



Holger Polch

# SAP Transportation Management 9.x Enhancement Guide

- An overview on the available enhancement techniques
- Coding and Configuration examples
- Tips & Tricks

## Table of Content

<b>1</b>	<b>INTRODUCTION .....</b>	<b>7</b>
1.1.1	Welcome to Edition 2 .....	7
1.1.2	What has been added, adjusted & removed? .....	7
<b>2</b>	<b>GLOSSARY .....</b>	<b>11</b>
<b>3</b>	<b>BOPF - OVERVIEW AND ARCHITECTURE .....</b>	<b>13</b>
<b>3.1</b>	<b>BOPF - Business Object Processing Framework .....</b>	<b>13</b>
3.1.1	BOPF Architecture .....	14
3.1.2	Business Object Model .....	15
3.1.3	BOPF Modeling Tool .....	18
<b>3.2</b>	<b>BOPF Consumer Implementation Basics .....</b>	<b>21</b>
3.2.1	Service Manager .....	21
3.2.2	Query .....	21
3.2.3	Retrieve .....	23
3.2.4	Retrieve By Association (Standard) .....	23
3.2.5	Retrieve By Association (XBO) .....	23
3.2.6	Retrieve By Association (Dependent Objects) .....	24
3.2.7	Do Action (Standard) .....	25
3.2.8	Do Action (Action Parameters) .....	26
3.2.9	Convert Alternative Key .....	26
3.2.10	Retrieve Property .....	27
3.2.11	Modify .....	27
<b>3.3</b>	<b>BOPF Enhancement Workbench .....</b>	<b>30</b>
3.3.1	Overview .....	30
3.3.2	First step: Creating an Enhancement Object .....	33
3.3.3	General remarks on creating enhancements .....	35
3.3.4	Creating Field Extensions .....	36
3.3.5	Creating Subnodes .....	38
3.3.6	Creating Actions .....	41
3.3.7	Creating Action Validations .....	43
3.3.8	Creating Pre- and Post-Action Enhancements .....	44
3.3.9	Creating Consistency Validations .....	46
3.3.10	Creating Determinations .....	48
3.3.11	Creating Queries .....	52
3.3.12	Creating custom Business Objects .....	55
<b>3.4</b>	<b>Advanced BOPF Topics .....</b>	<b>57</b>
3.4.1	Properties .....	57
3.4.2	Message Concept .....	60
3.4.3	Performance in the context of BOPF .....	62
3.4.4	Status & Action Management (Consistency Groups) .....	63
3.4.5	Change Document Adapter Enhancements .....	79
<b>4</b>	<b>TECHNIQUES FOR ENHANCING THE BUSINESS LOGIC .....</b>	<b>86</b>
<b>4.1</b>	<b>BADIs .....</b>	<b>86</b>
4.1.1	Where and how to find BADIs related to TM .....	86
4.1.2	Implementing a BADI .....	86
<b>4.2</b>	<b>Process Controller Strategies .....</b>	<b>90</b>
4.2.1	Relevant parts of the Process Controller .....	90
4.2.2	Setting up a Process Controller Strategy .....	91



4.2.3	Using the Process Controller Framework for a new process .....	95
4.2.4	Using Method Parameters .....	103
<b>4.3</b>	<b>Conditions .....</b>	<b>105</b>
4.3.1	Customizing: Condition Types and Data Access Definitions.....	105
4.3.2	Creating Data Access Definitions.....	106
4.3.3	Creating Condition Types .....	112
4.3.4	Assign Data Access Definitions to Condition Types .....	113
4.3.5	Creating Conditions .....	115
4.3.6	Simulating Conditions.....	118
4.3.7	Implementing a condition call in your coding .....	120
<b>4.4</b>	<b>Change Controller .....</b>	<b>122</b>
4.4.1	Basic Concept & technical aspects .....	122
4.4.2	Customizing settings for the Change Controller .....	123
4.4.3	Example Change Controller settings .....	124
4.4.4	The Change Controller and how it works at runtime .....	129
4.4.5	Enhancing the Change Controller.....	130
4.4.6	The Trigger Concept .....	132
<b>4.5</b>	<b>Implicit Enhancements.....</b>	<b>134</b>
4.5.1	Use Implicit Enhancements with care .....	134
4.5.2	Pre-, Post- and Overwrite Methods for existing methods.....	134
<b>4.6</b>	<b>Helper Classes provided by SAP TM .....</b>	<b>138</b>
4.6.1	How to find SAP TM Helper Classes .....	138
4.6.2	Why using SAP TM Helper Classes?.....	138
<b>5</b>	<b>USER INTERFACE ENHANCEMENTS .....</b>	<b>140</b>
<b>5.1</b>	<b>FPM – Floor Plan Manager .....</b>	<b>140</b>
5.1.1	User Interface Building Blocks .....	140
5.1.2	Feeder Classes .....	141
5.1.3	Wire Model.....	142
<b>5.2</b>	<b>FBI – Floor Plan Manager BOPF Integration .....</b>	<b>144</b>
5.2.1	FBI View (design time).....	144
5.2.2	FBI View Instance (runtime) .....	146
5.2.3	FBI Controller (runtime) .....	146
5.2.4	Conversion Classes .....	146
5.2.5	Exit Classes .....	147
<b>5.3</b>	<b>General remarks on user interface enhancements.....</b>	<b>148</b>
<b>5.4</b>	<b>Enhancing the User Interface .....</b>	<b>155</b>
5.4.1	Field Extensions .....	155
5.4.2	Adding a new action to a toolbar .....	163
5.4.3	Adding a new tab with data from a new BO subnode.....	167
5.4.4	Adding a new Action to the main tool bar .....	174
5.4.5	Adding a new Parameter Action with a Popup .....	177
5.4.6	Accessing and displaying data from external sources .....	185
5.4.7	Building a simple new User Interface .....	193
5.4.8	Copying a complete FPM-based Application.....	211
5.4.9	Adding a Web Dynpro Application to NWBC.....	213
<b>5.5</b>	<b>Transporting or removing UI enhancements.....</b>	<b>219</b>
<b>6</b>	<b>ENHANCING QUERIES AND POWL .....</b>	<b>221</b>

