RUIBB_constants=>gc_ruibb_notes_attributes-fpm_set_notes_detail` event, and set the ID in the **FPM_DOCUMENT_ID** parameter of the event.
- The Notes are stored in KPRO on triggering the FPM_SAVE event only. All the actions (Add/Delete/Edit) in KPRO are triggered on FPM_SAVE event.
- Once the Notes name, type and content are entered in the Dialog, choose *OK* in the *New Notes* dialog box. The note is temporarily stored and will be displayed in the UI.

- If the application requires a change in the standard behavior or needs to add or remove features, the application must configure a list component and specify an application-specific feeder class, which extends from the standard feeder class (CL_FPMGB_NOTES).
- The interaction between the Notes RUIBB and the dialog box views is carried out by the use of events.
- The dialog box cannot be modified by the application. If the application needs to change the dialog boxes, it can define its own dialog boxes and handle them through its feeder class (which inherits the standard feeder class, CL_FPMGB_NOTES).
- By default, there are no authorization checks in the standard feeder. An application can implement them by extending the default feeder class, CL_FPMGB_NOTES.

The standard Notes RUIBB is shown in the screenshot below:

Actions	Name	Text	Type	Created By	Created On	Changed By	Changed On
	SHOPPING CART	Shopping Cart # 3443534 Created	Customer Note	RAJASS	11.11.2010 09:25:16	RAJASS	26.11.2010 11:14:13
	MATERIAL # 56312	Material #23432 has to be tested with high temp	Customer Note	AMNR	11.11.2010 10:02:26	RAJASS	26.11.2010 11:14:44
	PRICE LIST	Please update the price list for all the materials	Supplier Note	RAJASS	11.11.2010 11:08:30	RAJASS	26.11.2010 11:15:28
	FILE0001	File is missing	Test Note	KSNR	12.11.2010 07:43:18	RAJASS	26.11.2010 11:15:57
	IMPORTANT	Please update the shopping cart with new material list	Test Note	AMNR	12.11.2010 08:37:47	RAJASS	26.11.2010 11:16:45

Note the following points:

- *Edit Notes* and *Delete Notes* are available as row actions.

User Actions at Runtime

You can perform the following actions at runtime on the Notes RUIBB:

- **Add a Note**

Choose *New* to display the *New Notes* dialog box. The dialog box cannot be modified by the application. If the application needs to change the dialog box, they can define their own dialog boxes and handle them through their feeder class (which inherits the standard feeder, CL_FPMGB_NOTES).

Choose *OK* to add the note to the view. When the note is added to the KPRO on FPM_SAVE event, the following attributes are stored as either PHIO (physical object) or LOIO (logical object) properties:

<i>Attribute</i>	<i>Type of Property</i>
Storage Category	PHIO

Language	PHIO
Description	PHIO
User-Specified Attribute (KPRO Attribute Name)	LOIO
Language	LOIO
Description	LOIO

Once the new note is added to the KPRO, it can be opened by choosing the link present in the *Text* column.

- **Edit Notes**

You edit the note name/content/type through the row action, *Edit*.

On FPM_SAVE event, in KPRO, both the LOIO and PHIO class information are modified according to the new note changes specified.

- **Delete Notes**

You delete the note through the row action, *Delete*.

On FPM_SAVE event, in KPRO, both the LOIO class and the PHIO class are deleted.

- **Display Notes**

You can click the note text link to display the note in a dialog *Display Notes*. The dialog box is displayed using the *OK* button (which can also be used to close the dialog box).

- **Sorting and Filtering**

Sorting and filtering is available on all columns except the *Actions* column.

Events

The following table details events that are triggered during runtime:

Event Name	Constant	Description
FPM_SET_DOCUMENT_DETAIL	IF_FPM_RUIBB_constants=>gc_ruibb_notes_attributes-fpm_set_notes_detail (The Parameter ID must be set in FPM_DOCUMENT_ID as an event parameter)	Event used for setting the unique object ID dynamically
ADD_NOTES	IF_FPM_RUIBB_constants=>gc_ruibb_notes_attributes-fpm_add_notes	Event triggered on click of <i>New</i> button
OPEN_NOTES_POPUP	IF_FPM_RUIBB_constants=>gc_ruibb_notes_attributes-fpm_open_notes	Event triggered for opening a <i>New Notes</i> dialog box
FPM_NEW_NOTES	IF_FPM_RUIBB_constants=>gc_ruibb_notes_attributes-fpm_new_notes	Event triggered for adding new notes
DISPLAY_NOTES	IF_FPM_RUIBB_constants=>gc_ruibb_notes_attributes-fpm_display_event	Event triggered for displaying notes
FPM_EDIT_NOTES	IF_FPM_RUIBB_constants=>gc_ruibb_notes_attributes-fpm_edit_notes	Event triggered for editing notes
ROW_ACTION_EDIT_TITLE	IF_FPM_RUIBB_constants=>gc_ruibb_notes_attributes-fpm_row_edit_title	Event triggered on click of row action <i>Edit</i> to show Edit dialog box
ROW_ACTION_DELETE	IF_FPM_RUIBB_constants=>gc_ruibb_notes_attributes-fpm_row_delete	Event triggered on click of row action <i>Delete</i>
FPM_GUIBB_LIST_CELL_ACTION		Event triggered on click of cell action to show the display dialog box.

Value/Input Helps for Generic UIBBs (GUIBBs)

It is possible to assign value helps to fields belonging to GUIBB components.

You can use the following options to assign value helps (arranged in order of precedence):

1. DDIC value helps assigned in the field description
2. OVS (Object Value Selector) assigned in the field description
3. Freestyle value helps assigned in the field description

4. Fixed values assigned in the field description
5. DDIC value helps assigned to a DDIC structure used as includes in the field catalog (nesting is possible).
6. DDIC value helps assigned to the data element used in the field catalog
7. Fixed values from the domain assigned in the data element used in the field catalog
8. Input help for data types DATS and TIMS.



If more than one of these options is assigned to a field, the one highest in the above list has priority.

Assignments in the Field Description

The assignments in the field description are described below:

DDIC Value Help

A DDIC value help can be assigned by the feeder class in the field description in field `DDIC_SHLP_NAME`.

OVS

Whenever a DDIC search help is not applicable, you should consider the OVS mechanism. OVS offers a generic UI like the DDIC value help. However, you must offer the selection logic. To do this, you must provide the name of a class implementing `IF_FPM_GUIBB_OVS` in the field description in field `OVS_NAME`. If the specified name is the same as the feeder class name, then the feeder class instance will be used, otherwise this class will be instantiated whenever it is needed.

This interface offers 4 methods:

- `HANDLE_PHASE_0` where the OVS popup can be “configured”
- `HANDLE_PHASE_1` to define the selection fields
- `HANDLE_PHASE_2` where the result list must be determined
- `HANDLE_PHASE_3` where the selected result list entry is passed back to the input field

In order to fulfill these tasks, an instance of `IF_WD_OVS` is passed as an importing parameter to these methods.

Freestyle Value Help

A freestyle value help is a WD ABAP component implementing the WD interface `IWD_VALUE_HELP`. The name of a freestyle value help component can be assigned by the feeder class in the field description in field `WD_VALUE_HELP`. The attribute values can be accessed via the context element attached to the value help listener. The context attribute names are the same as the field names of the field catalog except for the search attributes of the Search UIBB. Additional information can be provided if the freestyle component also implements the WD interface `IF_FPM_GUIBB_VH`. In method `INITIALIZE_GUIBB_VH`, the following parameters are passed:

- `IS_INSTANCE_KEY`: The instance key of the GUIBB
- `IV_FIELDNAME`: The field name from the field catalog of the value help field
- `IO_SEARCH_ATTR_READ`: An access interface providing the current search criteria

Fixed Values

A set of fixed values can be assigned by the feeder class in the field description in field `FIXED_VALUES`.

Drag-and-Dropping Data between UIBBs

You can use the *Drag-and-Drop* feature 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.

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 feeder class method `GET_DATA` 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 FPM, FLUID, 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.

Component Configuration FPM_TEST_DRAG_LIST_UIBB

Drag List GUIBB

Change Check New Window Deep-Copy Refresh View: Component-Defined

FPM_TEST_DRAG_AND_DROP_IN_LIST: FPM_TEST_DRAG_AND_DROP_IN_LIST > OIF: FPM_TEST_DRAG_AND_DROP_OIF > List UIBB: FPM_TEST_DRAG_LIST_UIBB

Hierarchy

Expand All Collapse All

Element

- List
 - Toolbar
 - Column: Airline
 - Column: Connection Number
 - Column: Airline local curre...
 - Column: Flight Date
 - Column: Client
 - Column: Booking total
 - Column: Plane Type
 - Settings

Airline	Connection Number	Airline local currency	Flight Date	Client	Booking total	Plane Type
0000					0.00	
0000					0.00	
0000					0.00	
0000					0.00	
0000					0.00	

Attributes List

Feeder Class: CL_FPM_DRAG_LIST_UIBB

Lead Selection Action Assignment: Assign Action Unassign Action

Drag and Drop Attributes:

Enable Drag Tags: TEST* Scope: Global

Enable Drop Tags: TESTDROP Scope: Global

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:

GET_DEFINITION	
Method of the feeder class. Allows the feeder class to provide all necessary information for configuring <i>Drag-and-Drop</i> via the parameter <code>et_dnd_definition</code> .	
<code>ET_DND_DEFINITION</code>	This attribute is of type structure <code>FPMGB_S_DND_DEFINITION</code> which defines the drag and drop attributes. The drag and drop has to be defined separately using the attribute <code>TYPE</code> value <code>DRAG</code> or <code>DROP</code> .
GET_DATA	
Method of the feeder class. Allows the feeder class to provide all necessary information for overriding drag-and-drop attributes via the parameter <code>ct_dnd_definition</code>	

CT_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.	
IS_DROP_INFO	This attribute sets the drop information of the GUIBB from ON_DROP_ACTION method.
IO_COMP	This attribute sets the source UIBB configuration details (Config ID + Config type + Config variant).
IV_COMP_NAME	This attribute sets the source UIBB Component name.

SET_FPM_DROP_DATA	
This method is called from the FLUSH method of the UIBB to set the drag and drop data.	
IO_DATA	This attribute has a reference to the data from where the drag occurred.
IO_EVENT	This attribute sets the event FPM_DROP_COMPLETED to which the dragged data has to be set.
IT_INDICES	This attribute sets the selected (dragged) data.
IV_DROP_POSITION (OPTIONAL)	This attribute sets the drop position in the drop target.
IV_CONFIG_KEY (OPTIONAL)	This attribute sets the drag source config key to identify the source.

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 `SET_FPM_DROP_DATA` method.

Dynamically Changing Drag-and-Drop

Drag-and-Drop attributes can be overridden at runtime if the *Override at Runtime* field is selected in the configuration editor.

For GUIBBs, the *drag-and-drop* attributes can be changed at runtime in the `GET_DATA` method of the feeder class using the parameter `CT_DND_ATTRIBUTES`; inform FPM using the parameter `EV_DND_ATTR_CHANGED`. This parameter can be set on any events or conditions.

For Freestyle UIBBs, the *drag-and-drop* attributes can be changed at runtime as determined by Web Dynpro.

Context Based Adaptations (CBA)

The option to adapt applications without the need of modification is one of the main benefits offered by FPM. The possibilities of Web Dynpro Customizing in combination with the Enhancement Framework already allow far more than the classic Dynpro in this respect.

Nevertheless, these two options still have quite strong limitations: There is only one customizing per client and Enhancements are system-wide active. This allows adapting applications to the need of a customer, but not to the need of different contexts within one customer system.

Use-Case Example

It is quite simple to adapt the screen below by customizing or enhancing. The customer can rearrange or hide fields; he can also add additional fields. However, he must also decide how this screen should look like for all situations. He cannot adapt it depending on some runtime parameter.

Purchase Order: 30000041, Vente Et Réparation de Ordinateur

Save | Cancel | New | Create Follow-Up

Purchase Order Details Edit

General Data

Purchase Order ID: 30000041

Employee Responsible: Klaus Stadthanner

Processing Data

Lifecycle Status: New

Approval Status: Initial

Confirmation Status: Initial

Delivered:

Dates

Created: 12.07.2010 14:54 CET

Supplier

* Business Partner ID: 10000026 Vente Et Réparation de Ordina

Street/ House Number: Rue Lourmel 42

ZIP Code/ City: 75013 Paris

Country: FR France [Show on Map](#)

Value

Total Net Amount: 1.667,00 EUR

Total Tax Amount: 16,67 EUR

Total Gross Amount: 1.683,67 EUR

Items Add

Insert

Actions	Purchase Or...	Product ID	Description	Quantity	Quantity Unit
	10	HT-1007	Ultra fast 3G UMTS/HSDPA Pocket PC, can be used with any GSM network in the w...	2,000	EA
	20	HT-1102	Complete package, 1 User, Network Software Utilities, Useful Applications and Docu...	1,000	EA
	30	HT-2000	7" LCD Screen, storage battery holds up to 6 hours!	2,000	EA

There could be the requirement to add additional information to (or remove it from) the screen for particular users (in our example, it has been decided to remove the details assignment block and some other fields to avoid unnecessary load on managers).

Previously, there was only one option to achieve this: copy the whole set of configurations, adapt it and have the particular users start the adapted configurations.

There was also another problem, that is, the need for data-dependent adaptations. In the previous screen, the *Street/House Number* fields are placed in the right order for German addresses. In France, house number and street name are the other way around. Therefore, an adaptation for French addresses would simply swap those two fields around. However, previously, this could only be realized in a very cumbersome way: The form configuration must be replaced at runtime via AppCC-coding.

The new functionality allows a far better and more comfortable approach to implement these use-cases.

Basic Concepts

Adaptation Schema

The adaptation schema is simply a list of characteristics (or 'dimensions') which can be used for adaptations. In the previous example, there is the need for two characteristics: role and country. The adaptation schema is created using SM30 views `FPM_V_ADAPT_SCHM` and `FPM_V_ADAPT_DIM`.

Adaptation Dimension

The adaptation dimension represents an individual characteristic within an adaptation schema. It is maintained in view `FPM_V_ADAPT_DIM`. A dimension is defined by a name for identification, an index and a data element. The index is used to determine the dominating adaptation dimension in case there is a collision. The data element is only used for the design-time allowing to you to provide field and F4 help while configuring context-based adaptations

Adaptation Context

The adaptation context is a set of values for a given adaptation schema. So, if the adaptation schema `TEST` consists of the dimensions `ROLE` and `COUNTRY`, a sample adaptation context would be `ROLE = MANAGER` and `COUNTRY = FRANCE`.



The term 'context' in this document is completely independent of the Web Dynpro term 'context'. To avoid misunderstandings, the term 'adaptation context' is used.

Inheritance of Component Configurations

Context-based adaptations use the new inheritance concept of component configurations. Each adapted configuration is represented by a *derived* configuration. A derived configuration contains only the delta to its base configuration.

Step-By-Step Example

In the first part of this step-by-step example we will extend the existing WD demo application `S_EPM_FPM_PO` by adding an Attachment UIBB to the overview page. This additional UIBB should only be available for Managers. As this can be done without any modification of the existing application, there is no need to create a copy first.

In the second part of this example we will exchange street and house number for the French address format.

Adding an Attachment UIBB for Managers

1. Create the Adaptation Schema

As we want to adapt the application with respect to role (only Managers should see the attachment UIBB) and country (French addresses need a different field order), we need to create an adaptation schema having the two dimensions, *Role* and *Country*.

As this adaptation schema already exists (`TEST_FPM`), you can use this and skip the following section.

- a. Launch transaction SM30 and enter `FPM_V_ADAPT_SCHM` as *Table/View*.

- b. Choose Create for a new adaptation schema.
- c. Edit `FPM_V_ADAPT_DIM` again with SM30 and create the two dimensions needed (see screenshot below).

Adapt. Dim...	Index of Dim...	Data element
COUNTRY	2	LAND1
ROLE	1	AGR_NAME

An adaptation schema should normally be valid throughout a whole application area; there is no need to create an individual schema for every application.

2. Create the WD Application

In order to run `S_EPM_FPM_PO` as an adaptable configuration, a new WD application must be created:

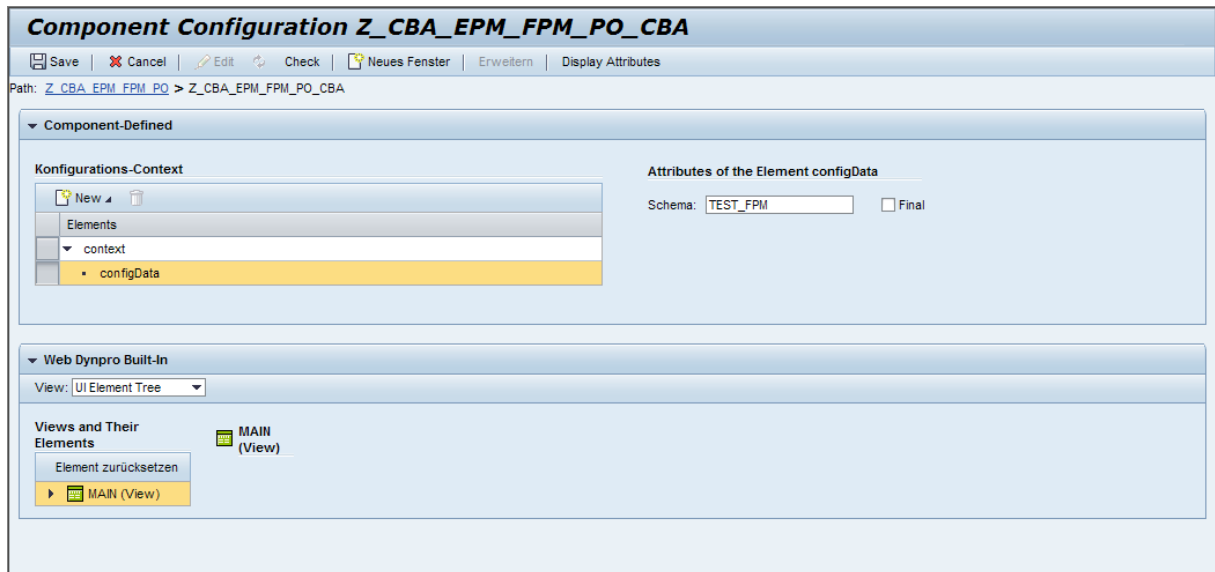
- a. Open SE80 and create a new WD application (e.g. `Z_CBA_EPM_FPM_PO`). When you create non-adaptable FPM applications you use one of the `FPM_<floorplan ID>_COMPONENT` components for your application. For adaptable applications, you use `FPM_ADAPTABLE_<floorplan ID>` instead.
- b. Therefore, as `S_EPM_FPM_PO` is an application using the OVP floorplan, you must create the new application using the following settings:

- Component `FPM_ADAPTABLE_OVP`
- Interface View `FPM_WINDOW`
- Plug Name `DEFAULT`

Application	Z_CBA_EPM_FPM_PO	New
<div style="display: flex; justify-content: space-between;"> Properties Parameters </div>		
Description	CBA Step-by-Step Example	
Component	FPM_ADAPTABLE_OVP	
Interface View	FPM_WINDOW	
Plug Name	DEFAULT	
Help Links		
Authorization Check		
<input checked="" type="radio"/> Check for Application <input type="radio"/> Check for Application and Application Configuration		
Handling of Messages		
<input checked="" type="radio"/> Show Message Component on Demand <input type="radio"/> Always Display Message Component		
Administrative Data		
Created by	GUENTHERC	Created on
		23.11.2010
Last Changed by		Changed on
Package		
Language		
URL	https://uxciqt6.wdf.sap.corp:44312/sap/bc/webdynpro/sap/z_cba_epm_fpm...	

- c. To be able to set the adaptation context of your application via URL parameters or via application configuration, add the relevant adaptation dimensions as application parameters (see following screenshot).

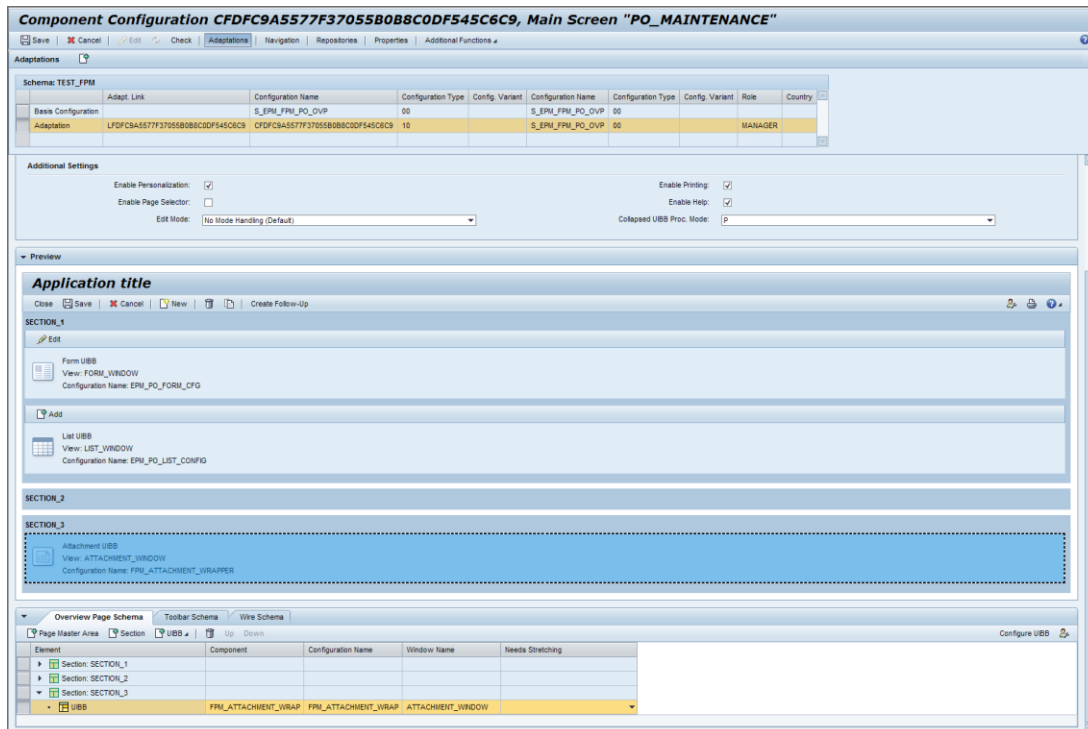
- a. There is only one attribute which has to be maintained within this configuration, the *Adaptation Schema*, under the context node *configData* (See screenshot below). Enter the name of your adaptation schema and save the configuration.



Now all the preparation work is done. If you run the new application it should look exactly like the original.

5. Create an Adaptation

- a. Call up your new application configuration again and navigate from there to the OVP component configuration. You should now see a new *Adaptations* toggle-button in the main toolbar; choose it to switch on the adaptation panel.
- b. One entry already exists in the *Adaptations* panel. This is the base configuration which you now can adapt.
- c. To create an adaptation for the *Manager* role, choose the *Add Adaptation* button and enter **MANAGER** as the Role in the dialog box that appears (leave *Country* empty). After entering package details, you will have another entry in the adaptation list. Select it and switch to edit mode.
- d. Now add a new UIBB to the page and save it (see example in the following screenshot):



If you run the application now, it is still unchanged. This is because there is still no adaptation context set. However, the application is now ready to show the additional UIBB when the adaptation context is set to role = MANAGER.

This can now easily be achieved by adding the relevant adaptation context as a URL parameter. In the URL of the application, add `&ROLE=MANAGER`, refresh the screen; you should now see that the application has been adapted and the attachment UIBB is displayed in the bottom of the overview page.

Before going on to the next part of this step-by-step example, let's have a quick summary:

We started with a delivered standard application and adapted it. We had to work through a couple of steps but we did not have to enter additional code or modify existing objects; all the steps were purely configurative.

Adapting the Address Layout

In the second part of this example, we will demonstrate some dynamical adjustments of the screen layout based on the data to be displayed. As the chosen application is not yet prepared to support CBA, this second part will require a little bit of coding. As we are using the same application we can skip the steps 1-4 from the previous part and directly continue with preparing the application to support CBA.

1. Extending the Form's Feeder Class

The data in the original application's form is provided by the feeder class `CL_EPM_PO_FORM_FEEDER`. In order to make the form adaptable (with regard to country), the feeder class must communicate the country of the data to the CBA framework.

The original feeder class does not do this because it was created before CBA was available: since CBA is new, this applies to all feeder classes at the moment. Therefore, this is quite a common use-case.

To provide the CBA functionality without modifying the standard feeder class, we need a new feeder class. As we do not want to re-implement all its functionality, we sub-class it and only redefine the one method which needs to be adapted.

Therefore, open transaction SE24 and create a new class which inherits from CL_EPM_PO_FORM_FEEDER. The only method we redefine is the IF_FPM_GUIBB_FORM~GET_DATA method and we implement it using the following code:

```
method IF_FPM_GUIBB_FORM~GET_DATA.
* call the GET_DATA method of the standard feeder class
CALL METHOD SUPER->IF_FPM_GUIBB_FORM~GET_DATA
EXPORTING
    IO_EVENT                = io_event
    IT_SELECTED_FIELDS      = IT_SELECTED_FIELDS
    IV_RAISED_BY_OWN_UI    = IV_RAISED_BY_OWN_UI
    IV_EDIT_MODE            = IV_EDIT_MODE
IMPORTING
    ET_MESSAGES             = ET_MESSAGES
    EV_DATA_CHANGED        = EV_DATA_CHANGED
    EV_FIELD_USAGE_CHANGED = EV_FIELD_USAGE_CHANGED
    EV_ACTION_USAGE_CHANGED = EV_ACTION_USAGE_CHANGED
CHANGING
    CS_DATA                 = CS_DATA
    CT_FIELD_USAGE          = CT_FIELD_USAGE
    CT_ACTION_USAGE         = CT_ACTION_USAGE.

* Check event id to avoid infinite loop
check io_event->MV_EVENT_ID ne
    IF_FPM_CONSTANTS=>GC_EVENT-ADAPT_CONTEXT.

data: lo_fpm type ref to if_fpm,
      lo_event type ref to CL_FPM_EVENT.
field-symbols: <fs_country_code> type SNWD_COUNTRY.
* determine the country code from the data delivered by the standard feeder
ASSIGN COMPONENT 'COUNTRY_CODE' of STRUCTURE cs_data
    to <FS_COUNTRY_CODE>.

* create the object to set the adaptation context.
CREATE OBJECT LO_EVENT
EXPORTING
    IV_EVENT_ID            = IF_FPM_CONSTANTS=>GC_EVENT-ADAPT_CONTEXT
    IV_ADAPTS_CONTEXT     = abap_true.

* Set the adaptation context via event parameters
lo_event->MO_EVENT_DATA->SET_VALUE (
    exporting iv_key      = 'COUNTRY'
             iv_value    = <FS_COUNTRY_CODE> ).

* finally raise the event
lo_fpm = CL_FPM_FACTORY=>GET_INSTANCE ( ).
```

```

lo_fpm->RAISE_EVENT( lo_event ).
endmethod.

```

Let's now walk through the code step-by-step:

- a. First we call the `IF_FPM_GUIBB_FORM~GET_DATA` of the parent class, making sure that the standard logic and data retrieval is executed, so the logic of the feeder class remains unchanged. All we are doing here is adding a few additional lines of code to inform the CBA framework about the changed adaptation context.
- b. We then check that the current event is not an adaptation event. This is because setting the adaptation context is done by raising another FPM event, which then calls this method again. Omitting this check would cause an infinite loop.
- c. Next we determine the country from the data provided by the standard implementation.
- d. Finally we create an event to change the adaptation context, adding the country information as an event parameter and raising the event.

Note that this code is far from optimal; it is kept as simple as possible for example purposes and to demonstrate how CBA works. A better version will be provided in a later chapter.

2. Replacing the Standard Feeder Class

Now we will use a trick to replace the standard feeder class with our newly created one. To do this, create an adaptation of the original form configuration. Call up your new application configuration again. From there, navigate via the OVP component configuration to the Form UIBB configuration. Create a new adaptation for it. As adaptation context set `ROLE = *` and leave the *Country* value empty. Then replace the feeder class within the adapted configuration.

`ROLE = *` means this adaptation should be applied to all adaptation contexts, where no more specific adaptation exists. Since, at the moment, this is the only adaptation of the form, this adaptation will always be applied. Therefore, this is a simple trick to replace a feeder class without modifying any standard object.

These two steps are much easier if you want to prepare your own application for CBA. In that case, you would simply add the necessary lines of code to the standard feeder class application and that's it; there is no need to create your own feeder class and to replace the feeder class via an adaptation.

Applications which want to provide the adaptation functionality should add the necessary code to their standard code. This way, customers are enabled using CBA completely without these two more complex steps.

3. Creating an Adaptation for French Addresses

The last step is to create an adaptation of the form for the context `COUNTRY = FR`. So, we again navigate to the form's configuration via the application and the OVP component configuration. Create a new adaptation and set the adaptation context to `ROLE = *` and `COUNTRY = FR`.

Replace the feeder class and add your own, and swap the street and house number fields (see following screenshot):

Component Configuration C910C8D2AF5D836845A39269311DC20A

Save | Cancel | Edit | Check | Adaptations | Navigation | Repositories | Properties | Additional Functions

Adaptations

Schema: TEST_FPM	Adapt. Link	Configuration Name	Configuration Type	Config. Variant	Configuration Name	Configuration Type	Config. Variant	Role	Country
Basis Configuration		EPM_PO_FORM_CFG	00		EPM_PO_FORM_CFG	00			
Adaptation	L910C8D2AF5D836845A39269311DC20A	C910C8D2AF5D836845A39269311DC20A	10		EPM_PO_FORM_CFG	00		*	FR
Adaptation	LBC5ECDCF9CDA21E9329A633480281D9	CBC5ECDCF9CDA21E9329A633480281D9	10		EPM_PO_FORM_CFG	00		*	

Z_CBA_EPM_FPM_PO; Z_CBA_EPM_FPM_PO > OVP: S_EPM_FPM_PO_OVP > Form UBB: EPM_PO_FORM_CFG

General Settings

Preview

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	General Data								Supplier							
2	Purchase Order ID: <input type="text"/>								* Business Partner ID: <input type="text"/>							
3									Street/ House Number: <input type="text"/>							
4	Employee Responsible: <input type="text"/>								ZIP Code/ City: <input type="text"/>							
5	Processing Data								Country: <input type="text"/>							
6	Lifecycle Status: <input type="text"/>															
7	Approval Status: <input type="text"/>															
8	Confirmation Status: <input type="text"/>								Value							
9	Delivered: <input type="checkbox"/>								Total Net Amount: <input type="text"/> 0,00							
10	Dates								Total Tax Amount: <input type="text"/> 0,00							
11	Created: <input type="text"/> 00:00								Total Gross Amount: <input type="text"/> 0,00							
12																

Run the application. If you select a French supplier you should now see the adjusted configuration.

Now run your application again, but this time add `ROLE=MANAGER` to the URL. The result is probably no surprise: the additional attachment UIBB is displayed because you started the application as a manager, and for French addresses you get the right order for street and house number.

4. Enhancing the French Address Adaptation

As mentioned already, there is some room for improvement in this example. There are two issues with the current solution:

- The feeder class is currently firing an additional event to set the adaptation context at every FPM event. This means that the number of FPM roundtrips is doubled. Worse still, is if you assume that it should be possible to run the feeder class in an application which does not support CBA, then even in

these applications the number of FPM events is doubled. As this might lead to performance implications, this should be avoided.

- b. The adaptation context is set globally. This means it will be applied to all UIBBs on the screen. This is normally OK. However, there might be situations where another UIBB on the screen uses another adaptation context.

This is discussed further in the following two sections.

Avoiding Unnecessary FPM Events

This can of course be solved very easily. To avoid that this CBA-specific code is executed in applications which doesn't support CBA, you can simply check if `IF_FPM->MO_ADAPTATION_MANAGER` is bound. Add the following line of code directly after the call of the parent's `GET_DATA` method:

```
check CL_FPM_FACTORY=>GET_INSTANCE ( )->MO_ADAPTATION_MANAGER is bound.
```

The next thing to do is to limit the additional FPM events, needed for setting the adaptation context, to those situations where the relevant context has indeed changed. This is trivial: simply store the country value at every event and, in the next event, check if it has changed. Only trigger the adaptation event if there is a change. Add a private attribute `MV_COUNTRY_CODE` (type `SNWD_COUNTRY`) to your class and add the following lines of code before you create the event:

```
check MV_COUNTRY_CODE ne <FS_COUNTRY_CODE>.
  MV_COUNTRY_CODE = <FS_COUNTRY_CODE>.
```

Setting the Adaptation Context Locally

To set the adaptation context for a single UIBB only, it is possible to set it locally. This can be achieved by adding the UIBB instance key to the event and using an appropriate event.

Let's start with using an appropriate event. There are two options:

Either use `if_fpm_constants=>gc_event-ADAPT_CONTEXT_LOCAL` as the event ID or use any other event (except `if_fpm_constants=>gc_event-ADAPT_CONTEXT`) and set its `ADAPTS_CONTEXT` attribute to `TRUE`. If you then provide the UIBB's configuration key as an event parameter, `if_fpm_constants=>gc_event_param-source_config_id`, it is automatically treated as a local event.

But, in any case, for a local adaptation change we need the UIBB instance key. A simple solution would be to hardcode this. However, this would then only work for the specified configuration and when the feeder class is reused in a different application, this would not work. Therefore, we should try to determine the instance key at runtime.

This information is passed to the feeder class in the `IF_FPM_GUIBB~INITIALIZE` method. So again the solution is obvious: Redefine this method, store the UIBB instance information and then use it when firing the event.

Add another private attribute to your feeder: `MS_INSTANCE_KEY` type `FPM_S_CONFIG_KEY`.

Redefine method IF_FPM_GUIBB~INITIALIZE and add the following code:

```
method IF_FPM_GUIBB~INITIALIZE.

    CALL METHOD SUPER->IF_FPM_GUIBB~INITIALIZE
        EXPORTING
            IT_PARAMETER           = it_parameter
            IO_APP_PARAMETER       = io_app_parameter
            IV_COMPONENT_NAME      = iv_component_name
            IS_CONFIG_KEY          = is_config_key
            IV_INSTANCE_ID         = iv_instance_id.

    move-CORRESPONDING IS_CONFIG_KEY to MS_INSTANCE_KEY.
    MS_INSTANCE_KEY-INSTANCE_ID = IV_INSTANCE_ID.

endmethod.
```

Trigger the local adaptation event. To do this, enhance the coding of your GET_DATA method by firing the if_fpm_constants=>gc_event-ADAPT_CONTEXT_LOCAL event instead of the if_fpm_constants=>gc_event-ADAPT_CONTEXT event and add the instance key parameters to the event. In the end, the whole GET_DATA method should look like this:

```
method IF_FPM_GUIBB_FORM~GET_DATA.
* call the GET_DATA method of the standard feeder class
    CALL METHOD SUPER->IF_FPM_GUIBB_FORM~GET_DATA
        EXPORTING
            IO_EVENT               = io_event
            IT_SELECTED_FIELDS     = IT_SELECTED_FIELDS
            IV_RAISED_BY_OWN_UI   = IV_RAISED_BY_OWN_UI
            IV_EDIT_MODE          = IV_EDIT_MODE
        IMPORTING
            ET_MESSAGES           = ET_MESSAGES
            EV_DATA_CHANGED       = EV_DATA_CHANGED
            EV_FIELD_USAGE_CHANGED = EV_FIELD_USAGE_CHANGED
            EV_ACTION_USAGE_CHANGED = EV_ACTION_USAGE_CHANGED
        CHANGING
            CS_DATA               = CS_DATA
            CT_FIELD_USAGE        = CT_FIELD_USAGE
            CT_ACTION_USAGE       = CT_ACTION_USAGE.

* Only do the CBA processing in case it's active
    check CL_FPM_FACTORY=>GET_INSTANCE( )->MO_ADAPTATION_MANAGER is bound.

* Check event id to avoid infinite loop
    check io_event->MV_EVENT_ID ne
        IF_FPM_CONSTANTS=>GC_EVENT-ADAPT_CONTEXT_LOCAL.

    data: lo_fpm type ref to if_fpm,
          lo_event type ref to CL_FPM_EVENT.
    field-symbols: <fs_country_code> type SNWD_COUNTRY.
* determine the country code from the data delivered by the standard feeder
    ASSIGN COMPONENT 'COUNTRY_CODE' of STRUCTURE cs_data
        to <FS_COUNTRY_CODE>.

* Check that adaptation context needs to be adjusted.
    check MV_COUNTRY_CODE ne <FS_COUNTRY_CODE>.
    MV_COUNTRY_CODE = <FS_COUNTRY_CODE>.
```

- * create the object to set the adaptation context. The adaptation context is only set locally for the form.

```
CREATE OBJECT LO_EVENT
  EXPORTING
    IV_EVENT_ID      = IF_FPM_CONSTANTS=>GC_EVENT-ADAPT_CONTEXT_LOCAL
    IV_ADAPTS_CONTEXT = abap_true.
```

- * Set the adaptation context via event parameters

```
lo_event->MO_EVENT_DATA->SET_VALUE (
  exporting iv_key   = 'COUNTRY'
  iv_value  = <FS_COUNTRY_CODE> ).
```

- * Add the information, for which UIBB the adaptation context is set

```
lo_event->MO_EVENT_DATA->SET_VALUE (
  exporting iv_key   = if_fpm_constants=>gc_event_param-source_config_id
  iv_value  = MS_INSTANCE_KEY-CONFIG_ID ).
```

```
lo_event->MO_EVENT_DATA->SET_VALUE (
  exporting iv_key   = if_fpm_constants=>gc_event_param-source_config_type
  iv_value  = MS_INSTANCE_KEY-CONFIG_TYPE ).
```

```
lo_event->MO_EVENT_DATA->SET_VALUE (
  exporting iv_key   = if_fpm_constants=>gc_event_param-source_config_var
  iv_value  = MS_INSTANCE_KEY-CONFIG_VAR ).
```

```
lo_event->MO_EVENT_DATA->SET_VALUE (
  exporting iv_key   = if_fpm_constants=>gc_event_param-SOURCE_INSTANCE_ID
  iv_value  = MS_INSTANCE_KEY-INSTANCE_ID ).
```

- * finally raise the event

```
lo_fpm = CL_FPM_FACTORY=>GET_INSTANCE ( ).
lo_fpm->RAISE_EVENT ( lo_event ).
```

endmethod.

If you now run the application, it will still behave as before but now the adaptation context is only applied locally to the Form UIBB.

Hiding of UIBBs

In this section, we want to prove that the adaptation context is really only applied locally. This gives us the opportunity to demonstrate another feature that CBA offers: dynamically hiding UIBBs based on the adaptation context.