<b>6.1</b>	<b>Queries .....</b>	<b>221</b>
6.1.1	General concept .....	221
6.1.2	Maintaining the standard query enhancement table .....	222
6.1.3	BAdI for creation of query enhancement table entries.....	226
6.1.4	Example 1: Enhancing a Custom Query.....	226
6.1.5	Example 2: Enhancing a Generic Result Query.....	228
<b>6.2</b>	<b>POWL (Personal Object Work Lists) .....</b>	<b>233</b>
6.2.1	Creating a new POWL.....	233
6.2.2	The POWL Feeder Class .....	233
6.2.3	The POWL Action Class.....	246
6.2.4	The basic POWL Customizing .....	247
6.2.5	Creating POWL Queries .....	248
6.2.6	Additional POWL Customizing.....	252
6.2.7	Enhancing a standard POWL .....	253
6.2.8	POWL Maintenance Reports .....	255
<b>7</b>	<b>ENHANCING PRINT FORMS.....</b>	<b>255</b>
<b>7.1</b>	<b>Enhancing a standard form .....</b>	<b>258</b>
7.1.1	Enhancing the involved BO(s).....	258
7.1.2	Copying the standard form.....	259
7.1.3	Enhancing the Print Structure of a Form .....	259
7.1.4	Providing data to enhanced fields .....	262
<b>7.2</b>	<b>Adjusting the Layout.....</b>	<b>263</b>
7.2.1	Adobe LiveCycle Designer Installation.....	263
7.2.2	Placing additional content on the form layout.....	263
<b>7.3</b>	<b>Creating a new form .....</b>	<b>265</b>
7.3.1	Creating a print structure and table type .....	265
7.3.2	Creating a form interface .....	266
7.3.3	Creating the Adobe form .....	267
7.3.4	Creating required coding in the backend .....	270
<b>7.4</b>	<b>Output Management Adapter and PPF Configuration.....</b>	<b>276</b>
7.4.1	Overview and general concepts .....	276
7.4.2	Creating a PPF Action Profile and Action Definitions .....	279
7.4.3	Creating PPF Conditions .....	283
7.4.4	Maintaining Output Management Adapter Settings.....	285
7.4.5	Preparing an example print document.....	287
<b>8</b>	<b>ENHANCING SERVICES.....</b>	<b>292</b>
<b>8.1</b>	<b>General remarks on Service Enhancements .....</b>	<b>292</b>
8.1.1	Example Service Enhancement .....	293
8.1.2	Basic steps to enhance an Enterprise Service .....	295
<b>8.2</b>	<b>Development in System Landscape Directory (SLD) .....</b>	<b>295</b>
8.2.1	Creating a Product Version and Software Component .....	295
8.2.2	Defining dependencies between EnSWCV and SWCVs .....	298
<b>8.3</b>	<b>Development in Enterprise Service Repository (ESR) .....</b>	<b>300</b>
8.3.1	Importing the EnSWCV into ESR.....	300
8.3.2	Creating a Namespace in the EnSWCV .....	301
8.3.3	Create a Data Type in the EnSWCV. ....	302
8.3.4	Create the Data Type Enhancement for the TM side.....	304
8.3.5	Create an Enhancement Data Type for the ECC side. ....	307



<b>8.4</b>	<b>Development in the Backend Systems .....</b>	<b>311</b>
8.4.1	Generating the Enhancement Proxy Structure in ECC .....	311
8.4.2	Generating the Enhancement Proxy Structure in TM .....	314
<b>9</b>	<b>ENHANCING FURTHER OBJECTS, FEATURES &amp; FUNCTIONS.....</b>	<b>324</b>
<b>9.1</b>	<b>Gantt Chart for Planning Functionality.....</b>	<b>324</b>
<b>9.2</b>	<b>Transportation Charge Management Enhancements .....</b>	<b>329</b>
9.2.1	Adding a new scale base.....	329
9.2.2	Adding a new calculation base .....	329
9.2.3	Adding a new resolution base .....	329
<b>9.3</b>	<b>Master Data Objects .....</b>	<b>330</b>

## Disclaimer

This document outlines the SAP general product direction and should not be relied on in making a purchase decision.

This document is not subject to your license agreement or any other agreement with SAP. SAP has no obligation to pursue any course of business outlined in this document or to develop or release any functionality mentioned in this document / presentation. This document / presentation and SAP's strategy and possible future developments are subject to change and may be changed by SAP at any time for any reason without notice.

The information in this document is not a commitment, promise or legal obligation to deliver any material, code or functionality. This document is provided without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP assumes no responsibility for errors or omissions in this document, and shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of this document.

All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as for their dates, and they should not be relied upon in making purchasing or any other decisions.

**© Copyright 2017 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 SE. The information contained herein may be changed without prior notice.

# 1 Introduction

## 1.1.1 Welcome to Edition 2

The first edition of the Guide turned out to be quite helpful for customers, partners, consultants and developers to get started with their SAP TM projects (the first edition was requested more than 500 times so far). What started with TM 8.0 and the first successful *Go Lives* has in the meantime developed further and SAP TM 9.5 has been finalized.

While many aspects described in the first version of the Enhancement Guide are valid across the releases, some things have changed. As per TM 9.0 SAP NetWeaver 7.31 is used and with this, a few things have changed, especially in the area of the Configuration Editor for the FPM/FBI-based User Interfaces. With TM 9.5 SAP NetWeaver 7.50 is the foundation for the application. In case of differences between 7.31 and 7.50 I'll point out to this too.

This second edition therefore provides a completely revised chapter 5 dealing with User Interface enhancements based on the new/changed Configuration Editor environment. But also, the content of the other chapters was partly rewritten, adjusted and enhanced to sort out errors, make things a bit easier to understand and add additional information which was gathered during further projects, workshops and other occasions.

The target of the second edition is to describe the possibilities and used technologies to enhance SAP Transportation Management based on release 9.1™ and following releases. Again, it does not intend to provide a complete and detailed description of all possible enhancements.

It describes the usage of the available enhancement technologies based on some basic examples. These examples are chosen to be representative for similar enhancements in multiple areas of the application. In some cases, links to more detailed descriptions are provided. Overall, the document content is valid for all currently available SAP TM releases, i.e. 8.x, and 9.x.

## 1.1.2 What has been added, adjusted & removed?

The following table lists the topics and aspects that have been added, adjusted or removed in this second edition SAP TM Enhancement Guide compared to the first one:

Chapter	Topic	Type	Comment
1	Introduction	Adjusted	New introduction with notes on target TM Release and list of changes compared to edition 1 of the Enhancement Guide.
3.1.1	BOPF Architecture	Adjusted	Screenshots reworked.
3.1.3	BOPF Modeling Tool	Adjusted	A few screenshots reworked.
3.3.1ff	Overview BOPF Enhancement Workbench	Adjusted	Reworked the examples, their description and related screenshots. New features mentioned: Creating customer/partner-specific BOs.
3.3.10	Creating Determinations	New	New Determination Pattern available: Create Properties (available for Determinations added via Enhancement Workbench).
3.3.10	Creating Determinations	New	Remark on best practice for adding Determinations → Grouping of Determinations.
3.3.12	Creating custom BOs	New	The Enhancement Workbench allows creating customer/partner-specific Business Objects.
3.4	Advanced BOPF Topics	New	BOPF Properties, BOPF Message



			Concept and hints on Performance in the context of using BOPF.
3.4.4	Status & Action Management	New	Two examples based on real customer enhancement use cases: Using S&AM to let a Validation set a BO node status and Using a Consistency Group to prevent Save of a transaction in case of errors.
3.4.5	Change Document Adapter Enhancements	New	An example how to enhance an existing Change Document Object to also allow tracking of changes to Enhancement Node data.
4.3.5	Creating Conditions	Adjusted	Screenshots and description on how to create conditions as per TM 9.0. UIs have slightly changed.
4.3.6	Simulating Conditions	Adjusted	As per TM 9.0, the BRF+ screens for simulating conditions are fully integrated into the TM User Interface for handling conditions. Nevertheless transaction BRF+ can also still be used for these functionalities.
4.4	Change Controller	New	The section describes the involved concepts and customizing for the Change Controller and shows concepts for enhancing it.
4.5	Implicit Enhancements	Adjusted	Complete section rewritten and screen shots adjusted. One single how-to explanation for all available implicit enhancements.
4.6	Helper Classes provided by SAP TM	New	When enhancing the business logic via additional coding in BADIs, Implicit Enhancements, etc. some implementation tasks occur again and again. SAP TM provides a large number of so called Helper Classes that already provide reusable coding for accessing specific data and other functions.
5.2.1	FBI View (design time)	Adjusted	Screenshot of example FBI View updated.
5.3	General remarks on user interface enhancements	Adjusted	As per SAP TM 9.0 NW 7.31 is used that provides enhanced functionality to easily navigate to UI configurations to get them enhanced.
5.4	Enhancing the User Interface	Adjusted	The section and its sub sections is completely reworked and adjusted to the TM 9.0 NW 7.31 environment.
5.4.1	Field Extensions	Adjusted	Screen Shots adjusted to the current Component Configurator. Descriptions and example data reworked.
5.4.2	Adding a new action to a toolbar	Adjusted	Screen Shots adjusted to the current Component Configurator. Descriptions and example data reworked. A few more functional options of the Component Configurator are described. The example is now illustrating the usage of related views.
5.4.3	Adding a new tab with data	Adjusted	Screen Shots adjusted to the current