1. Let's create an adaptation for the List UIBB underneath the form using the same adaptation context and see that it not applied. Navigate again via the application configuration to the List UIBB's configuration and create an adaptation for `ROLE = * COUNTRY = FR`. In the dialog box where you enter the adaptation context, flag the check box *Only Hide UIBBs*. After closing the dialog box, you are already done. In flagging this checkbox you have stated that this UIBB shall be hidden if the adaptation context fits.
2. Run the application again. It should be unchanged, that is, the List UIBB should still be there. Now let's try it the other way round. Go to your `GET_DATA` method again

and change the code, so that the global adaptation event is thrown (simply delete the `_LOCAL` in the `event id` constant):

```
...
CREATE OBJECT LO_EVENT
  EXPORTING
    IV_EVENT_ID      = IF_FPM_CONSTANTS=>GC_EVENT-
ADAPT_CONTEXT_LOCAL
    IV_ADAPTS_CONTEXT = abap_true.
...
```

3. When you execute the application now, the List UIBB should be removed. (Alternatively, you can set a break-point in the first statement after the `CREATE OBJECT LO_EVENT` statement. Run the application again, select a French supplier and when the break-point is reached, change the value of `LO_EVENT->MV_EVENT_ID` from `FPM_ADAPT_CONTEXT_LOCAL` to `FPM_ADAPT_CONTEXT`. This way, we are changing the event from local to global. Then let the program continue).

Although this is probably a very useful feature, it was only introduced as a work-around for a limitation of CBA.

Limitation:

Dynamic changes of the adaptation context at runtime will only affect adaptations of UIBB configurations. The floorplan configurations are loaded at startup of the application and there is no way to replace it later on. Therefore, it is not possible to change the UIBB assembly of a page at runtime via CBA. The only option is the hiding of UIBBs.

The natural approach to remove the List GUIBB in the French adaptation context would be to create an adaptation for the floorplan component (as in the first part of this tutorial) for the adaptation context `ROLE = *`, `COUNTRY = FR` and remove the List UIBB in this adaptation. But this will not work! The list UIBB will remain on the screen!

Navigation with Launchpads

To navigate to a specific application outside of your FPM application (for example, to another URL or Web Dynpro application, transaction, or report), you use the following FPM toolbar menus:

- *You Can Also*
- *Related Links*

These FPM toolbar menus utilize launchpads.

The launchpad is displayed as a dropdown list. Descriptions that you have written for launchpad applications cannot be displayed in Floorplan Manager.

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.

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.

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 an FPM-based 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 in Administrator Mode* . The application is launched in a separate browser window.
3. In this window, go to the application's identification region and choose the *Customize Page* link.
4. On the *Editor for Web Dynpro ABAP Components* screen, choose *Continue in Change Mode*. The FPM configuration editor, FLUID, opens in edit mode.
5. Choose the *Toolbar Schema* panel and choose *New Toolbar Element*.
6. Choose either the `You Can Also` or `Related Links` element. The element now appears in the table of elements in the application.
7. Turn on the `Attributes` panel.

8. In the panel's table, choose the element you have just added to display its attributes in the `Attributes` panel.
9. In the `Role` field, enter the name of the launchpad role.
10. In the `Instance` field, enter the name of the launchpad instance.
11. To change the name of the button element, enter a different name in the `Name` field.
12. Save the configuration.
13. Test the new configuration.

General Settings of Launchpads

On the *Change Launchpad Role* screen, choose *Extras, General Settings* to display settings which are valid for the whole launchpad and its applications. In the dialog box, you can set the *Check Application Alias is Unique* flag to check that a destination application alias being used by an FPM application is unique. Application aliases are not necessarily unique.

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_LPD_CONTROL`
 - `APB_LPD_OTR_KEYS`
 - `APB_LPD_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.

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: MT_TARGETS

Pa	T	Type	Description

parameter	type	Kind	
entry_type	T y p e	FPM_NAVIGATION_TARGET_TYPE	Entry_type Type FPM_NAVIGATION_TARGET_TYPE Type of application.
parent	T y p e	STRING	GUID of the parent folder or initial.
key	T y p e	STRING	GUID of application.
alias	T y p e	STRING	A (unique) identifier for an application. It is defined in the Customizing of the launchpad.
text	T y p e	TEXT255	Text of the link.
description	T y p e	STRING	Description.
icon_path	T y p e	STRING	Path to an icon.
enable	T y p e	BOOLEAN	Determines if an application is active/enabled or inactive/disabled.
visible	T y p e	BOOLEAN	Determines the visibility of an application
Attributes:			
ID		Description	
MD_OPENING_LAUNCHPAD_F		True indicates that the launchpad could not be opened successfully	

AILED	
Domain: FPM_NAVIGATION_TARGET_TYPE	
ID	Description
APP	Line contains an application.
FOL	Line contains a folder.
SEP	Line contains a separator.

Methods

This navigation interface provides the methods described in the tables below:

NAVIGATE:				
Starts the navigation of an application.				
Parameters	Direction	Type kind	Type	Description
IV_TARGET_KEY	importing	Type	STRING	GUID of Application.
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.				
Parameters	Direction	Type kind	Type	Description
IV_VISIBLE	importing	Type	BOOLE_D	Set an application to visible/invisible.
IV_ENABLE	importing	Type	BOOLE_D	Enable/disable an application.
IV_TEXT	importing	Type	STRING	An alternative text for the application.
IV_DESCRIPTION	importing	Type	STRING	An alternative description for the application.
IV_TARGET_KEY	importing	Type	STRING	GUID of Application.
IV_NOTIFY	importing	Type	BOOLE_D	Invokes notification on all registered nodes / objects.

SET_FILTER:				
Allows you to display the content of another folder.				
Parameters	Direction	Type kind	Type	Description
IT_Filter	importing	Type	T_FILTER	GUIDs of folder which content should be displayed.
MODIFY_PARAMETERS:				
Changes the values of existing parameters or adds a parameter if none exists. (Modifies Application,				

Business and Additional Information parameters).				
Parameters	Direction	Type kind	Type	Description
ID_TARGET_KEY	importing	Type	STRING	GUID of Application.
IT_APPLICATION_PARAMETER	importing	Type	APB_LPD_T_PARAMS	Contains application parameters that will be added or changed.
IT_BUSINESS_PARAMETER	importing	Type	APB_LPD_T_PARAMS	Contains business parameters that will be added or changed.
IT_ADD_INFO_PARAMETER	importing	Type	APB_LPD_T_PARAMS	Additional information which should be modified.
SUPPRESS_REBUILD_OF_MT_TARGETS:				
The rebuilding of table MT_TARGETS can be switched on or off (for mass changes).				
Parameters	Direction	Type kind	Type	Description
IV_SUPPRESS_REBUILD	importing	Type	ABAP_BOOL	Flag to switch the rebuild on or off.

ADD_BEX_ANALYZER:				
Adds an application of type BEx Analyzer to a given launchpad.				
Parameters	Direction	Type kind	Type	Description
IV_PARENT_FOLDER_ID	importing	Type	FPM_APPLICATION_ID	GUID of parent folder. If the parameter is empty, the application will be added at top level.
IS_BEX_ANALYZER_FIELDS	importing	Type	FPM_STRUCTURE_BEX_ANALYZER	Structure that contains the fields to add with BEx Analyzer application type.
EV_APPLICATION_ID	exporting	Type	FPM_APPLICATION_ID	GUID of Application.
ET_MESSAGES	exporting	Type	FPM_T_T100_MESSAGES	Error messages.
EV_ERROR	exporting	Type	BOOLE_D	Status = false - the application was added; Status = true - an error occurred.

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
- ADD_CRYSTAL_REPORT
- ADD_XCELSIUS_DASHBOARD

REMOVE:

Removes an application from a launchpad.

Parameters	Direction	Type kind	Type	Description
ID_APPLICATION_ID	importing	Type	STRING	GUID of Application.

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`
Allows 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 `EV_RETURN = IF_FPM_CONSTANTS~GC_EVENT_RESULT-OK`; if the result of the event processing is not ok, you can return `EV_RETURN = IF_FPM_CONSTANTS~GC_EVENT_RESULT-FAILED`



To prevent a loss of data, you can implement the `NEEDS_CONFIRMATION` method. This method is located in the interface `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:
`eo_confirmation_request = cl_fpm_confirmation_request=>go_data_loss`.

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 `IF_FPM`. This provides the method `GET_NAVIGATE_TO ()` which returns an instance of the Navigation API `IF_FPM_NAVIGATE_TO`.

This interface contains the methods described in the table below and the following list.

Methods of IF_FPM_NAVIGATE TO API

LAUNCH_BEX_ANALYZER:				
Launches an application of type BEx Analyzer.				
Parameters:	Direction	Type kind	Type	Description
<code>IS_BEX_ANALYZER_FIELDS</code>	importing	Type	<code>FPM_S_BEX_ANALYZER</code>	Structure that contains the fields to add with BEx Analyzer application type.
<code>ET_MESSAGES</code>	exporting	Type	<code>FPM_T_T100_MESSAGES</code>	Error messages
<code>EV_ERROR</code>	exporting	Type	<code>BOOLE_D</code>	Status: false - the application was added; true - an error occurred

The following are other methods with a similar interface to `LAUNCH_BEX_ANALYZER`, which allow you to launch a specified application:

- `LAUNCH_URL`
- `LAUNCH_TRANSACTION`

- LAUNCH_REPORT_WRITER
- LAUNCH_OBN
- LAUNCH_INFOSET_QUERY
- LAUNCH_FOLDER
- LAUNCH_BI_ENTERPRISE_REPORT
- LAUNCH_BI_QUERY
- LAUNCH_BI_TEMPLATE
- LAUNCH_KM_DOCUMENT
- LAUNCH_PORTAL_PAGE
- LAUNCH_VISUAL_COMPOSER
- LAUNCH_WEBDYNPRO_ABAP
- LAUNCH_WEBDYNPRO_JAVA
- LAUNCH_CRYSTAL_REPORT
- LAUNCH_XCELSIUS_DASHBOARD

Restarting a WD ABAP Application

The `IF_FPM_NAVIGATE_TO` interface contains a `RESTART` method. This method will restart the currently running WD application.

Parameter	Description
<code>ID_HEADER_TEXT</code>	Text that will be displayed as title in the window.
<code>ID_WD_CONFIGURATION</code>	An optional WD ABAP application configuration
<code>IT_PARAMETERS</code>	An optional set of parameters that will be forwarded to the application
<code>ET_MESSAGES</code>	Messages
<code>EV_ERROR</code>	Is set to true if an error occurred

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.

For further information, see SAP Note 1285228.

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*.
4. On the *Change Launchpad Role* screen, choose *New Application*.
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

Customizing in LPD_CUST is similar as above but has to select the WDA or WDJ in Application category field.

1. Check the *Activate Suspend and Resume Functionality* checkbox after providing all other information.
2. 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_event_resume.
            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:



```
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*):



```

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( ).
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

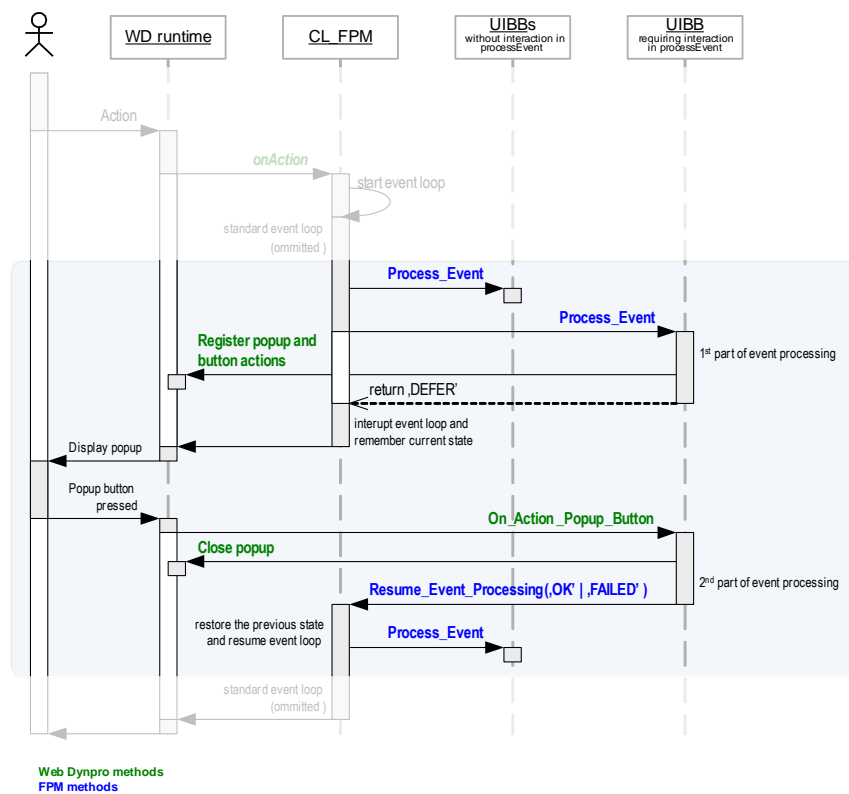
Once the required user input has been obtained, the frozen FPM event is continued (either receiving the result *OK* or *FAILED*). To do this, call the FPM method `RESUME_EVENT_PROCESSING` within the action handler methods for the buttons of the dialog box. The sample code below shows how this might look:



```
DATA lo_fpm TYPE REF TO if_fpm.
lo_fpm = cl_fpm_factory=>get_instance( ).
lo_fpm->resume_event_processing( if_fpm_constants=>gc_event_result-ok ).
```

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:



IF_FPM_WORK_PROTECTION Interface

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

Method Name	Method Description
<code>IS_DIRTY</code>	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).



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:



```
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.



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:
 - If the message lifetime is marked to be bound to a controller, but the controller is NULL or not reachable.
 - If the component for the message is missing.
 - 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.

Parameter	Relevant Method	Description
IO_COMPONENT	All	<p>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</p>
IV_SEVERITY	All	<p>The severity of the message to be reported. There are three possible values, as follows:</p> <ul style="list-style-type: none"> • Error (E) • Warning (W) • Success (I) <p>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</p>

		<p>relating to an error message in an OIF application. The following three values can be passed:</p> <ul style="list-style-type: none"> • GC_SEVERITY_ERROR for Error • GC_SEVERITY_WARNING for Warning • GC_SEVERITY_SUCCESS for Success <p>This is an optional parameter. The default is Error.</p>
IV_LIFETIME	All	<p>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:</p> <ul style="list-style-type: none"> • Lifetime: Determines how long the message exists in the message area; that is, the creation and deletion of the message. The available lifetimes are: <ul style="list-style-type: none"> ○ <i>automatic:</i> FPM handles the destruction of the message as defined by the UI guidelines for the floorplan ○ <i>Manual:</i> the application developer handles the deletion of the message from the message area. • Visibility: Determines when the message appears in the message area. The following values apply: <ul style="list-style-type: none"> ○ <i>Automatic:</i> FPM takes care of the visibility based on the UI guidelines ○ <i>View:</i> the message is visible as long as the view to which the message is bound is available ○ <i>Controller:</i> the message is visible as long as the controller that has raised

		<p>the message is available (see the parameter controller IO_ Controller for details)</p> <ul style="list-style-type: none"> ○ <i>Application</i>: the message is permanently displayed (until it is deleted manually by the application developer) and is visible whilst the application is running. ○ <i>Pop-up</i>: the message is visible only in a dialog box. <p>The default values for both <i>Lifetime</i> and <i>Visibility</i> are <i>Automatic</i>. Not all combinations of lifetime and visibility are possible. Some combinations, for example <i>Lifetime = Manual + Visibility = Pop-up</i> are not available. The permitted combinations are as follows (showing the constant to be used - Lifetime + Visibility):</p> <ul style="list-style-type: none"> ● GC_LIFE_VISIBILITY_AUTOMATIC : <i>Automatic + Automatic</i> (Fully handled by FPM) ● GC_LIFE_VISIBILITY_AUT_DIALOG_BOX: <i>Automatic + Pop-up</i> (Creation and destruction handled by FPM; visible as long as the dialog box is visible) ● GC_LIFE_VISIBILITY_MANU_VIEW : <i>Manual + View</i> (Should be deleted by the application; visible until the view that created it is visible) ● GC_LIFE_VISIBILITY_MANU_CONTROLLER : <i>Manual + Controller</i> (Should be deleted by the application; visible as long as the controller that created it is visible) ● GC_LIFE_VISIBILITY_MANU_APPLICATION : <i>Manual + Application</i> (Should be deleted by the application; visible as long as the application is running)
IV_PARAMETERS	All	A group of parameters of the type Web

		Dynpro <code>R_NAME_VALUE_LIST</code> 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
<code>IR_MESSAGE_USER_DATA</code>	All	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.
<code>IV_MESSAGE_INDEX</code>	All	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: <ul style="list-style-type: none"> • Error severity • Message index (parameter <code>MSG_INDEX</code>) • Context element (if it exists) • Context attribute (if it exists)
<code>IO_ELEMENT</code>	All	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.
<code>IV_ATTRIBUTE_NAME</code>	All	The element attribute to which the message must be mapped. This parameter is used in conjunction with the <code>IO_ELEMENT</code> . If the message is to be mapped to more than one attribute, use parameter <code>IT_ATTRIBUTES</code> instead.
<code>IV_IS_VALIDATION_INDEPENDENT</code>	All	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 <code>ABAP_FALSE</code> (default value), the standard action is no longer executed after this message is created. However, if the parameter's value is <code>ABAP_TRUE</code> , the standard action is executed.
<code>IO_CONTROLLER</code>	All	Pass the reference to the controller whose lifetime will dictate the lifetime of the messages which have the lifetime set to the context.
<code>IS_NAVIGATION_ALLOWED</code>	All	Use this flag if you need to allow navigation, even on an 'E' message in the GAF. Relevant for GAF applications only.

IV_VIEW	All	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.
IV_MESSAGE_TEXT	REPORT_MESSAGE	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.
IS_BAPIRET2	REPORT_BAPIRET2_MESSAGE	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 BAPIRET2 structure. Note that if the BAPIRET2 structure contains a severity value of A, the message is converted into an exception.
IR_MESSAGE_OBJECT	REPORT_OBJECT_MESSAGE	<i>Exception Object</i> providing access to message long text.
IT_ATTRIBUTES	All	<i>Table of attributes to which the message is mapped.</i> 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. <i>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.</i> 2. Optional Parameter of type String table. <i>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.</i>
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:

Method	Mandatory Parameters
REPORT_MESSAGE	Message text
REPORT_T100_MESSAGE	Message class and message number
REPORT_BAPIRET2_MESSAGE	BAPIRET2 structure

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.

Parameter	Relevant Method	Description
IV_TEXT	RAISE_EXCEPTION	Optional text that can be passed while raising a simple exception. This text is logged and can later be used for analysis
IV_MSGID	RAISE_T100_EXCEPTION	Message class ID for the T100 message. Use this parameter to raise an exception whose text is based on the T100 message mechanism.
IV_MSGNO	RAISE_T100_EXCEPTION	Message number of the T100 message class.

IV_PARAMETER_1 IV_PARAMETER_2 IV_PARAMETER_3 IV_PARAMETER_4	RAISE_T100_EXCEPTION	Optional message parameters.
IO_EXCEPTION	RAISE_CX_ROOT_EXCEPTION	The exception class inheriting from CX_ROOT. This parameter is a mandatory parameter.
IV_ADDITIONAL_TEXT	RAISE_CX_ROOT_EXCEPTION	Additional text to be added while reporting an exception based on CX_ROOT.
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.

Parameter	Relevant Method	Description
IO_COMPONENT	CLEAR_MESSAGES	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.
IV_EXCLUDE_ERROR	CLEAR_MESSAGES	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.
IV_EXCLUDE_WARNING	CLEAR_MESSAGES	Default is false. Override it with true, if warnings raised for

		the component are to be saved.
IV_EXCLUDE_SUCCESS	CLEAR_MESSAGES	Default is false. Override it with true, if success messages raised for the component are to be saved.

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.

Lifetime/Visibility	Message Behavior
Automatic	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. This behavior is similar to the automatic messages in the parent window.
View	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.
Controller	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.
Application	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.
Pop-up	Pop-up: The message is visible only in a dialog box as long as the Non-FPM dialog box is open.

Message Manager – ON_NAVIGATE Event

Applications can use the ON_NAVIGATE event to perform any action on click of the message link. This is applicable *only for messages which are displayed as a link in the message area*.

While reporting a message, the IS_ENABLE_MESSAGE_NAVIGATION parameter must be set to X. Only then will the message be displayed as a link. When this link is clicked, FPM raises an ON_NAVIGATE event; the application performs its business logic by handling this event.

The parameters that are passed as part of event data (MO_EVENT_DATA) are as follows:

- ID (Type: String)
- CONTEXT_ELEMENT (Type: IF_WD_CONTEXT_ELEMENT)
- MESSAGE_ID (Type: String)

If the application needs to pass additional information which might be used in the action handler of the ON_NAVIGATE event, use parameter IR_MESSAGE_USER_DATA while reporting a message. In the ON_NAVIGATE event, to receive the information passed, get the ID from the mo_event_data method:

```

CALL METHOD MO_EVENT_DATA->GET_VALUE
EXPORTING
  IV_KEY   = 'MESSAGE_ID'
IMPORTING
  EV_VALUE = lv_msg_id.

```

This ID can be used when calling the `IF_WD_MESSAGE_MANAGER~GET_MESSAGE_FOR_ID` method, which returns all information about the message.