	from a new BO subnode		Configurator. Descriptions and example data reworked.
5.4.4	Adding a new Action to the main tool bar	Adjusted	Screen Shots adjusted to the current Configurator. Descriptions and example data reworked.
5.4.5	Adding a new Parameter Action with a Popup	Adjusted	Screen Shots adjusted to the current Configurator. Descriptions and example data reworked.
5.4.6	Accessing and displaying data from external sources	Adjusted	Screen Shots added. Descriptions and example data reworked.
5.4.7	Building a simple new User Interface	Adjusted	Screen Shots added. Descriptions and example data reworked. Adjusted to the current Configuration Editor
5.4.8	Copying a complete FPM-based Application	New	A specific Web Dynpro Application allows creating complete and deep copies of an existing FPM Application
5.4.9	Adding a Web Dynpro Application to NWBC	New	An example to integrate a new FPM-based (Web Dynpro) application into NWBC. The example integrates the application created in 5.4.7 based on a given PFCG role.
5.5	Transporting or removing UI enhancements	Adjusted	Screen Shots reworked and added some more detailed descriptions.
6.1	Queries	Adjusted	Reworked the description of the query enhancement concept and revised the enhancement examples (sorted out a few errors from the old versions).
6.2	POWL (Personal Object Work Lists)	Adjusted	Reworked the example how to create a completely new POWL and adjusted the example coding (including error corrections in the coding and an extension of the example).
6.2.5	Creating POWL Queries	New	Added a few comments on how to create specific settings for existing POWL Queries and saving them as Views.
6.2.6	Additional POWL Customizing	New	Described a few more customizing transactions that allow defining POWLs and the assignment to roles and/or users.
6.2.7	Enhancing a standard POWL	New	Based on the findings from section 6.2 some basic hints how to enhance existing standard POWLs.
7.1	Enhancing a standard form	Adjusted	Reworked description and screen shots.
7.2	Adjusting the Layout	Adjusted	Reworked description and screen shots.
7.3	Creating a new form	Adjusted	Reworked description and screen shots.
7.4	Output Management Adapter and PPF Configuration	Adjusted	Added additional details about PPF, Output Management and some general remarks to improve understandability of the configuration example in specific and PPF configuration in general. All screen shots have been updated to reflect

			the UI changes that came along with TM 9.x
7.4.1	Overview and general concepts	New	This section provides an overview of the basic PPF terms and concepts as well as the relation between PPF and BOPF-implemented BOs.
8	Enhancing further Objects, Features and Functions	New	This new main section is intended to cover cross topics and specific enhancements
8.1	Gantt Chart for Planning Functionality	New	The section describes how to set up a Transportation Cockpit Layout and include the Gantt Chart there (new as per SAP TM 9.2), It provides some comments and hints how this Gantt Chart can be configured and enhanced (currently only draft)



## 2 Glossary

The following abbreviations will be used in this document:

English Term	English Abbrev.	German Term (if applicable)	German Abbrev. (if applicable)	Definition
Business Object Processing Framework	BOPF		BOPF	
User Interface	UI		UI	
Business Object	BO		BO	
Business Object Repository	BOR		BOR	
Business Application Development Interface	BAdI		BAdI	
Transportation Management	TM		TM	
Floor Plan Manager	FPM		FPM	
Floor Plan Manager BOPF Integration	FBI		FBI	
Transportation Management	TM		TM	
Transportation Charges Management	TCM		TCM	
Process Controller Framework	PCF		PCF	
User Interface Building Block	UIBB		UIBB	
Generic Interface Building Block	GUIBB		GUIBB	
Post Processing Framework	PPF		PPF	
Enterprise Service Repository	ESR		ESR	
Software Component	SWC		SWC	
Software Component Version	SWCV		SWCV	
Enhancement Software Component Version	EnSWCV		EnSWCV	
Web Dynpro Component Configuration	WDCC		WDCC	
Change Document Object	CDO		CDO	
Status & Action Management	SAM		SAM	
Data Dictionary	DDIC		DDIC	



### 3 BOPF - Overview and Architecture

SAP TM is based on a set of Frameworks that help to realize different aspects of the application. The Business Objects are modeled and implemented with the *Business Object Processing Framework (BOPF)*. The User Interface is based on *ABAP Web Dynpro* and is realized with the *Floor Plan Manager (FPM)* which supports modeling, implementing and configuring the User Interfaces. The *Floor Plan Manager BOPF Integration (FBI)* is used to connect the Backend with the User Interface. It provides the connection between the Business Objects in the backend with the corresponding User Interface realized with the *FPM*.

To utilize the enhancement capabilities of SAP TM, some general knowledge on these Frameworks is required. Besides these Frameworks, general knowledge on the following implementation and configuration technologies are prerequisite for creating enhancements:

- BAIs (Implementation)
- Process Controller Strategies (Configuration)
- Conditions (Configuration)
- Change Controller (Configuration / Implementation)
- Implicit Enhancements (Implementation)
- BOPF Enhancement Workbench (Configuration / Implementation, part of the BOPF Framework)

The mentioned frameworks and technologies shall be described in the following sections to provide a very basic insight on how they are involved in the SAP TM application and how they are used for creating enhancements. This document can for sure not cover all aspects. Therefore, links to more detailed information sources will be provided where appropriate.

#### 3.1 BOPF - Business Object Processing Framework

Business Objects are the basis of the SAP TM application. Each Business Object represents a type of a uniquely identifiable business entity, described by a structural model, an internal process model as well as one or more Service Interfaces. The business processes provided with SAP TM operate on these Business Objects. Examples for TM Business Objects are the Forwarding Order or the Freight Order.

*BOPF* controls the application business logic as well as the data retrieval of the buffer and persistency layer. The main design principles are a clear separation of the business logic and the buffering of data as well as a clear structuring of the business logic into small parts with a clear separation of changing and checking business logic. The *BOPF* approach for implementing business objects breaks down business logic into the following four concepts (described in more detail on the next pages):

- Actions
- Determinations
- Validations
- Queries

The reason for this breakdown is to avoid the mixing of the four types of functionality into one single entity. This improves the potential for reusing implementations and simplifies maintenance by reducing the complexity and dependencies, and thereby reducing the development effort.

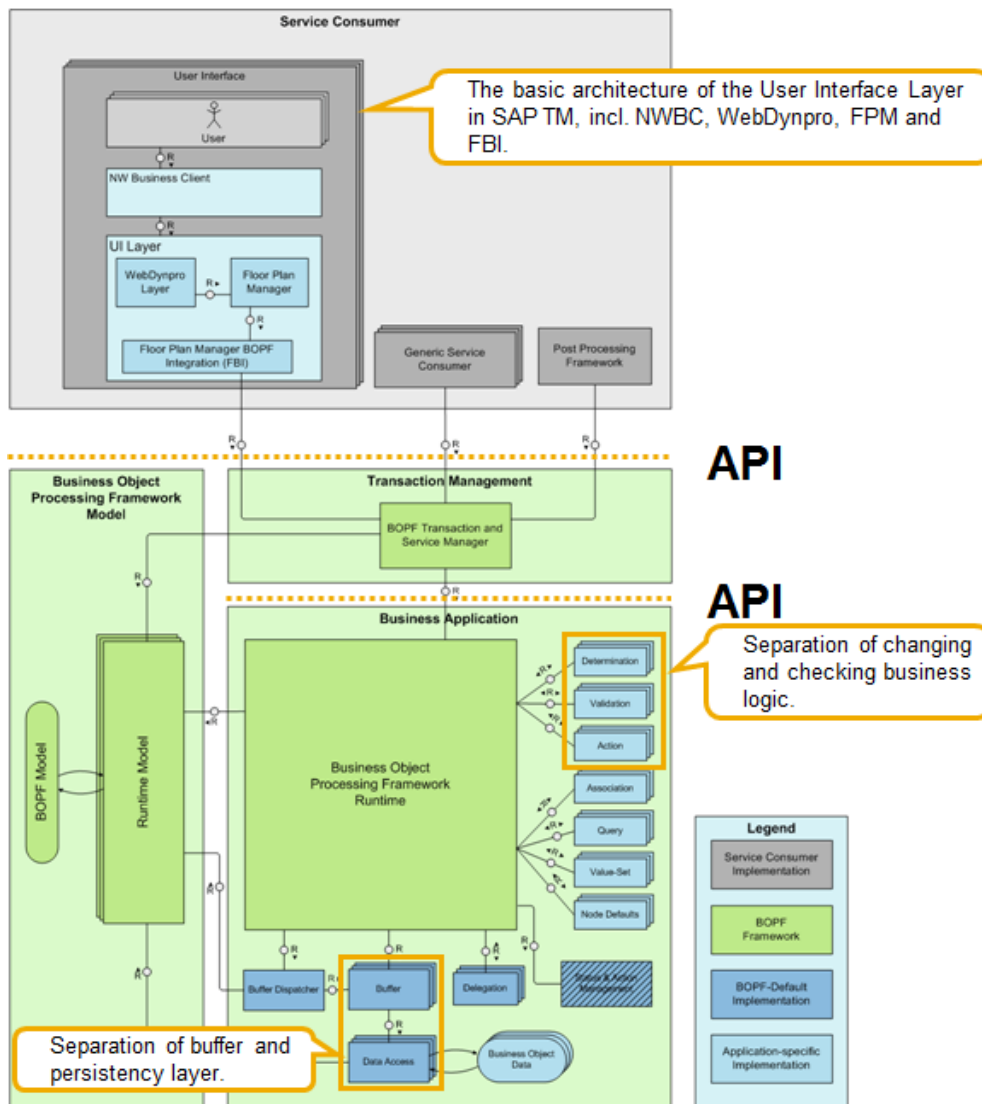


### 3.1.1 BOPF Architecture

The architecture of *BOPF* comprises two principal areas:

- **Business Application**, which is the heart of the application and provides an interface between the business data, the business logic and the end user.
- **BOPF Model**, where the runtime configuration parameters for each of the implemented business objects are located.