```

DATA: l_message      TYPE if_wd_message_manager=>ty_s_message.
l_message = wd_this->message_manager->get_message_for_id( lv_msg_id ).

```

`l_message-msg_user_data` gives the user data set while reporting the message.

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.

Message mapping can be done for messages which are reported for GUIBBs also.

Enabling Message Mapper

You enable message mapping for an application in the component configuration editor of the application. Choose *Display -> Message Mapper Settings*. The *Message Mapper Settings* dialog box appears. Select *Enable Message Mapping*.

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:

- 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.
- 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.

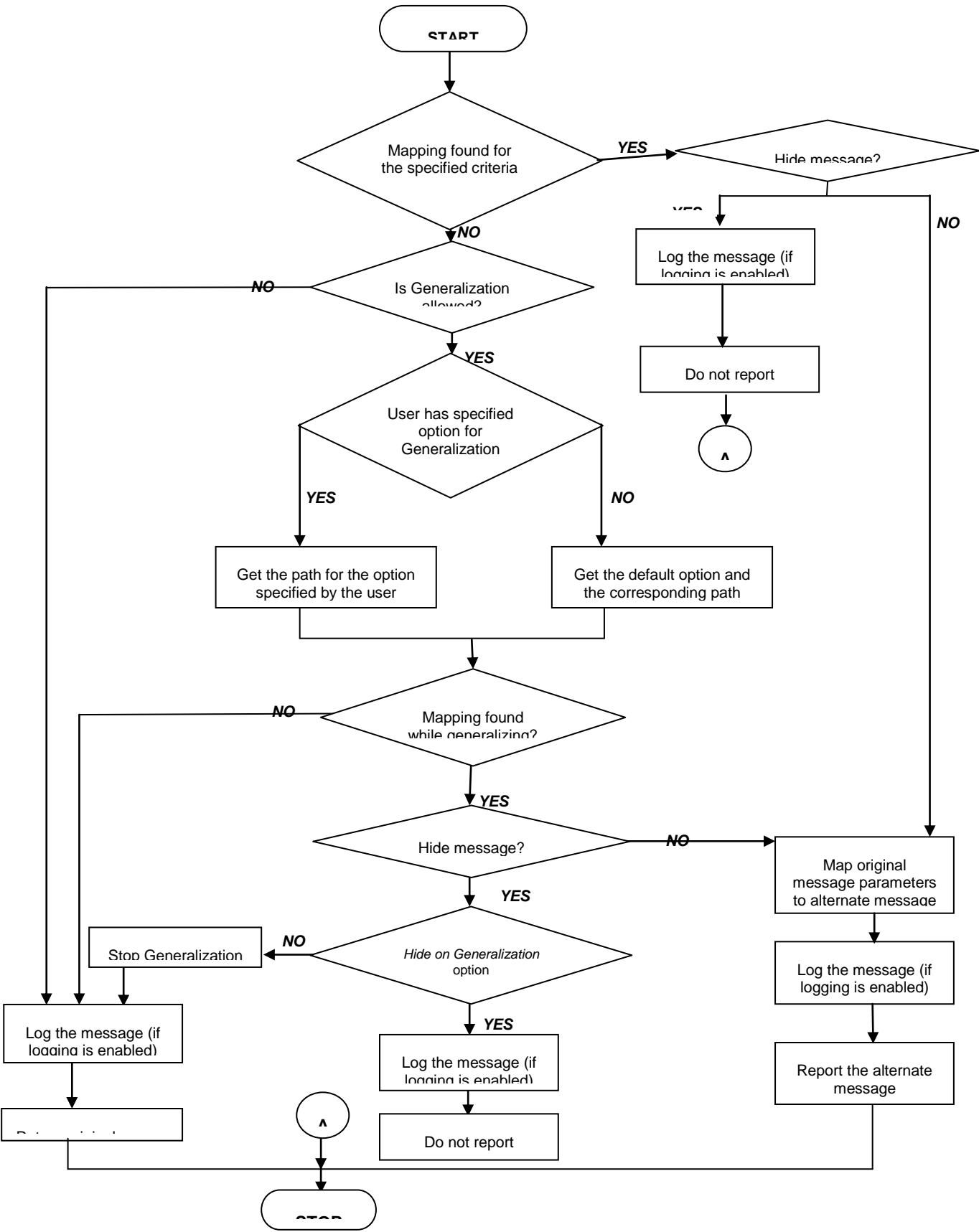
By default, the *Generalization Type* is Type 1.

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:

Generalization Level	Message Fields				
G8	Context				
G7	Context				Message Type
G6	Context			Category	
G5	Context			Category	Message Type
G4	Context	Message Id			Message Type
G3	Context	Message Id	Message No		
G2	Context	Message Id	Message No		Message Type
G1	Context	Message Id	Message No	Category	
G0	Context	Message Id	Message No	Category	Message Type

MESSAGE MAPPER FLOW



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:

Field Name	Field Entry
<i>Object</i>	FPM_MSG_MAPPER
<i>Sub-Object</i>	RUNTIME

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

G8	Context			
G7	Context			Message Type
G6	Context		Category	
G5	Context		Category	Message Type
G4	Context	Message Id		Message Type

G3	Context	Message Id/Message No		
G2	Context	Message Id/Message No		Message Type
G1	Context	Message Id/Message No	Category	
G0	Context	Message Id/Message No	Category	Message Type

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:

Parameter Name	Parameter Type	Opt	Data Type
IV_CONTEXT	Importing	Y	FPM_S_MSG_MAPPER-MSG_CONTEXT
IV_CATEGORY	Importing	Y	FPM_S_MSG_MAPPER-MSG_CATEGORY
IV_GENERALIZATION	Importing	Y	FPM MSG_GENERALIZE

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_CATEGORYT`
This is a text table for `FPM_MSGCATEGORY`.

The fields for each table are described in detail below.

Table FPM_T_MSG_MAPPER

Field	Field Type	Description	Field Entry
<u>NAMESPACE</u>	FPM_NAMESPACE	Namespace for Message Mapper.	Mandatory
<u>MSG_CONTEXT</u>	FPM_CONTEXT	FPM application context name. Indicates, for example, the business unit or role in which the message is mapped.	Mandatory
<u>MSGID</u>	SYMSGID	Message class of system message.	Optional
<u>MSGNO</u>	SYMSGNO	Message number of system message.	Optional
<u>MSGTY</u>	SYMSGTY	Message type of system message.	Optional
<u>MSG_CATEGORY</u>	FPM_MSG_CATEGORY	Message category of system message.	Optional
<u>ALT_MSGID</u>	SYMSGID	Message class for alternate message.	Optional
<u>ALT_MSGNO</u>	SYMSGNO	Message number for alternate message.	Optional
<u>ALT_MSG</u>	FPM_ALT_MSG	Alternate message. Type your message text here.	Optional
<u>ALT_MSGTY</u>	SYMSGTY	Message type for alternate message.	Optional
<u>HIDE_MSG</u>	BOOLEAN	Hides the message.	Optional
<u>FINAL</u>	BOOLEAN	Ensures that the message mapping cannot be edited.	Optional

Table FPM_MSGCATEGORY

Field	Field Type	Description	Field Entry
<u>MSG_CATEGORY</u>	FPM_MSG_CATEGORY	Message category.	Mandatory
<u>NAMESPACE</u>	FPM_NAMESPACE	Namespace for Message Mapper.	Mandatory

Table FPM_CATEGORYT

Field	Field Type	Description	Field Entry
MSG_CATEGORY	FPM_MSG_CATEGORY	Message category.	Mandatory
NAMESPACE	FPM_NAMESPACE	Namespace for Message Mapper.	Mandatory
LANGU	SPRAS	Language key.	Optional
MSG_CATEGORY_NAM	FPM_MSG_CATEGORY_NAM E	Description of the message category.	Optional

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.

[Help](#)

Error

Enter a flight code

Error Details

Technical Exception Test.Null Object Exception

Error Description Enter a flight code. Diagnosis Application cannot proceed without flight code. System Response Procedure Enter a valid flight code. Procedure for System Administration

Additional Information In this screen you can perform the following actions:

- Add flight details
- You can add flight details (for example, date or flight time).
- Skip main steps and navigate to step Review by clicking the Finish button.
- Navigate to substeps by clicking the Flight Details button.
- Navigate to the airline's Web site in a new browser window by clicking on the Airline URL link within the identification region.
- Navigate to the airline's Web site in a new browser window by clicking on the Go to <Airline Name> link within the You can also drop down list.
- Navigate to the airline's Web site in a new browser window by clicking on the Search Airline link within the You can also drop down list. This starts a Google search for the specified airline.

The following FPM features implemented in this screen:

- Dynamically adding GAF standard toolbar button
- Dynamically adding You can also link to the toolbar
- Dynamically adding links to You can also
- Clickable IDR links
- Navigating to subroadmap step
- Skipping the main steps

Right Click to Show/Hide Quick Help

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.

Method name	Parameters	Description
DISPLAY_ERROR_PAGE	IO_ERROR_DETAILS-type IF_FPM_ERROR_DETAILS	This method is called from application passing the error data, to display the error page.

- 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

Method name	Parameters	Description
CREATE_FROM_BAPIRET2	IS_BAPIRET2 IV_TECHNICAL_EXCEPTION IV_ADDITIONAL_INFO IV_ERROR_ID RO_ERROR_DETAILS	It is used to created error data if the application has error information in form of bapiret2.
CREATE_FROM_T100	IS_T100 IV_TECHNICAL_EXCEPTION IV_ADDITIONAL_INFO IV_ERROR_ID RO_ERROR_DETAILS	It is used to created error data if the application has error information in form of T100 table.
CREATE_FROM_OBJECT	IO_EXCEPTION_OBJ IV_TECHNICAL_EXCEPTION IV_ADDITIONAL_INFO IV_ERROR_ID RO_ERROR_DETAILS	It is used to created error data if the application has error information in form of exception object.
CREATE_FROM_OTR	IV_OTR IV_TECHNICAL_EXCEPTION IV_ADDITIONAL_INFO IV_ERROR_ID RO_ERROR_DETAILS	It is used to created error data if the application has error information in form of OTR.
CREATE_FROM_EMPTY_DATA	IV_ERROR_ID RO_ERROR_DETAILS	To get error object when no error data is available.

Example:

```
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 symmsg.
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

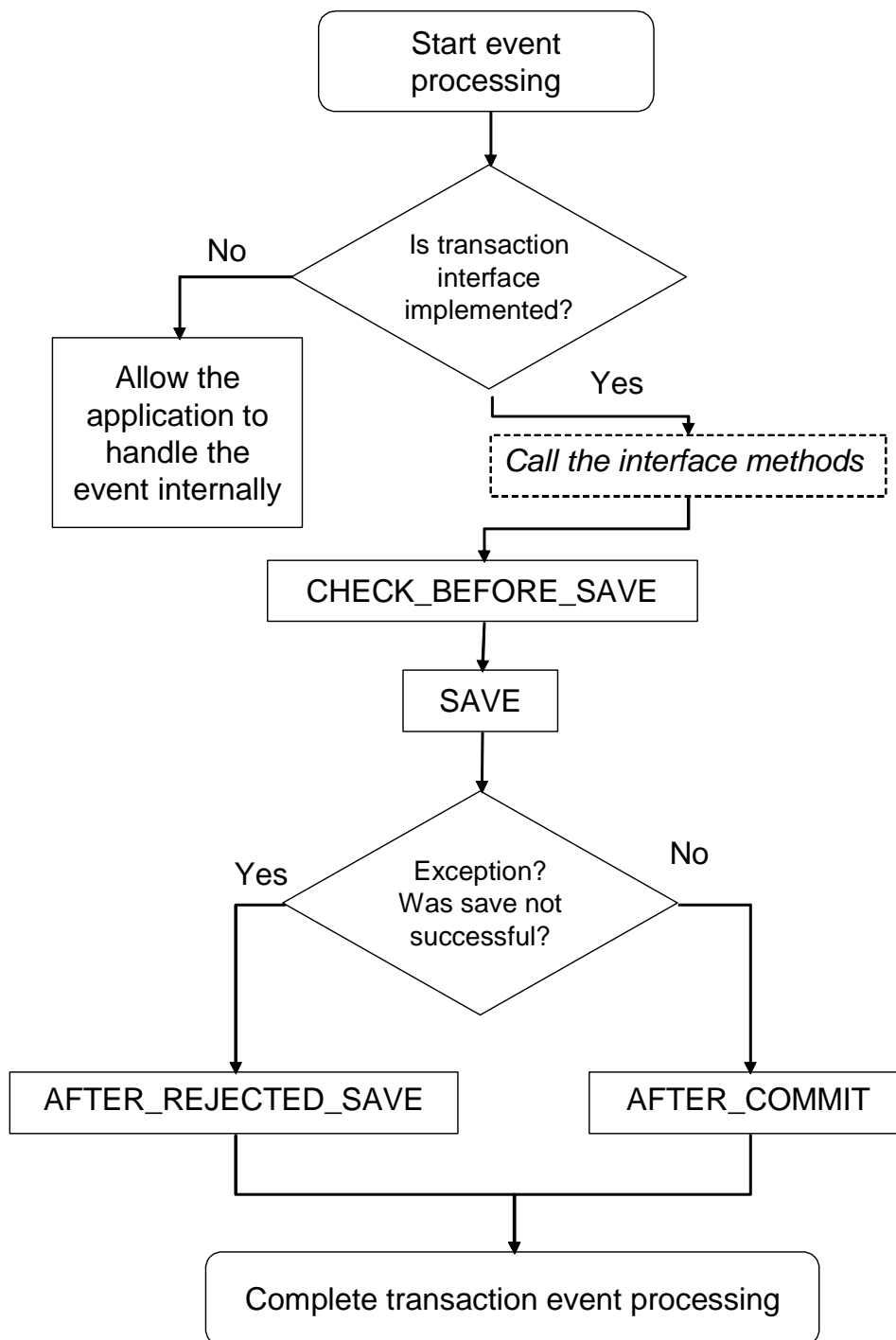
This Web Dynpro interface provides you with methods to handle transactions in a systematic manner by FPM. This is an optional interface; an application can handle the transactions independently, without implementing the interface.

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.

Method Name	Method Description
CHECK_BEFORE_SAVE	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.
SAVE	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.
AFTER_COMMIT	You can perform cleanup activities such as releasing database locks, releasing other resources, triggering an event for processing after a successful commit.
AFTER_REJECTED_SAVE	Here you can perform your roll back activities. You can also release locks and resources.

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

- Pure Master-Detail (the application is built only on the Master Detail)
- 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 `IF_FPM_RESOURCE_MANAGER` interface has been implemented by the component. If the interface is implemented, it calls the `on_Hide` method else releases the component.
- In the `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.

Requirement	Transient Flag	Implement <code>IF_FPM_RESOURCE_MANAGER</code>	Veto	Coding with the resource manager to handle application data
I do not want transient behavior for any UIBB and I do not want to release any memory.	False	NA	NA	
I do not want transient behavior for any UIBB but I would like to release some resources.	False	Yes	NA	Yes Based on need from business logic.
I want all my UIBBs to be transient. I do not have the need to release any resources explicitly.	True	NA	NA	NA
I want all my UIBBs to be transient. I want to release some resources explicitly.	True	Yes	No	Yes Based on need from business logic.
I want only some of the UIBBs to be released. I would like to retain some due to business logic reasons.	True	Yes	Yes (only for those that do not need to be released).	Based on need from business logic.
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.	True	Only on those that need to veto (or not be released).	Yes (only for those that do not need to be released).	NA

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:



```
method ON_HIDE.
  data: lv_veto type boole_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.

IV_RELEASE_COMPONENT	EV_VETO_RELEASE	FPM action
False	True/False	Ignore the veto value; do not release component.

True	False	Release the component.
True	True	Do not release the component.

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:



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.

Method	Method Description
<code>GET_CURRENT_STATE</code>	Returns the current navigation state within the application.
<code>GET_VARIANTS</code>	Returns a list of all available variants.
<code>GET_UIBB_KEYS</code>	Returns a list of all UIBB assigned to a specified main step, substep, main view or subview.
<code>GET_UIBB_KEYS_FOR_CONF_STEP</code>	Returns a list of all UIBB assigned to a

	confirmation screen.
GET_UIBB_KEYS_FOR_INIT_SCREEN	Returns a list of all UIBB assigned to an initial screen.

The following table gives you an overview of all methods available for an OIF application.

Method	Method Description
GET_MAINVIEWS	Returns a list of all main views for a given variant.
GET_SUBVIEWS	Returns a list of all subviews for a given main view.

The following table gives you an overview of all methods available for a GAF application.

Method	Method Description
GET_MAINSTEPS	Returns a list of all main steps for a given variant.
GET_SUBSTEP_VARIANTS	Returns a list of all substep variants for a given main step.
GET_SUBSTEPS	Returns a list of all substeps for a given substep variant.
GET_HIDDEN_MAINSTEPS	Returns a list of all hidden main steps for a given variant.

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	Method Description
ADD_UIBB	Adds dynamically another UIBB to a main view, subview, main step, or substep.
REMOVE_UIBB	Removes dynamically another UIBB to a main view, subview, main step, or substep.

The following table gives you an overview of all methods available for an OIF application.

Method	Method Description
ADD_MAINVIEW	Adds dynamically another main view at runtime.
REMOVE_MAINVIEW	Deletes dynamically a given main view at runtime.
ADD_SUBVIEW	Adds dynamically another subview at runtime.
REMOVE_SUBVIEW	Deletes dynamically a given subview at runtime.
RENAME_MAINVIEW	Renames dynamically a given main view at runtime.
RENAME_SUBVIEW	Renames dynamically a given subview at runtime.
SET_SELECTED_SUBVIEW	Changes the target subview within a given main view.

	This method must only be used in order to enforce a given main view to switch to the provided subview instead of the default subview.
--	---

The following table gives you an overview of all methods available for a GAF application.

Method	Method Description
RENAME_MAINSTEP	Renames dynamically a given main step at runtime.
RENAME_SUBSTEP	Renames dynamically a given substep at runtime.
ENABLE_MAINSTEP	Enables or disables a given main step at runtime.
HIDE_MAINSTEP	Hides a given main step within the roadmap. The affected main step will not be visible as a main step in the roadmap anymore. Nevertheless, the hidden main steps continue to be processed in the background in order to keep the business logic untouched.

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)



```

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.