The Business Application includes specific entities that support the configuration and runtime operation of each business object, and offers access to the business object's data via Buffer Classes and Data Access Classes. Furthermore, the Business Application includes specific determinations, validations, actions and associations that dictate the specific behavior for each and every implemented business object.



Picture: The basic BOPF Architecture.

The Business Objects are accessed only via a defined API (Service Manager). Changing and checking Business Logic of a *BOPF* Business Object is clearly separated. There is no mixture of methods that change the business object with methods that have the purpose to check the business objects consistency.

Moreover, business logic and data buffering are clearly separated. The business logic is built on top of the Business Object and the buffer to behave independent of the way how data is buffered and where data is buffered. *BOPF* allows replacing buffer and data access classes for Business Objects. Both do not contain business Logic.

Data buffer and persistency are also clearly separated from each other as well as from the business logic. This allows establish individual buffer and persistency implementations, i.e. both are exchangeable (e.g. to achieve specific performance requirements).

Besides the basic *BOPF* architecture, the picture above also depicts the basic architecture of the Transportation Management User Interface.

### 3.1.2 Business Object Model

A Business Object is a representation of a type of uniquely identifiable business entities described by a structural model and an internal process model. Implemented business processes operate on business objects. Most important for the context of this document: A Business Object and its characteristics as well as its configuration settings can be enhanced. We'll later see how this is done. First, let's take a brief look at the parts a *BOPF* Business Object consists of. A *BOPF* Business Object model consists of the following entities:

#### **Nodes:**

A Node is a semantically related set of attributes of a business object. Nodes can be used to define and structure your business object. The attributes of a business object node are defined by dictionary data types.

Nodes can be hierarchically defined and related. Each business object has only one Root Node. Nodes are defined via compositions in a tree, but nodes can also be related in an arbitrary structure via associations that can be separate from the tree structure.

Business Object Representation nodes are placeholders for other business objects and the associations to these. They are only for visualization of the association to other business objects.

#### **Associations:**

An association is a direct, unidirectional, binary relationship between two business object nodes.

Associations can be used to relate two nodes in a well-defined direction. The association can be used to navigate from one node (source node) to the related node (target node). The associated nodes can be nodes within one business object or in different business objects (cross business object association).

Associations can have parameters to filter the result of the related nodes. They can only be defined between two nodes and in one defined direction. Moreover, they have a defined cardinality which gives information about the existence of an association and the number of associated nodes.

#### **Actions:**

An action is an element of a business object node that describes an operation performed on that node.

An action can be used to allow the external triggering of business logic (in contrast to a determination). When the action is performed, you must specify the key for the instances on which it is to be performed (if it is not a static action) and any input parameters that the action requires.

An action can only be performed with the number of instances that is configured in the cardinality of the action. It is performed for all instances if an error in the action validation has not occurred. If errors occur, then the behavior depends on the action settings.

### Determinations:

An element of a business object node that describes internal changing business logic on the business object. It can be used to trigger business logic based on internal changes (in contrast to an action). There are two types of determinations: Transient and Persistent. This categorization indicates whether a determination will alter persistent or only transient data. A determination is mostly used to compute data that can be derived from the values of other attributes. Examples:

- Products (for example, item amount = quantity × list price) and ratios.
- Totals of items (for example, invoice amount =  $\Sigma$  item amounts).
- Statuses.

The determined attribute and the determining attributes can belong to the same node (example 1) or to different nodes (example 2). There are also values that do not depend on any other value but still have to be determined automatically upon creation or modification of a node instance, for example, IDs, UUIDs, and GUIDs.

For each determination, it is necessary to specify which changes (such as create, update, delete or load) on which nodes will trigger the determination at a specific time. A determination is called at different points in time (determination time), depending on the model. The following determination times exist:

Execution Time	Use Case
After Loading	Dependent fields that are not saved (redundant) have to be recalculated.
Before Retrieve	Before Retrieve Determining contents of transient nodes before their first retrieval. After the first retrieval of a node instance determinations for this determination-time are not executed, as changes to data during retrieval are not allowed.
After Modify	Recalculation of fields that depend on changed fields. This is especially useful for derived fields that are of interest to the “outside world” and need to be updated immediately.
After Validation	This point in time can be used to modify data based on the outcome of consistency validations in the Determination & Validation cycle. A typical use case is to perform some follow-up actions depending on whether there were error messages in the consistency validations.
Before Save (Finalize)	Determine data that must not be determined prior to saving or for data that is not visible to the “outside world” (so it’s determination can be postponed until saving for performance reasons).
Before Save (Draw Numbers)	Determine data that must not be determined unless the transaction succeeds but may be used by other Business Objects. A typical use case for such very late changes is drawing numbers to assure gapless numbering.
During Save	Determine data that must not be determined unless the transaction succeeds. Determinations for this determination-time will be executed at most once in a LUW.
After Commit	Determine data after a transaction was successfully committed. A typical use case for this determination-time is starting asynchronous processes.
After Failed Save Attempt	Do cleanups after a try to save a transaction was rejected during the Finalize or Check before Save stages. A determination is only triggered if request nodes are assigned to it and instances of these request nodes are changed.



**Validations:**

A validation is an element of a business object node that describes some internal checking business logic on the business object.

Validations can be used to check if an action is allowed. Action validations can be assigned to object-specific actions and to the framework actions create, update, delete and save. They can be used to check if an action can be carried out. An action validation is carried out when an action is called before it is performed. If some validations fail, the action is not performed for the instances where the validation failed. Depending on the action settings, the action is also not performed.

A validation can be used to check the consistency of a business object. Consistency validations can be used to check the consistency of a business object. They can be assigned to the framework actions check of each node. Consistency validations are carried out when this action is called or automatically after a change is made if they are triggered via trigger nodes based on the changes. It is only triggered if some of the trigger nodes are assigned and instances of these trigger nodes are changed.

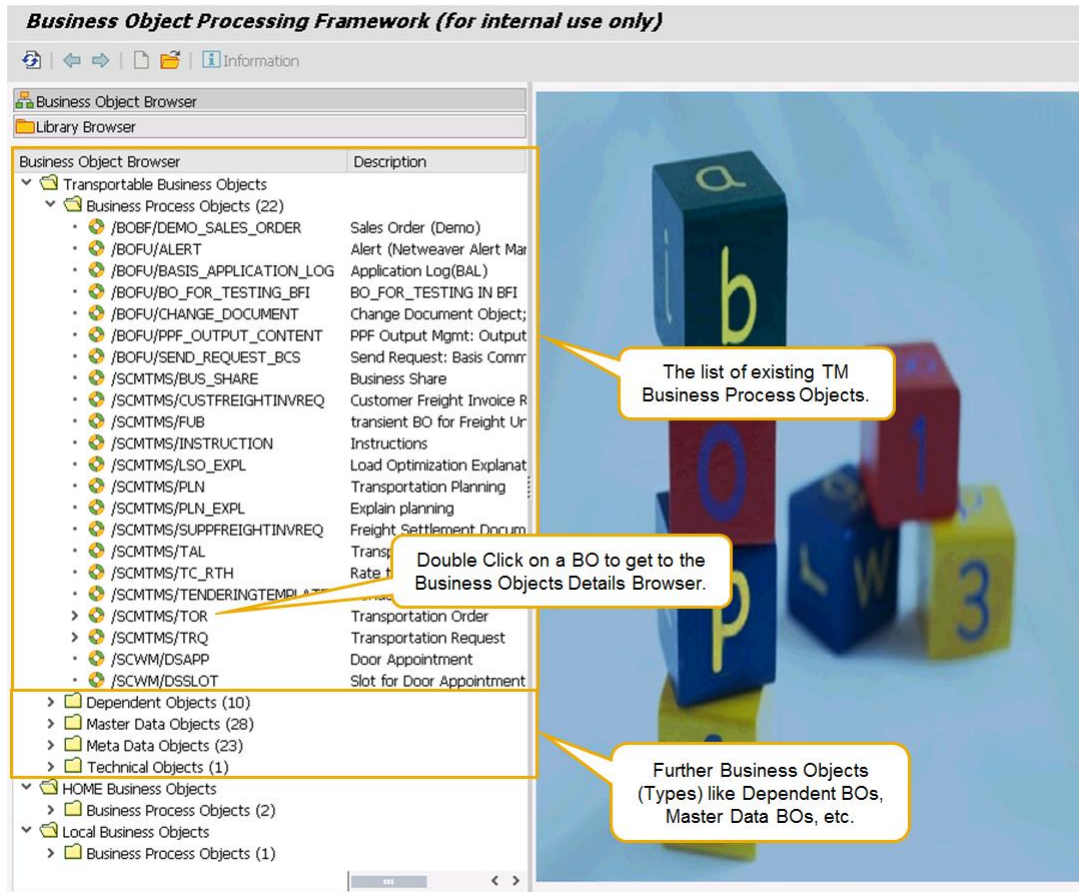
**Queries:**

Queries represent a defined set of attributes, such as search parameters, that return the queried IDs of the business object node instances.

A query allows you to perform searches on a business object. They provide the initial point of access to business objects. Each query has an associated parameter structure. The result of the query is a set of all the record IDs in a business object that match the query criteria.

### 3.1.3 BOPF Modeling Tool

The models of the TM business objects can be displayed with the BOPF Modeling Tool. It can be started via transaction `/BOBF/CONF_UI`. It allows browsing through the list of the business objects of the application. From here, you can navigate to the details of each business object to display its node structure and hierarchy, the configuration, the DDIC structures for each node, the node elements (e.g. Associations, Actions, Determinations, Validations and Queries), etc. Moreover, it allows navigating to the implementing ABAP classes of the business object.