FPM CHIP Integration

In principle, a floorplan component can be used as a CHIP: The wrapper component `FPM_CHIP` implements the CHIP interface, and the OVP component directly implements it. However, this is not recommended since a full floorplan UI is usually too large to be displayed in a CHIP. However, there is a need for displaying single GUIBBs in a CHIP, for example, a Form UIBB. For this purpose FPM offers a single UIBB floorplan, the UIBB CHIP Wrapper (UCW).

Structure of the UCW

The UCW only displays a single UIBB without an IDR or page header. However it is permitted to assign nested (composite) UIBBs to the UCW (for example, Composite or Tabbed UIBBs or Search UIBBs (which can embed a result List UIBB)). The configuration is rather simple and must be maintained with the WD ABAP default configuration editor.

Multi-Instantiability

As there may be several FPM-based CHIPS on a page, it must be possible to run multiple FPM instances in parallel. In this case, the FPM factory to access the central FPM instance (reference to `IF_FPM`) can no longer be used.

To achieve multi-instantiability, FPM offers the interface `IF_FPM_MULTI_INSTANTIABLE`, both on WD ABAP and ABAP OO level. The interfaces have a method `FPM_INITIALIZE` which passes the local FPM instance. With the other method, `FPM_IS_MULTI_INSTANTIABLE`, an FPM object can declare itself as being multi-instantiable with a Boolean return value. An FPM object that returns a `TRUE` value commits itself to obey the following rules:

1. Only the local FPM instance (`IF_FPM`) passed in `FPM_INITIALIZE` must be accessed. `CL_FPM_FACTORY=>GET_INSTANCE` must not be accessed except for a simple determination between runtime and design time.
2. The methods of `CL_FPM_SERVICE_MANAGER` are always called with the local FPM instance.
3. Method `CL_FPM_UIBB_API_FACTORY` is always called with the local FPM instance.
4. Instances of `CL_FPM_LPD_PROVIDER` are always created with the local FPM instance.
5. The application logic itself must compatibly run with parallel FPM instances. (Example: If FPM application objects use the same singleton pattern to access shared transactional data, they should not declare themselves as being multi-instantiable.)

An entire FPM instance is multi-instantiable as long as all its instantiated FPM application-specific objects (that is, application controllers, freestyle UIBBs, GUIBB feeder classes, wire model connector classes and

wire model transaction handlers) declare themselves as being multi-instantiable too. GUIBBs are multi-instantiable if, and only if, their feeder classes are multi-instantiable too. If there is more than one non multi-instantiable FPM instance in one roll area, the FPM runtime raises a short dump.

To avoid conflicts due to multi-instantiation, it is recommended that new developments ensure multi-instantiation of FPM objects by implementing `IF_FPM_MULTI_INSTANTIABLE`, accessing solely the FPM instance passed in method `FPM_INITIALIZE`, and returning a `TRUE` value in `FPM_IS_MULTI_INSTANTIABLE`. In particular, it is strongly recommended that UIBBs and feeder classes developed for usage in the UCW be multi-instantiable.

Communication between FPM CHIPS

UIBBs and GUIBB feeder classes can optionally implement the WD interfaces `IF_FPM_CHIP` and `IF_FPM_CHIP_FEEDER` respectively. Using the Web Dynpro CHIP API passed in method `FPM_CHIP_DO_INIT` and `CHIP_INIT`, it is possible to add dynamic inports. Inport events are communicated in the UCW via the FPM event `FPM_HANDLE_CHIP_INPORT`, and the CHIP port event object is attached as the event parameter with the key `FPM_CHIP_PORT_EVENT`.

Creating a CHIP for a Single UIBB

FPM applications are usually too large to be completely displayed inside a CHIP. However, you can use a special FPM CHIP wrapper (known as the UCW - UIBB Chip Wrapper) - basically, a single UIBB floorplan - which allows you to display just a single UIBB in a side panel or other *Page Builder* page.

Prerequisite:

You have created a UIBB (either a freestyle WD component or one based on a GUIBB component).

Procedure:

1. Create the UCW configuration (and add your existing UIBB to the UCW floorplan)

1. Create a configuration for WD component `FPM_UCW_COMPONENT` (found in `APB_FPM_CORE` package)
 - Choose the `FPM_UCW_COMPONENT` component and from the context-menu, choose Create/Change Configuration.
 - Enter a name for your new configuration in the Configuration ID field and choose New
 - Enter a description and package details in the dialog boxes that follow.
2. In the *Component-Defined* panel, select the *ucwApplication* row, and add a new UIBB. A list of UIBB attributes appears.
3. Enter the following attributes for the UIBB you have created (see Prerequisites above):
 - Component: this is the original component that your existing configuration is based on (e.g. `FPM_LIST_UIBB`)
 - Window Name
 - Configuration Name
4. Save your configuration.

2. Create the CHIP (and refer it to the UCW configuration)

1. In APB_FPM_CORE package, choose the WD component FPM_CHIP.
2. In the context menu, choose Create, Web Dynpro CHIP.
3. Enter a name and package details in the dialog boxes that follow.
4. On the *Properties* tab, enter a display name and description and the following information:
 - Component - FPM_UCW_COMPONENT (overwrite the entry FPM_CHIP)
 - Configuration Name - the name of the configuration you created in step 1
 - Interface View - FPM_WINDOW
 - Plug Name - DEFAULT
5. Save.

You have now created a CHIP containing a single UIBB. You can add it to a side panel or any other *Page Builder* type page.

Appendix I: Authorization Profiles

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.

Appendix II: Building FPM Applications on BOL with NW703/WEBCUIF702

Using the feeder class concept it is possible to build generic adapters on generic business logic encapsulations. Examples are BOL, BOPF or the ESF. Such adapters allow some reuse of generic feeder classes by application development so that UI logic does not need to be re-implemented for each application.

FPM provides a rich adapter implementation on BOL which allows creating application UIs without a line of code, purely consisting of Web Dynpro ABAP configurations.

The adapter objects can be found in package `APB_FPM_BOL_CORE` and test applications in `APB_FPM_BOL_TEST` in the `WEBCUIF` software layer.

GUIBB Configuration with Generic BOL Feeder Class

Search GUIBB

You can create a Search GUIBB configuration by assigning the feeder class `CL_GUIBB_BOL_DQUERY`. In the parameters dialog box (below), enter the BOL component and a dynamic query.

The screenshot shows a dialog box titled "Edit Parameters" with a blue header bar. Below the header, there are two sections: "Feeder Class" and "Parameters".

Feeder Class: A text field contains "CL_GUIBB_BOL_DQUERY" and a button labeled "Generic BOL Feeder Search".

Parameters:

- * Component Name: Text field with "EPM" and a copy icon.
- * Dyn Query Name: Text field with "AdvQueryOrderByHeader" and a copy icon.
- Execution Mode: Dropdown menu with "On GUIBB specific button event" selected.
- Selection Mode: Dropdown menu with "Single" selected.

At the bottom right, there are "OK" and "Cancel" buttons.

After confirmation of the dialog box, a default configuration is provided which can be adjusted according to your needs.

It is recommended not to configure result list attributes in the search configuration. Instead, assign a result list GUIBB configuration which is configured on the result object of the dynamic query.

Form GUIBB

You can create a Form GUIBB configuration by assigning the feeder class `CL_GUIBB_BOL_FORM`. In the parameters dialog box enter the BOL component and object (see following screenshot):

Edit Parameters

Enter valid feeder parameters

Feeder Class

* Feeder Class: Generic BOL Feeder Form

Parameters

* Component Name:

* Object Name:

Editable:

Join Structure

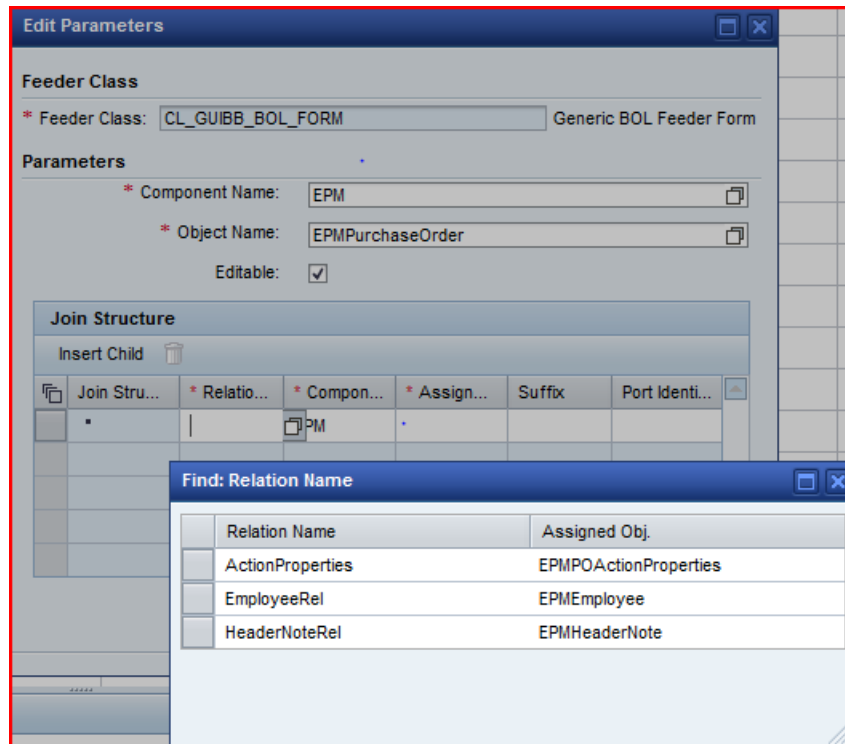
Insert Child

	Join Struc...	* Relation...	* Compon...	* Assigne...	Suffix	Port Identi...

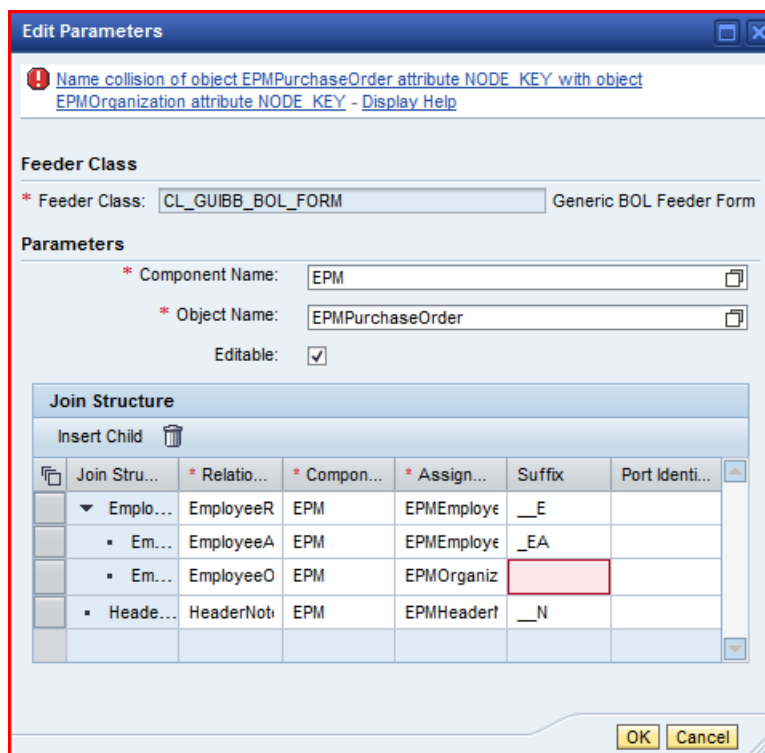
OK Cancel

Set the flag “Editable” if you would like to have editable fields in your list. If you do not set the flag, the objects will not be locked, and all fields will be statically read-only.

You can also add fields of related objects (by relation with non-multiple target cardinality) by specifying the related objects in the “Join Structure” table. Press the “Insert Child” button to create a record, and use the value help to select a related object (see following screenshot).



Upon validation, it might be necessary to define a suffix for the joined objects attributes to avoid name collisions (see following screenshot).



You can also create nested joins by selecting a row and choosing the "Insert Child" button. If no row is selected, joins on the top levels (that is, directly related to the main BOL object) are created.

You may assign a port identifier if you want to have separate outports for joined objects to access it directly via FPM wiring.

After confirmation of the dialog box, a default configuration is provided which can be adjusted according to your needs.

List GUIBB

You can create a List GUIBB configuration by assigning feeder class `CL_GUIBB_BOL_LIST`.

Set the flag “Editable” if you would like to have editable fields in your list. If you do not set the flag, the objects will not be locked, and all fields will be statically read-only.

You can also join related BOL objects. However, for the list component, you should avoid configuring unnecessarily excessive joins if large amounts of data are expected at runtime, as the join relation tree must be processed for each row.

If the flag “Init Lead Sel” (Initial Lead Selection) is checked, the feeder always provides at least one row as selected at runtime.

With the parameter “Fast Entry Mode” you can control whether there should be blank rows created if the list UIBB is switched to edit mode in the OVP floorplan at runtime (see screenshot).

Edit Parameters

Feeder Class

* Feeder Class: Generic BOL Feeder List

Parameters

* Component Name:

* Object Name:

Editable:

Join Structure

Insert Child

Join Stru...	* Relatio...	* Compo...	* Assign...	Suffix	Port Ident...
▼ Empl...	EmployeeR	EPM	EPMEmploye	__E	
▪ Em...	EmployeeA	EPM	EPMEmploye	__EA	
▪ Head...	HeaderNot	EPM	EPMHeaderI	__N	

Init Lead Sel:

Fast Entry Mode:

OK Cancel

The default is “Not Active”, and you can activate the mode such that either single lines are inserted or blocks of the visible row size of the table. These rows reflect non-sendable BOL entities which are only transferred to sendable entities once an attribute is changed in the UI.

The assumption is that the feeder class only receives the tree's top level records as entities from the wiring at runtime. For all these top level entities the feeder recursively builds up the tree, retrieving the tree's lower level records as the child entities provided by the cyclic BOL relation.

The field which carries the text for the master column can be specified by the feeder parameter "Master Column Text Reference". The initial tree expansion level can be specified with the field "Initial Expansion" with the following values:

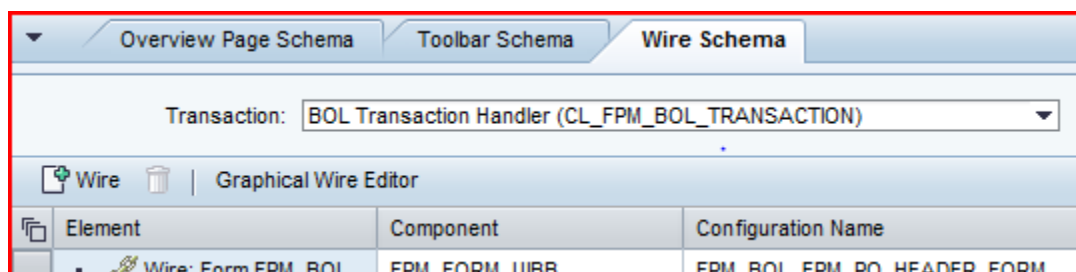
- 0
Not even the top level entities are initially expanded
- 1
Only the top level entities are initially expanded
- 2
Only the top level entities and its direct children are initially expanded, and so on for larger integers.
- -1
The tree is initially expanded until no child entities are found. This "infinite" expansion must, however, be chosen with care since a cycle (for example, if an entity is related to its own child) then leads to infinite loops at runtime.

Floorplan Configuration

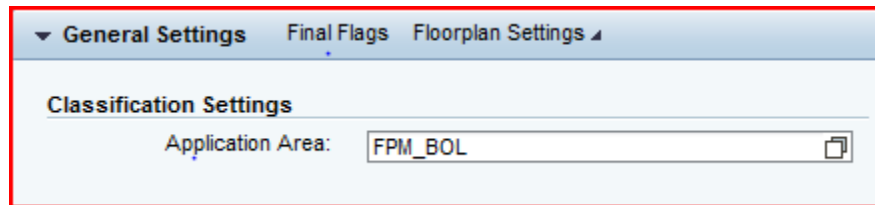
For the generic FPM BOL adapter it is necessary to define settings on the floorplan level, in particular the transaction handling and the dependencies between UI components ("Wiring") must be defined.

BOL-Specific Settings

In the floorplan configuration it is necessary to assign the FPM BOL transaction handler class, `CL_FPM_BOL_TRANSACTION`, as the wire-model transaction handler. This is done in the wire schema and provides transaction integration of your FPM BOL application, taking care of transaction action handling such as "Save" and "Cancel" global message handling (see screenshot).



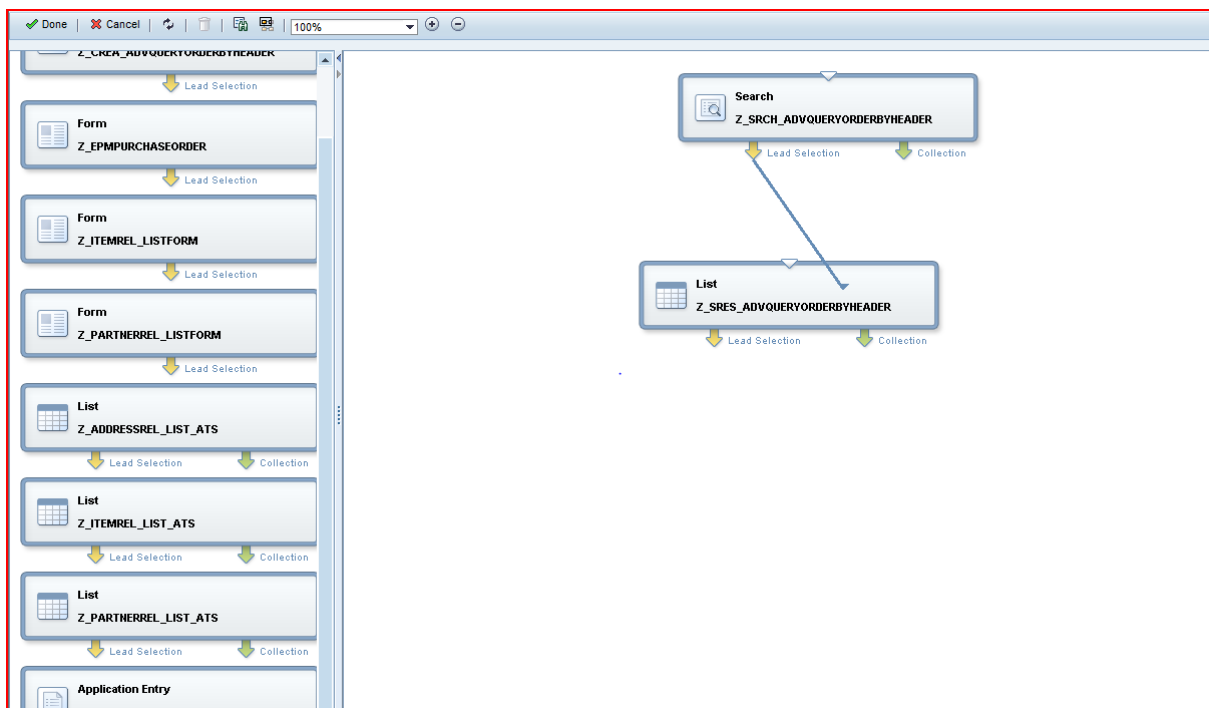
Though functionally not necessary, it is recommended to maintain the application area in the *General Settings* with 'FPM_BOL'.



With this classification, FLUID will propose the standard BOL feeder classes whenever you create a UIBB configuration by drilling down from a floorplan, tabbed or composite configuration.

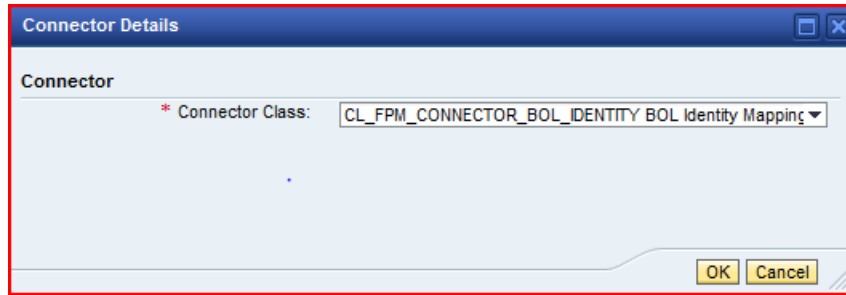
Wiring

You can maintain the wires in the Wire Schema table. However, it is easier to do it graphically, using the *Graphical Wire Editor* (button on the Wire Schema toolbar). Drag the UIBBs you want to connect with wires from the repository into the work area. There, you can draw a wire connection by dragging from an output of the source UIBB and dropping onto the target UIBB (see screenshot).

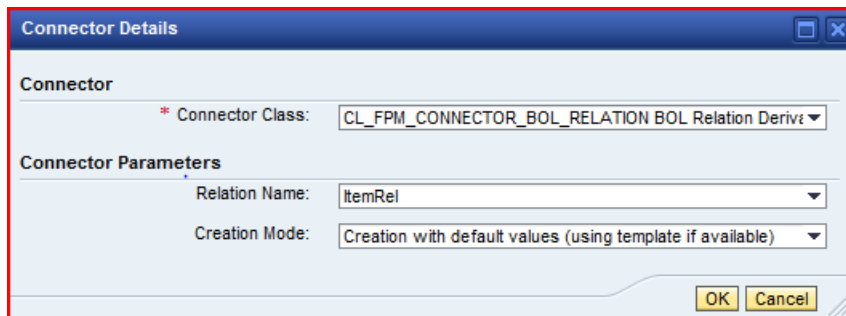


A dialog box appears which prompts you for the connector class. Choose one of the following:

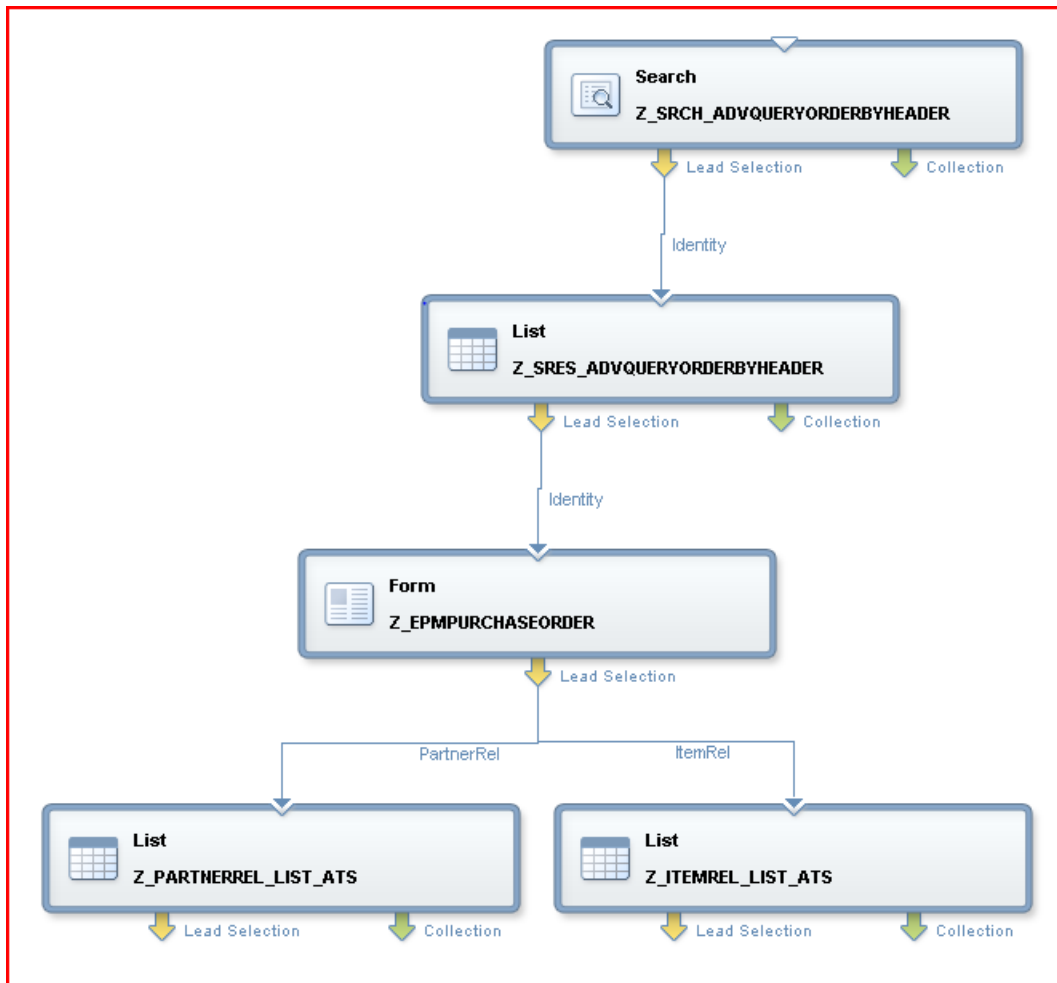
- Choose `CL_FPM_CONNECTOR_BOL_IDENTITY BOL` if no logic has to be processed between the UIBBs (for example, between a search and a result list or between a master list and its detail form).



- Choose `CL_FPM_CONNECTOR_BOL_RELATION BOL` if a BOL relation should be processed between the UIBBs. In this case, you can select from the dropdown list the BOL relations that exist between the BOL objects assigned to the source UIBB's output and the target UIBB. If only one relation exists, it is preselected.



Leave the default value for the creation mode and confirm the dialog box. This way, you can build up the data flow of your application.



OVP Application with Ex-place Navigation

According to the UX guidelines 2.0, navigation from a search page to the main application should be ex-place, that is, a new window shall be opened. Technically, this is realized in a single OVP configuration; the search page is an initial screen with a search UIBB and a result list UIBB. The result list contains columns with links.

There is always a primary link column which restarts the OVP for the main search object in a mode such that the search page is executed with parameters in the background. There may also be secondary links which start OVP applications.

The use-case is that the search result is a BOL root object. The primary link needs to be configured as a link to action and the event `FPM_NAVIGATE` must be assigned. To this event, the feeder attaches an event parameter structure of type `FPM_S_EXTERNAL_NAVIGATION_INFO` or, in the case that the root object key structure contains many attributes, a larger structure including `FPM_S_EXTERNAL_NAVIGATION_INFO`. The attributes "Role", "Instance ID" and "Application Alias" specify a launchpad entry for the application. This is not mandatory for the primary link since the FPM is able to dynamically create a launchpad navigation at runtime for restarting the current application. By default, the value of the selected link field is extracted into a URL parameter, and in the restarted session the value is filled to a search attribute with the same name. For instance, if you have a search page for sales orders and you click on a sales order link "00004711" with technical field name `SO_ID`, then the application is started in a new window where the search is maintained with `SO_ID = '0004711'`. If the technical field name of the result list column does not match the technical field

name of the search attribute, then you need to maintain a launchpad customizing with a URL parameter mapping and reference this launchpad entry in the event parameters for the primary link too.

With the attribute “Source Attribute” a column can be specified which carries the identifying attribute if it is not the link column. Choose “Execute Search and Leave” for the Processing Mode so that the search page will be executed in dark mode in the new window. Choose the Edit Mode state in which the application shall be started (usually this is read-only mode).

With “Page ID” you can specify a specific page as a start page for the application. This is only necessary if the search page initial screen is not the only initial screen and not flagged as a default page.

If you want to support a create-scenario for your application based on a BOL root object, you need to add another initial screen to your application. It should essentially be a copy of the search page, that is, containing the same assignment of the Search UIBB with the result list UIBB, but these UIBBs are flagged as “Technical: Hidden but processed in event loop”.

In addition you need to configure a Form UIBB based on feeder class `CL_GUIBB_BOL_ROOT_PARAM_CREATE` and parameterized on the BOL root object. Maintain the feeder parameter “Execution Mode” with “On FPM initial screen’s start-button event” so that, at runtime, the object creation is processed when the *Continue* button is pressed.

Edit Parameters

Feeder Class
 * Feeder Class: Generic BOL Feeder Form for Create Parameters

Parameters
 * Component Name:
 * Object Name:

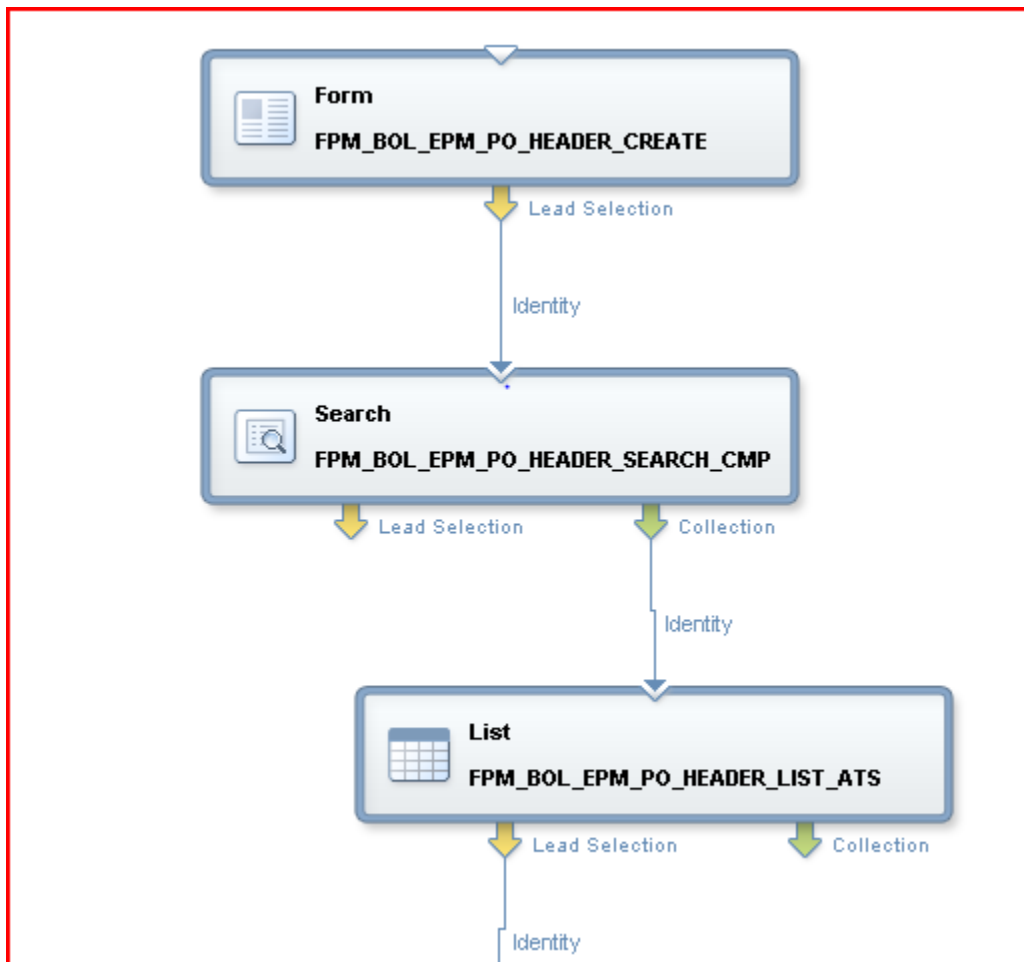
Join Structure

Join Structure	* Relation Na...	* Component ...	* Assigned Obj.	Suffix	Port Identifier

Execution Mode:

OK Cancel

In the OVP configuration, the search page should not be flagged as the default initial page since there is an alternative initial page. For the wiring, the 'creation' Form UIBB must be on top of the Search UIBB.



In the result list toolbar, you need to add a “New” button which also has to be assigned to the `FPM_NAVIGATE` event. The “Processing Mode” should be “Create”, the “Edit” mode should be selected, and the “Page ID” shall be maintained with the ID of the page carrying the create Form UIBB.

The screenshot shows the 'Attributes of Toolbar Element: New' configuration window. The 'Action Assignment' section is highlighted, showing the following settings:

- FPM Event ID:** FPM_NAVIGATE (External Navigation)
- Text:** New
- Processing Mode:** Create
- Edit Mode:** Edit
- Page Id:** CREATE

Having done this, the new button should open a new window displaying the initial screen with the creation Form UIBB, and once you enter create parameters and press 'Continue', you enter the overview page where you can further edit the new object.

Break-out Scenarios

With the FPM BOL adapter it is possible to create running applications without a line of code, that is, based purely on application-specific Web Dynpro ABAP configuration objects and reusing generic code such as feeder, connector and transaction handler classes. In application development it may, however, be necessary to add application-specific UI code. There are multiple options to do this. Most of them rely on the OO inheritance concept: the generic classes are modularized in a fine granular way, offering a variety of protected methods which can be redefined by application classes.

Feeder Class Redefinition

There are many options for redefining the standard feeder classes. The following example illustrates this. Suppose that we want to derive a specialized list feeder which has the following features:

1. It is specialized to a specific BOL object so that it cannot be parameterized on other BOL objects.
2. It does not allow joined objects.
3. It assigns a value help for a specific field as OVS.
4. It provides a specific set of values for the OVS.

To realize the first feature, we redefine method `IF_FPM_GUIBB~GET_PARAMETER_LIST` and remove the feeder parameters for the BOL component and object (see following screenshot).

Method	IF_FPM_GUIBB~GET_PARAMETER_LIST	Active
1	METHOD if_fpm_guibb~get_parameter_list.	
2		
3	<i>*----- inherit</i>	
4	rt_parameter_descr = super->if_fpm_guibb~get_parameter_list().	
5		
6	<i>*----- remove BOL component and object</i>	
7	DELETE rt_parameter_descr	
8	WHERE name = cv_param_component	
9	OR name = cv_param_object.	
10		
11	ENDMETHOD.	

We also set this data hard-coded by redefining method EVALUATE_PARAMETERS.

Method	EVALUATE_PARAMETERS	Active
1	METHOD evaluate_parameters.	
2		
3	<i>*----- set BOL component and object hard coded</i>	
4	ms_object_key-component_name = 'EPM'.	
5	ms_object_key-object_name = 'EPMPurchaseOrder'.	
6		
7	<i>*----- inherit</i>	
8	CALL METHOD super->evaluate_parameters	
9	EXPORTING	
10	it_parameter = it_parameter.	
11		
12	ENDMETHOD.	

The field catalog is now based on the EPM Purchase Order BOL object. The second feature is realized by returning ABAP_FALSE in a redefinition of IS_JOIN_ENABLED.

Method	IS_JOIN_ENABLED	Active
1	METHOD is_join_enabled.	
2		
3	<i>*----- do not allow joins</i>	
4	rv_enabled = abap_false.	
5		
6	ENDMETHOD.	

As a result, we can no longer assign different BOL objects in the GUIBB configuration.

The third feature is achieved by returning ABAP_TRUE in a redefinition of IS_OVS_ATTRIBUTE for a specific field.

```

Method IS_OVS_ATTRIBUTE Active
1  METHOD is_ovs_attribute.
2
3  *----- which attribute?
4  CASE iv_name.
5
6  *----- Currency
7  WHEN 'CURRENCY_CODE'.
8
9  *----- set ovs
10     rv_is_ovs_attribute = abap_true.
11
12 *----- anything else
13 WHEN OTHERS.
14
15 *----- inherit
16     rv_is_ovs_attribute = super->is_ovs_attribute( iv_name ).
17 ENDCASE.
18
19 ENDMETHOD.

```

Finally, we build up an OVS result list in a redefinition of OVS_HANDLE_PHASE_2 where we fill the output table referenced by the exporting parameter ER_OUTPUT.

```

Method OVS_HANDLE_PHASE_2 Active
1  METHOD ovs_handle_phase_2.
2
3  TYPES:
4  BEGIN OF s_output,
5      currency_code TYPE snwd_curr_code,
6      currency_text TYPE wdy_conf_transl_text,
7  END OF s_output.
8  TYPES:
9      t_output TYPE STANDARD TABLE OF s_output.
10
11 DATA:
12     ls_output TYPE s_output.
13
14 FIELD-SYMBOLS:
15     <lt_output> TYPE t_output.
16
17
18 *----- which attribute?
19 CASE iv_field_name.
20
21 *----- currency
22     WHEN 'CURRENCY_CODE'.
23         .
24 *----- currency table
25     CREATE DATA er_output TYPE t_output.
26     ASSIGN er_output->* TO <lt_output>.
27     ls_output-currency_code = 'USD'.
28     ls_output-currency_text = 'US Dollar'.
29     APPEND ls_output TO <lt_output>.
30     ls_output-currency_code = 'EUR'.
31     ls_output-currency_text = 'European Single currency'.
32     APPEND ls_output TO <lt_output>.
33
34
35 *----- anything else
36     WHEN OTHERS.
37
38 *----- inherit
39     CALL METHOD super->ovs_handle_phase_2
40     EXPORTING
41         iv_field_name      = iv_field_name
42         ir_query_parameter = ir_query_parameter
43     IMPORTING
44         er_output          = er_output
45         ev_table_header   = ev_table_header
46         et_column_texts   = et_column_texts.
47     ENDCASE.
48
49 ENDMETHOD.

```

This is enough to set up an application-defined value help.

Total Gross Amount	Currency Code	Business Pa
384,35	EUR	100000022
1.963,50	EUR	100000033
2.992,63	USD	100000004
641.723	JPY	100000006
467.881,70	MXN	100000008

Currency	Text
USD	US Dollar
EUR	European Single currency

Connector Class Redefinition

In special cases it may also be necessary to redefine methods of the standard FPM BOL connector classes. You can also derive directly from the abstract class `CL_FPM_CONNECTOR_BOL_BASE`.

Transaction-Handler Class Redefinition

In order to implement deviations of the transactional behavior of the FPM BOL adapter, it is also possible to derive from the standard transaction handler `CL_FPM_BOL_TRANSACTION`. For instance, you can redefine method `EVALUATE_SAVE_INTERNAL` in order to issue specific success messages upon successful save.

Freestyle UIBBs

The classes `CL_BOL_SAMPLE_FREESTYLE_ASSIST` and `CL_BOL_SAMPLE_FREE_ASSIST_TAB` can be directly used or redefined as assistance classes of freestyle UI building blocks on BOL objects. The implementation assumes that there is a context node containing (a subset of) the BOL object's attributes. It is required to pass it in method `WDDOINIT` of the component controller.

```

Method List | Method
Method | WDDOINIT
-----
1 | METHOD wddoinit .
2 |
3 | DATA:
4 |     lo_attr_node TYPE REF TO if_wd_context_node,
5 |     lo_wd_component TYPE REF TO if_wd_component.
6 |
7 |     lo_attr_node = wd_context->get_child_node( name = wd_this->wdctx_item_attr ).
8 |     lo_wd_component = wd_this->wd_get_api( ).
9 |     wd_assist->init(
10 |         iv_component_name = 'SAMPLE'
11 |         iv_object_name = 'OrderItem'
12 |         io_attr_node = lo_attr_node
13 |         io_wd_component = lo_wd_component ).
14 |
15 | ENDMETHOD.

```

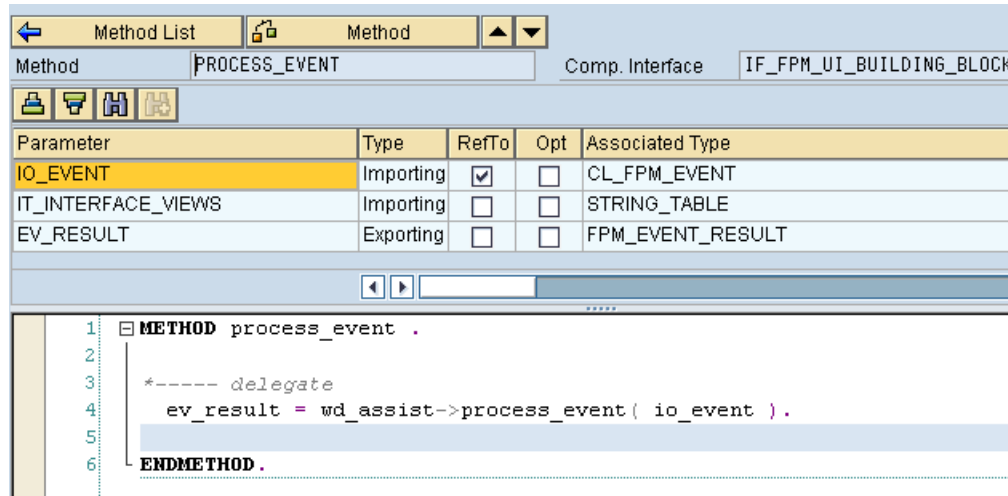
The interface `GET_MODEL_API` must simply return the assistance class.

```

1 | METHOD get_model_api .
2 |
3 |     ro_feeder_model = wd_assist.
4 |
5 | ENDMETHOD.

```

The interface methods `FLUSH`, `PROCESS_BEFORE_OUTPUT` and `PROCESS_EVENT` need simply to delegate to the assistance class, for example:



Parameter	Type	RefTo	Opt	Associated Type
IO_EVENT	Importing	<input checked="" type="checkbox"/>	<input type="checkbox"/>	CL_FPM_EVENT
IT_INTERFACE_VIEWS	Importing	<input type="checkbox"/>	<input type="checkbox"/>	STRING_TABLE
EV_RESULT	Exporting	<input type="checkbox"/>	<input type="checkbox"/>	FPM_EVENT_RESULT

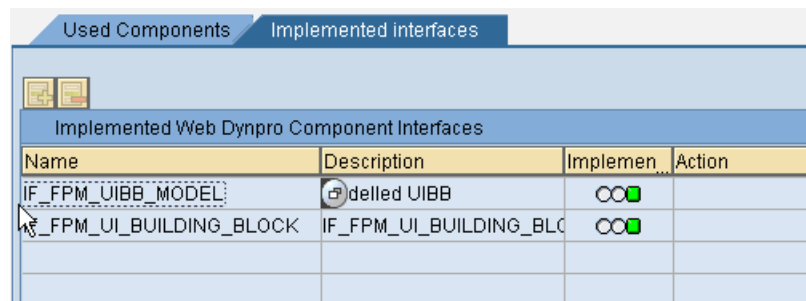
```

1  METHOD process_event .
2
3      *----- delegate
4      ev_result = wd_assist->process_event( io_event ).
5
6  ENDMETHOD .

```

The components `FPM_BOL_SAMPLE_FREESTYLE` and `FPM_BOL_SAMPLE_FREE_TAB` may be conferred as samples.

Create your WD component as a UI building block according to the FPM standard. The WD component needs to implement interface `IF_FPM_UIBB_MODEL`.



Name	Description	Implemen...	Action
IF_FPM_UIBB_MODEL	Modelled UIBB		
IF_FPM_UI_BUILDING_BLOCK	IF_FPM_UI_BUILDING_BLOC		

The method `IF_FPM_FEEDER_MODEL~GET_NAMESPACE` has to return the value 'BOL'.

Ty.	Parameter	Type spec.	Description
	VALUE(RV_NAMESPACE)	TYPE FPM_MODEL_NAMESPACE	Namespace for Wire

Method: IF_FPM_FEEDER_MODEL~GET_NAMESPACE Active

```

1  METHOD if_fpm_feeder_model~get_namespace .
2
3      rv_namespace = cl_gui_bol_base=>cv_namespace_bol .
4
5  ENDMETHOD .

```

The method `IF_FPM_FEEDER_MODEL~SET_CONNECTOR` is used to set the connector instance which serves as a data source. It should be kept in a local object variable of 'type ref to `IF_FPM_CONNECTOR`'.

Ty.	Parameter	Type spec.	Descri...
	IO_CONNECTOR	TYPE REF TO IF_FPM_CONNECTOR	

Method	IF_FPM_FEEDER_MODEL~SET_CONNECTOR	Active
1	METHOD if_fpm_feeder_model~set_connector.	
2		
3	mo_connector = io_connector.	
4		
5	ENDMETHOD .	

At runtime, the connector method GET_OUTPUT should be called on BEFORE_PROCESS_BEFORE_OUTPUT so that the actual data can be displayed. This can be implemented, for example, in a public method of the assistance class.

```
*----- connector set?
CHECK mo_connector IS BOUND.

*----- retrieve
mo_collection ?= mo_connector->get_output( ).
```

Now the collection (type ref to IF_BOL_ENTITY_COL) contains the current data and can be used to fill the WD context bound to the view elements.

The method IF_FPM_FEEDER_MODEL~GET_INPORT_KEY has to return the object key consisting of the component name and the object name relevant for the data retrieval.

Ty.	Parameter	Type spec.	Descri...
	VALUE(RR_OBJECT_KEY)	TYPE REF TO DATA	

Method	IF_FPM_FEEDER_MODEL~GET_INPORT_KEY	Active
1	METHOD if_fpm_feeder_model~get_inport_key.	
2		
3	FIELD-SYMBOLS:	
4	<ls_object_key> TYPE cl_gui_bol_base=>s_object_key.	
5		
6		
7	CREATE DATA rr_object_key TYPE cl_gui_bol_base=>s_object_key.	
8	ASSIGN rr_object_key->* TO <ls_object_key>.	
9	<ls_object_key>-component_name = 'SAMPLE'.	
10	<ls_object_key>-object_name = 'Order'.	
11		
12	ENDMETHOD .	

The method IF_FPM_FEEDER_MODEL~GET_OUTPORTS needs to provide the definition of the outports. Each port consists of a port type (Collection, Selection or Lead Selection), a freely chosen identifier, a descriptive text and the object key. If the freestyle component shall pass data to dependent UI building blocks, the corresponding outports have to be defined here.

Ty.	Parameter	Type spec.	Descri...
	ET_OUTPUT	TYPE TY_T_PORT	

Method	IF_FPM_FEEDER_MODEL~GET_OUTPUTS	Active
1	METHOD if_fpm_feeder_model~get_outputs.	
2		
3	DATA:	
4	ls_port LIKE LINE OF et_output.	
5		
6	FIELD-SYMBOLS:	
7	<ls_object_key> type cl_gui_bol_base=>s_object_key.	
8		
9		
10	ls_port-type = if_fpm_feeder_model=>cs_port_type-collection.	
11	ls_port-identifier = 'STANDARD'.	
12	ls_port-description = text-100.	
13	CREATE DATA ls_port-object_key type cl_gui_bol_base=>s_object_key.	
14	ASSIGN ls_port-object_key->* TO <ls_object_key>.	
15	<ls_object_key>-component_name = 'SAMPLE'.	
16	<ls_object_key>-object_name = 'Order'.	
17	APPEND ls_port TO et_output.	
18		
19	ENDMETHOD.	

If the feeder class model has outputs, the method IF_FPM_FEEDER_MODEL~GET_OUTPUTPORT_DATA is used to access the output data. This has to be a BOL collection (type ref to IF_BOL_ENTITY_COL).

Ty.	Parameter	Type spec.	Description
	IV_PORT_TYPE	TYPE FPM_MODEL_PORT_TYPE	
	IV_PORT_IDENTIFIER	TYPE FPM_MODEL_PORT_IDENTIFIER	FPM: Model Port Identifier
	VALUE(RO_DATA)	TYPE REF TO OBJECT	

Method	IF_FPM_FEEDER_MODEL~GET_OUTPUTPORT_DATA	Active
1	METHOD if_fpm_feeder_model~get_outputport_data.	
2		
3	ro_data = mo_collection.	
4		
5	ENDMETHOD.	

Application Controllers

The FPM BOL adapter makes use of the wire model transaction-handler and does not need an application controller. If necessary, an application can assign its own application controller. It is possible for an application controller to receive UI information from the wiring if a WD ABAP application controller implements IF_FPM_UIBB_MODEL and provides a feeder model, or if an ABAP OO application controller implements IF_FPM_UIBB_MODEL (the feeder model must return the namespace 'BOL').

Special Topics

FPM BOL CHIP Integration

The standard BOL feeder classes define tags for each attribute. The tag name is by default the BOL attribute name. This can be changed by redefining method `GET_FIELD_DESCRIPTION`. As is standard, the tagging for a field can be activated in the attribute view in the GUIBB configuration in FLUID.

All standard model classes, in particular the feeder classes, are ready for multi-instantiability and implement the chip feeder interface `IF_FPM_CHIP_FEEDER`. A special form feeder class `CL_GUIBB_BOL_FORM_CHIP_ENTRY` is provided which can be configured on (dynamic) queries. This class reacts to tagging port events with a tag name coincided with a (dynamic) query attribute. Deviating names can be defined by redefining method `MAP_QUERY_PARAM_TO_ATTR`. It executes the query at each such inport event and, from the result collection – which ideally contains a single entity if the tagging is defined in a meaningful way – the first entity is displayed.

Example: If you have a query on Products, and there is a query attribute for the unique key field `PRODUCT_ID`, then, when an inport is received for a tag `PRODUCT_ID` with value '4711', the form displays the product data of the product with ID '4711'. This may be realized, for example, by assigning the Form UIBB in a UCW configuration which, in turn, is assigned in a WD CHIP. The CHIP may be assigned in a side panel configuration.

Appendix III: Guidelines for Edit Scenarios for List ATS UIBB

Objectives

The new List ATS UIBB provides, with the help of ABAP Table Services (ATS), sorting and filtering services. These services do not work directly on the data table of the application; they copy the required information into their own and local structures. They compute and manage the mapping between rows of the data table (source) and rows shown on the UI (result). Therefore, the feeder class cannot presuppose any connection between the data table and the UI unless the execution of services is forbidden.

The following questions arise if edit scenarios are to be realized:

- How do I handle a row on the UI if the value of a sorted column is changed and the new value is not in the sort order?
- How do I handle a row on the UI if the value of a filtered column is changed and the new value does not fulfill the filter criteria?
- Where should a row on the UI be displayed which is inserted by the feeder class?

The desired behavior of the UI is outlined in the UI Guidelines 2.0. In brief, these guidelines state that for edit scenarios the UI must be stable after values are changed. This can be summarized as follows:

- If values in a row are changed, the row maintains its position on the UI independent of the kind of change.
- If rows are inserted into the data table by feeder class, these rows appear on the UI after the last selected row. Additionally, the order of the rows on the UI is not changed.
- If rows are deleted from the data table by feeder class, these rows are removed from the UI. Additionally, the order of rows on the UI is not changed.

Therefore, after changes to values (on the UI or by feeder class) or after deletion or insertion of rows (by feeder class), the services (sorting and filtering) should not be executed. However, these services manage the mapping between the data table and the UI. Therefore, the services must be informed about changes to the data table in order to actualize the mapping.

This document describes how to realize edit scenarios which allow the usage of services (sorting and filtering) and which conform to the current UI guidelines (v2.0).

Prerequisites

- An application works in roundtrips (phase model of FPM). The feeder class method GET_DATA is called once per roundtrip. The data is shown (and perhaps changed) on the UI between two calls of GET_DATA.
- The insertion and deletion of rows in the data table is only made in the feeder class method GET_DATA. Therefore, between two calls of this method the number of rows in the data table is constant.
- Changes to values in any rows are possible (in method GET_DATA and on UI).

Note:

The following scenarios and programming examples must be extended if changes to the data table are made in the feeder class method FLUSH (in fact forbidden but possible).

Change Log

A new change log has been created which contains all changes needed for actualization of internal data of ATS. The starting point is the *before* image of the data table (on entry of method GET_DATA) and the end point is the *after* image of the data table (after leaving the method GET_DATA). The editor change log describes the transition from the *before* to *after* image.

The change log must be created by the application and then handed over to ATS. The change log is described by the interface IF_SALV_ITAB_CHANGE_LOG. Its methods are described in the following table and screenshot:

Method	Description
get_index_map	Gets the indexes of deleted rows (<i>before</i> image), the indexes of inserted rows (<i>after</i> image), the mapping of rows between the <i>before</i> and <i>after</i> image and the indexes of rows (<i>before</i> image) which must be moved to an insert position
get_columns_modified	Gets the names of columns in which values are changed
get_rows_modified	Gets the indexes of rows (<i>after</i> image) in which values are changed
data_is_new	This is TRUE if the data is completely new and no connection between the <i>before</i> and <i>after</i> image exists.

```
interface if_salv_itab_change_log.
```

```
methods:
```

```

get_index_map
  exporting et_deleted type yt_range           " line numbers (before image)
           et_inserted type yt_range          " line numbers (after image)
           et_moved    type yt_range_map      " mapping of lines (before to after image)
           et_move_to_insert_position type yt_row_index, " lines (before image) for moving to insert position

get_columns_modified
  exporting e_type      type y_modification_type " unknown, see table
           et_field     type yt_field_path,      " column names

get_lines_modified
  exporting e_type      type y_modification_type " unknown, see table
           et_line_index type yt_range,          " line numbers (after image)

data_is_new
  returning value(r_is_new) type abap_bool.    " true if new query was performed

```

Note

- See also the interface documentation in the system
- The methods GET_COLUMNS_MODIFIED and GET_ROWS_MODIFIED are optional. The change log does not have to supply this information. But the ATS works faster if this information is available.
- Row numbers are always described as ranges in the change log. If a range consists of only one row, the lower and the upper value are the same.
- The mapping of rows (ET_MOVED) describes how to map ranges of rows of the *before* image to ranges of rows of the *after* image. The range in the *after* image is described only by the lower value, as the following example demonstrates:

((3, 5), 7) means that rows 3, 4 and 5 of the *before* image are mapped to rows 7, 8 and 9 of the *after* image.

- Entries in table ET_MOVED of method GET_INDEX_MAP are only needed for rows which change their position in the data table between the *before* and *after* image. Rows which do not change their position do not have to be listed in this table.

Application Scenarios

The creation of a change log by feeder classes is supported by special classes of ATS. Dependent on this support, there are different application scenarios:

- **Unique Key Mode**
The rows of the data table have a unique key. The key can consist of an arbitrary number of fields. You can edit the values of the key but the uniqueness of the key must always be guaranteed for the whole data table. Additionally, arbitrary operations with the table are allowed.
- **Stable Line Mode**
The application does not change the order of rows in the data table but insert and delete operations are allowed.

In the above scenarios, the application can delegate the creation of change logs to helper classes of ATS (see below). The application must guarantee special conditions (agreements). Violation of these agreements leads to erroneous mapping between the *before* and *after* image and, therefore, a correct mapping cannot be calculated in all cases. Therefore, conformity with the current UI guidelines is not guaranteed. However, no data will be lost.

- **Own Delta Handling**
There is no support by ATS; the application is responsible for the creation of a correct change log.
- **No Delta Handling**
There is no delta handling; after changes to data the services (sorting and filtering) are always executed. This is the behavior of the 'old' List UIBB. This behavior does not conform to current UI guidelines.

Note

The existing feeder classes can be used furthermore with the new List ATS UIBB without changes. This corresponds to the 'no delta handling' scenario.

Extension of Feeder Interface

The signature of the method GET_DATA of the feeder interface is extended with an exporting parameter. This parameter must reference an object which implements the interface for the editor change log.

```
methods:
  get_data
    " ...
    exporting"...
      eo_itab_change_log type ref to if_salv_itab_change_log
    "...
```

Together with the existing export parameter `EV_DATA_CHANGED` the following situations are possible:

<code>EV_DATA_CHANGED</code>	<code>eo_itab_change_log</code>	Meaning
FALSE	is not observed	Data not changed; change log is not observed → Services are not executed
TRUE	initial	Data changed; change log not calculated → Services are executed at new
	not initial : <code>data_is_new() = true</code>	Data changed; change log calculated but <code>data_new</code> is set → Services are executed at new
	not initial : <code>data_is_new() = false</code>	Data changed; change log calculated (access via <code>get_index_map</code>) → Services are not executed

Unique Key Mode

This section describes what an application must do to realize the scenario *Unique Key Mode*.

To-Dos for Application

- The flag `EV_DATA_CHANGED` must always be set if changes to data are made inside the method `GET_DATA`.
 - The flag will not be set if changes are made by UI and accepted by the application.
 - The flag must also be set if only the order of the rows is changed (for example, by sorting the table).
- The uniqueness of the key must always be ensured.
 - Applies also for changes of keys on the UI.
 - Applies also for insert operations, that is, new inserted rows should already have a unique key.
- For the creation of the editor change log, the class `CL_SALV_ITAB_EDITOR_KEY_MODE` can be used.
 - See below for programming examples
 - See also the class documentation in the system



A violation of uniqueness results in erroneous mapping between rows of the *before* and *after* images. However, no data is lost.

The methods for class `CL_SALV_ITAB_EDITOR_KEY_MODE` are described in the following table:

Method	Description
<code>Log_new_data</code>	Resets the change log (<code>DATA_IS_NEW</code> returns TRUE)
<code>START_RECORDING</code>	Starts recording of change log; an existing change log is deleted
<code>KEY_CHANGED</code>	Notifies that a key was changed
<code>STOP_RECORDING</code>	Stops recording of change log
<code>move_to_insert_position</code>	Marks a row for moving to an insert position

Notes

- The constructor of this class has the following parameters:
 - IT_KEY_NAME = names of key fields
 - I_UI_CAN_CHANGE_KEY_FIELDS = flag indicating whether changes of key fields are possible by UI

If no such changes are possible, a quicker calculation of the change log is possible.

- The start of recording with the method START_RECORD is absolutely necessary. The actual state of the data table must be handed over as a parameter at the start.
- The calculation of the change log is expensive. It takes place either during a call of method STOP_RECORDING or, if the method STOP_RECORDING is not called before, during a call of 'get' methods of interface IF_SALV_ITAB_CHANGE_LOG (see above).
- The method STOP_RECORDING should be used to stop the recording; further accesses to the change log (using 'get' methods) are realized more quickly as the change log does not have to be calculated for each access.
But in principle the method STOP_RECORDING is not required.
- The call of method START_RECORDING deletes an existing change log. Therefore, the method START_RECORDING may be called once within the method GET_DATA.
- The notification of key changes is essential for the calculation of a correct change log. If a key is changed and no notification is made, it is assumed that any row is deleted and any row is inserted. This is different behavior to changing a key (no new row, and row is moved).
- If you want to notify that new data exists using the change log, you can use the method LOG_NEW_DATA (see section '*Extension of Feeder Interface*').

Moving of Rows

As described above, the application knows no connection between the data table and the UI; it does not affect the order in which the rows of the data table are displayed on the UI. In certain situations, it is necessary to move rows around on the UI. For example, when implementing drag-and-drop for rows, it must be possible to mark those rows in the data table which have to be moved in the UI to an insert position. For this reason, the class CL_SALV_ITAB_EDITOR_KEY_MODE has the method MOVE_TO_INSERT_POSITION. This method can be passed a line that appears in the UI at the insertion position. Several calls to this method with different rows are possible. These rows are shifted simultaneously. Changes to the data table are not necessary.

Programming Examples

The following programming example shows the principal construction of method GET_DATA if the class CL_SALV_ITAB_EDITOR_KEY_MODE is used for creating the change log. The application must start and stop the recording, must notify key changes and must export the change log using the exporting parameter.

```

data: mo_editor type ref to cl_salv_itab_editor_key_mode. " implementing if_salv_itab_change_log

methods get_data. " exporting ...
                "      ev_data_changed type boole_d
                "      ...
                "      eo_itab_change_log type ref to if_salv_itab_change_log
                " changing ct_data type data
                "      ...

data: lt_key_name type if_salv_service_types=>yt_field_path, " names of key_fields
      l_line      type ys_data,                          " line of ct_data
      l_line_old  type ys_data.                          " line of ct_data

if mo_editor is initial.
*   lt_key = ...
    create object mo_editor
        exporting it_key_name          = lt_key_name
                  i_ui_can_change_key_fields = abap_true.
endif.
mo_editor->start_recording( ct_data ).

...
" change of key
line_old = line.
*   line = ...
    mo_editor->key_changed( line_before = line_old
                          line_after  = line ).
...

mo_editor->stop_recording( ).
if ev_data_changed = abap_true.
    eo_itab_change_log = mo_editor.
else.
    clear eo_itab_change_log.
endif.

endmethod.

```

Notes

- The constructor and the methods START_RECORDING and KEY_CHANGED can throw exceptions. In the programming example above, the handling of exceptions has been waived.

Stable Line Order Mode

This section describes what an application must do to realize the scenario Stable Line Order Mode.

To-Dos for Application

- The flag `EV_DATA_CHANGED` must always be set if changes on data are made inside the method `GET_DATA`.
 - The flag will not be set if changes are made by UI and accepted by application.
- The order of rows in the data table must not be changed.
 - Sorting is forbidden.
 - Do not call routines (methods, function modules) which change the order of rows in the table.
 - Insert and delete operations are possible. However, no mapping is detected if a row is deleted and then inserted.
- The class `CL_SALV_ITAB_EDITOR_LINE_MODE` can be used for creating the editor change log
 - See programming examples below
 - See also class documentation in the system.
 - This class contains methods for all table operations (except for `LOOP` and `READ`) there are methods of this class.
 - It is possible to execute all table operations themselves and only use the log functions of this class

Notes

- Changes in row order lead to erroneous mapping between the *before* and *after* images. However, no data is lost.
- If, in certain situations (for example during sorting), there is a need to change the order of table rows, it should proceed as if there is new data (run method `RESET`).

The methods for class CL_SALV_ITAB_EDITOR_LINE_MODE are described in the following table:

Method	Description
START_RECORDING	Starts recording of change log; an existing change log is deleted
SET_NEW_DATA	Gets new data and resets the change log (DATA_IS_NEW returns TRUE)
<i>Table operations with recording</i>	
APPEND_...	Substitute for ABAP statement APPEND (3 variants)
CLEAR_TABLE	Substitute for ABAP statement CLEAR
COLLECT_LINE	Substitute for ABAP statement COLLECT
DELETE_...	Substitute for ABAP statement DELETE (2 variants)
INSERT_...	Substitute for ABAP statement INSERT (3 variants)
MODIFY_LINE	Substitute for ABAP statement MODIFY
MOVE_TO_INSERT_POSITION	Marks a row for moving to an insert position
<i>Only recording of table operations</i>	
LOG_APPEND_ROWS	Records appended rows
LOG_DELETE_ROWS	Records deleted rows
LOG_INSERT_ROWS	Records inserted rows
LET_NEW_DATA	Resets the change log (data_is_new returns true)
MOVE_TO_INSERT_POSITION	Marks a line for moving to an insert position

Notes

- The start of recording with the method START_RECORD is absolutely necessary. During the start the actual state of the data table must be handed over as parameter. A method for stopping the recording is not needed and does not exist.
- The call of method START_RECORDING deletes an existing change log. Therefore, the method START_RECORDING may be called once within the method GET_DATA.
- The method LOG_NEW_DATA can be used if you want to notify using the change log that new data exists (see section '*Extension of Feeder Interface*').
- This class provides a complete set of operations to change a table. Reading operations (READ, LOOP) must be executed directly using ABAP statements.
- All operations that change a table can be executed as ABAP statements too if, after each statement, the corresponding recording method LOG_... is called. However, it is recommended that changes to a table are always made with the appropriate methods for such table operations.

Moving of Rows

As described above, the application knows no connection between the data table and the UI, that is, it does not affect the order in which the rows of the data table are displayed on the UI. In certain situations it is necessary to move rows around the UI. For example, when implementing the drag-and-drop of rows, it must be possible to mark those rows in the data table which have to be moved in the UI to an insert position.

For this reason, the class CL_SALV_ITAB_EDITOR_LINE_MODE has the methods MOVE_TO_INSERT_POSITION and LOG_MOVE_TO_INSERT_POSITION. These methods can be passed a row number that appears in the UI at the insert position. Several calls to these methods with different row numbers are possible. These rows are

shifted simultaneously. Changes to the data table are not necessary, that is, it is not necessary to move these rows inside the data table.

Programming Examples

The following code demonstrates how to replace ABAP statements for changing table content using calls of methods of class CL_SALV_ITAB_EDITOR_LINE_MODE:

<code>append initial line to ct_data.</code>	<code>io_editor->append_initial_line().</code>
<code>append l_data to ct_data.</code>	<code>io_editor->append_line(i_line = l_data).</code>
<code>append lines of lt_data from l_from to l_to to ct_data.</code>	<code>io_editor->append_lines(it_lines = lt_data i_from = l_from i_to = l_to).</code>
<code>collect l_data into ct_data.</code>	<code>io_editor->collect_line(i_line = l_data).</code>
<code>delete ct_data index l_idx.</code>	<code>io_editor->delete_line(i_index = l_idx).</code>
<code>delete ct_data from l_from to l_to.</code>	<code>io_editor->delete_lines(i_from = l_from i_to = l_to).</code>
<code>insert initial line into ct_data index l_idx.</code>	<code>io_editor->insert_initial_line(i_index = l_idx).</code>
<code>insert l_data into ct_data index l_idx.</code>	<code>io_editor->insert_line(i_line = l_data i_index = l_idx).</code>
<code>insert lines of lt_data from l_from to l_to into ct_data index l_idx.</code>	<code>io_editor->insert_lines(it_lines = lt_data i_from = l_from i_to = l_to i_index = l_idx).</code>

Notes

- If the corresponding ABAP statement returns any system fields (SY-...), the method has an exporting or returning parameter with the value of this system field (see the signatures of the methods).
- Regarding the validity of indexes, the same rules apply as for the corresponding ABAP statements. Therefore, invalid values for insert indexes, for example, lead to dumps as they would in the corresponding ABAP statement.
- Most of the methods above can throw exceptions. The handling of exceptions has been waived in the example code above.

The following programming example shows the principal construction of the method GET_DATA if the class CL_SALV_ITAB_EDITOR_LINE_MODE is used for creating the change log. The application uses the methods for table operations (this is demonstrated by the deletion and insertion of a line in this example):

```

data: mo_editor type ref to cl_salv_itab_editor_line_mode. " implementing if_salv_itab_change_log

methods get_data. " exporting ...
                "          ev_data_changed type boole_d
                "          ...
                "          eo_itab_change_log type ref to if_salv_itab_change_log
                " changing ct_data type data
                "          ...

data: l_line type ys_data.          " line of ct_data

if mo_editor is initial.
  create object mo_editor.
endif.
mo_editor->start_recording( changing ct_data = ct_data ).
...
mo_editor->delete_line( i_index = 5 ).
...
mo_editor->insert_line( i_line   = l_line
                      i_index  = 3 ).
...
if ev_data_changed = abap_true.
  eo_itab_change_log = mo_editor.
else.
  clear eo_itab_change_log.
endif.

endmethod.

```

Note that the handling of exceptions is waived in the above example.

In the following programming example the table operations are made by ABAP statements and then recorded by recording methods:

```

data: mo_editor type ref to cl_salv_itab_editor_line_mode. " implementing if_salv_itab_change_log

methods get_data. " exporting ...
      "          ev_data_changed type boole_d
      "          ...
      "          eo_itab_change_log type ref to if_salv_itab_change_log
      " changing ct_data type data
      "          ...

data: l_line type ys_data.          " line of ct_data

if mo_editor is initial.
  create object mo_editor.
endif.
mo_editor->start_recording( changing ct_data = ct_data ).
...
  delete ct_data index 5.
  mo_editor->log_delete_lines( i_index      = 5
                             i_line_number = 1 ).
...
  insert l_line into ct_data index 3.
  mo_editor->log_insert_lines( i_index      = 3
                             i_line_number = 1 ).
...
if ev_data_changed = abap_true.
  eo_itab_change_log = mo_editor.
else.
  clear eo_itab_change_log.
endif.

endmethod.

```

Note that the handling of exceptions is waived in the example above.

Both the programming examples above have a drawback; table operations using ABAP statements can be mixed with calls of methods for table operations. This increases the occurrence of erroneous change logs. The following example avoids this problem and forces you to use methods for table operations in all cases:

```

data: mo_editor type ref to cl_salv_itab_editor_line_mode. " implementing if_salv_itab_change_log

methods get_data. " exporting ...
                "          ev_data_changed type boole_d
                "          ...
                "          eo_itab_change_log type ref to if_salv_itab_change_log
                " changing ct_data type data
                "          ...

if mo_editor is initial.
    create object mo_editor.
endif.
mo_editor->start_recording( changing ct_data = ct_data ).
get_data_internal( exporting"...
                  io_editor = mo_editor
                  it_data   = ct_data "changes only by io_editor possible
                  "...
                  importing ev_data_changed = ev_data_changed
                  "...
                                ).

if ev_data_changed = abap_true.
    eo_itab_change_log = mo_editor.
else.
    clear eo_itab_change_log.
endif.

endmethod.

```

In the above example the entire code is moved to method GET_DATA_INTERNAL. The data table is transferred to this method using an importing parameter. Therefore, inside the method GET_DATA_INTERNAL, ABAP statements for changing the table are no longer allowed. All changes relate to calls of editor methods.

Own Delta Handling

This section describes what an application must do to realize the scenario *Own Delta Handling*. The application is responsible themselves for creation of change log.

To-Dos for Application

- The flag EV_DATA_CHANGED must always be set if changes to data are made inside the method GET_DATA.
 - The flag will not be set if changes are made by UI and accepted by the application.
- If the data table is changed, a change log must be created. The recording methods (LOG_...) of class CL_SALV_ITAB_EDITOR_LINE_MODE may be used. This corresponds with the second example of the preceding section.

No Delta Handling

This case describes the behavior of the previous ('old') List UIBB.

To-Dos for Application

- The flag EV_DATA_CHANGED must always be set if changes to data are made inside the method GET_DATA.
 - The flag will not be set if changes are made by UI and accepted by application.

Note that the existing feeder classes can be used with the new List ATS UIBB without changes. But, of course, the behavior during edit scenarios does not conform to the current UI guidelines under certain conditions.

Appendix IV: Multi-Value Fields

How to Use a Multi-Value Field

With a multi-value field, a user has the opportunity to define a user-specific input field containing multiple input data.

The different input values are entered in a dialog box. This data is formatted by the FPM and is displayed on the search screen.

The screenshot below details a search screen at runtime, detailing the multi-value elements:

Search Criteria

free text search for company: is [] +

business partner last name is [] + -

business partner first name is [] + -

the region of the business partner is [DE - MUNICH
US - NEW YORK CITY] + -

Maximum Number of Results: 100

Save Search As: []

Search Clear Entries Reset to Default

Hide Search Criteria

search criteria with multi-value data

Formatted multi-value input

Enter the MV-data

The application defines the dialog box which the users must use to enter the multi-value data. An example of such a dialog box is shown below:

Partner via Region

Country: DE City: MUNICH

Country: US City: NEW YORK CITY

Close & Send

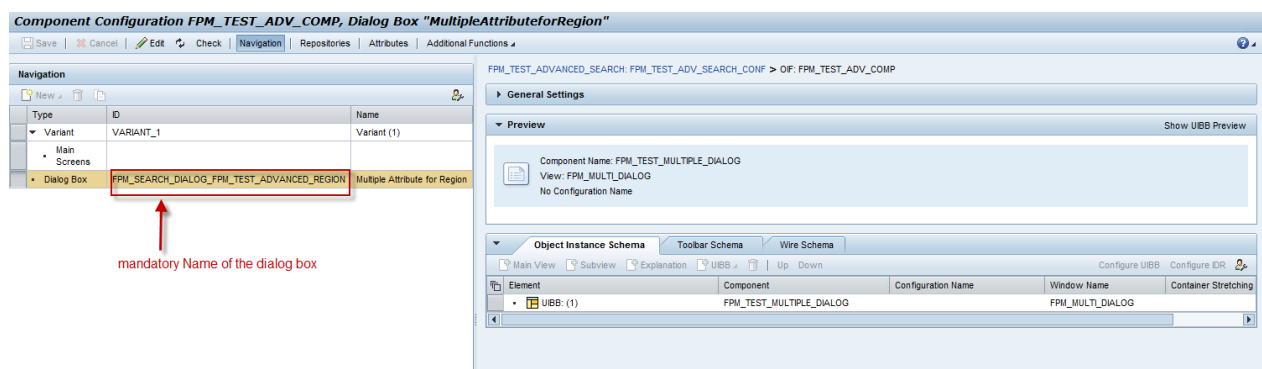
This is the pop up to enter the multi-value data.
It is defined by the application.

In order to use such a multi-value (MV) field, you must perform the following steps:

Example in R/3- system:

Web Dynpro application configuration: FPM_TEST_ADV_SEARCH_CONF
 Component configuration of Search UIBB: FPM_TEST_ADVANCED
 Web Dynpro component of dialog box: FPM_TEST_MULTIPLE_DIALOG

1. In method `GET_DEFINITION`, define in the field catalog a field with type string (in our example `s_business_partner-region`).
2. In method `GET_DEFINITION`, enter the attribute `FIELD_DESCRIPTION-multi_value_struct` for the MV criteria. This field is of type `CL_ABAP_STRUCTDESCR`. The components of this object are the input values of the dialog box. The simplest way to provide these fields is to create a flat DDIC structure containing the fields and then get it via `FIELD_DESCRIPTION-MULTI_VALUE_STRUCT ?= CL_ABAP_STRUCTDESCR=>DESCRIBE_BY_DATA(...)`
 If the MV field is to read-only, set the parameter `ES_OPTIONS-SET_MULTI_ATTR_TE_READ_ONLY = ABAP_TRUE`.
3. Create a Web Dynpro component for the dialog box where the user can enter the data for the multi-value field. In our example, `FPM_TEST_MULTIPLE_DIALOG`. The interface `IF_FPM_UI_BUILDING_BLOCK` must be implemented. Design the view with your input fields.
4. In the component controller of the WD component, define the context with your input fields.
5. Map the context from the component controller to the view context.
6. Open the FPM component configuration (in our example `FPM_TEST_ADV_COMP`). Open the *Navigation* panel. Add a dialog box. The name of the dialog box must be concatenated in the following way:
`FPM_SEARCH_DIALOG_<config_key-config_id>_<name of MV attribute>`
 In our example, this name is: `FPM_SEARCH_DIALOG_FPM_TEST_ADVANCED_REGION`.



If you do not want to use this concatenated name for the dialog box ID, you can set the parameter `ES_OPTIONS-USE_STD_DIALOG_MULTI_EDIT = ABPA_TRUE` in the feeder class method `GET_DEFINITION`. Then you can use the dialog box ID `FPM_SEARCH_STD_DIALOG` instead of the concatenated ID.

7. When you close the dialog box, you must raise the FPM event

IF_FPM_CONSTANTS=>GC_EVENT-CLOSE_DIALOG. In our example, this is done in the DIALOG_VIEW in the method ONACTIONSEND_CLOSE.

8. You must implement the method PROCESS_EVENT in the component controller. Here you can react on the FPM event IF_FPM_CONSTANTS=>GC_EVENT-CLOSE_DIALOG. You must create and raise the FPM event IF_FPM_CONSTANTS=>GC_EVENT-MULTIPLE_VALUE. With this event, you pass the multi-value data entered in the dialog box to the Search UIBB component.

You must set the following values for this event:

```
lo_event->mo_event_data->
set_value( iv_key   = if_fpm_guihb_search=>event_param_multi_name
           iv_value = <name of MV attribute> ).
           lo_event->mo_event_data->
set_value( iv_key   = if_fpm_guihb_search=>event_param_multi_struct
           iv_value = <data entered for MV attribute>).
```

In the search component, the values passed by this event are displayed automatically formatted. The separation between the two input fields from the dialog box is defined in ES_OPTIONS-MULTI_ATTR_SEPERATOR.

9. There is the option that it is not the search component that formats the multi-value string, but the application; the application passes a formatted multi-value string table. This string table will be displayed in the search component. The application passes this string table as an event parameter (IF_FPM_GUIHB_SEARCH=>EVENT_PARAM_MULTI_STRING_TABLE) of the event IF_FPM_CONSTANTS=>GC_EVENT-MULTIPLE_VALUE. This can be done in the component controller in method PROCESS_EVENT:

```
lo_event->mo_event_data->set_value(
EXPORTING iv_key   = if_fpm_guihb_search=>event_param_multi_string_table
           iv_value = <name of string table with formatted values>).
```

10. You must check for the FPM event IF_FPM_CONSTANTS=>GC_EVENT-OPEN_DIALOG in the method PROCESS_BEFORE_OUTPUT of the component controller of your MV WD component. The event parameters contain the input of the multi-value field in the search component. You need this data in order to fill the input fields of your dialog box correctly. Reading of the event parameters is done in the following way:

```
io_event->mo_event_data->get_value(
EXPORTING iv_key   = if_fpm_guihb_search=>event_param_multi_name
IMPORTING ev_value = <name of multi value attribute>).
io_event->mo_event_data-> get_value(
EXPORTING iv_key   = if_fpm_guihb_search=>event_param_multi_struct
IMPORTING er_value = <values of multi-value attribute>).
```

In our example this is done in the method PROCESS_BEFORE_OUTPUT of the component controller in the WD application FPM_TEST_MULTIPLE_DIALOG.