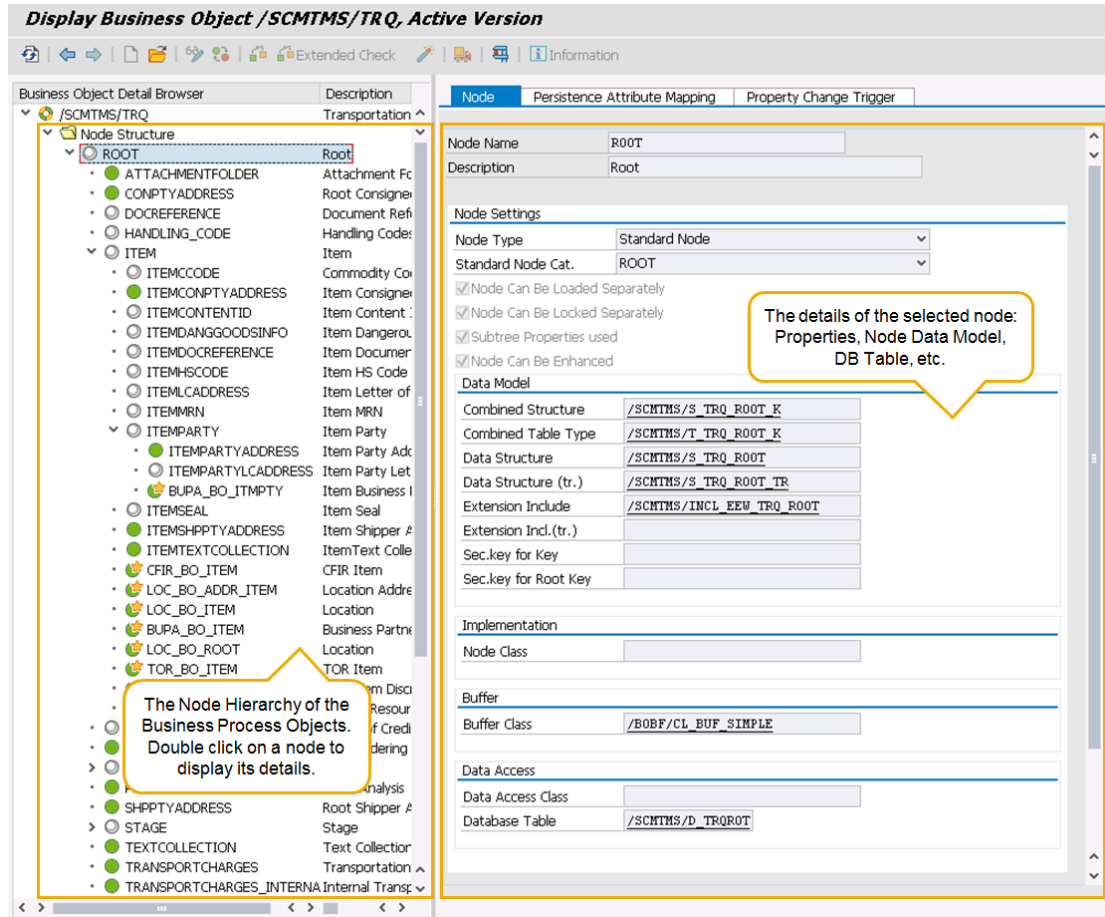
Picture: The Business Object Browser.

On the initial screen (Picture 2) the user can browse through the available TM business objects as well as four other object categories which are used in the context of TM. These are:

- Dependent Objects:**  
 Used in SAP TM for reusable parts of business objects that are not objects on their own, i.e. they only exist in the context of a business objects (the hosting object). Examples are address, attachment folder, text collection, and transportation charges.
- Master Data Objects:**  
 Most master data BOs call the SCM Basis Master Data Layer (MDL) via an adapter in a read-only way. The content of these master data objects is maintained via the standard transactions in SCM Basis. The master data distribution between SAP ERP and SAP TM follows the standard SCM middleware architecture of the SCM Core Interface (CIF). Within SAP TM, access to master data occurs via master data BOs only. Examples are Location, Business Partner, Material, etc.

- **Meta Data Objects:**

Examples are Freight Unit Building Rule, Planning and Selection Profiles, etc. which e.g. define data that is taken into consideration at runtime to define the required behavior of a business process (e.g. how shall Freight Units get built from the Forwarding Order data or which Freight Units shall be selected to be planned in the Transportation Cockpit of SAP TM).



Picture: The Business Object Detail Browser - Node Structure.

In the Business Object Detail Browser, you can navigate through the node hierarchy of the business object and display the node details. Besides other information, the node details show the data model of the node.

- **Combined Structure and Table Type:**

This DDIC structure includes the data structure of a node. In addition it includes a fixed BOPF DDIC structure which contains the node instance key (*KEY*), the key of the direct parent node instance (*PARENT\_KEY*) as well as the key of the related business object instance (*ROOT\_KEY*). The Combined Table Type has the Combined Structure as its line type.

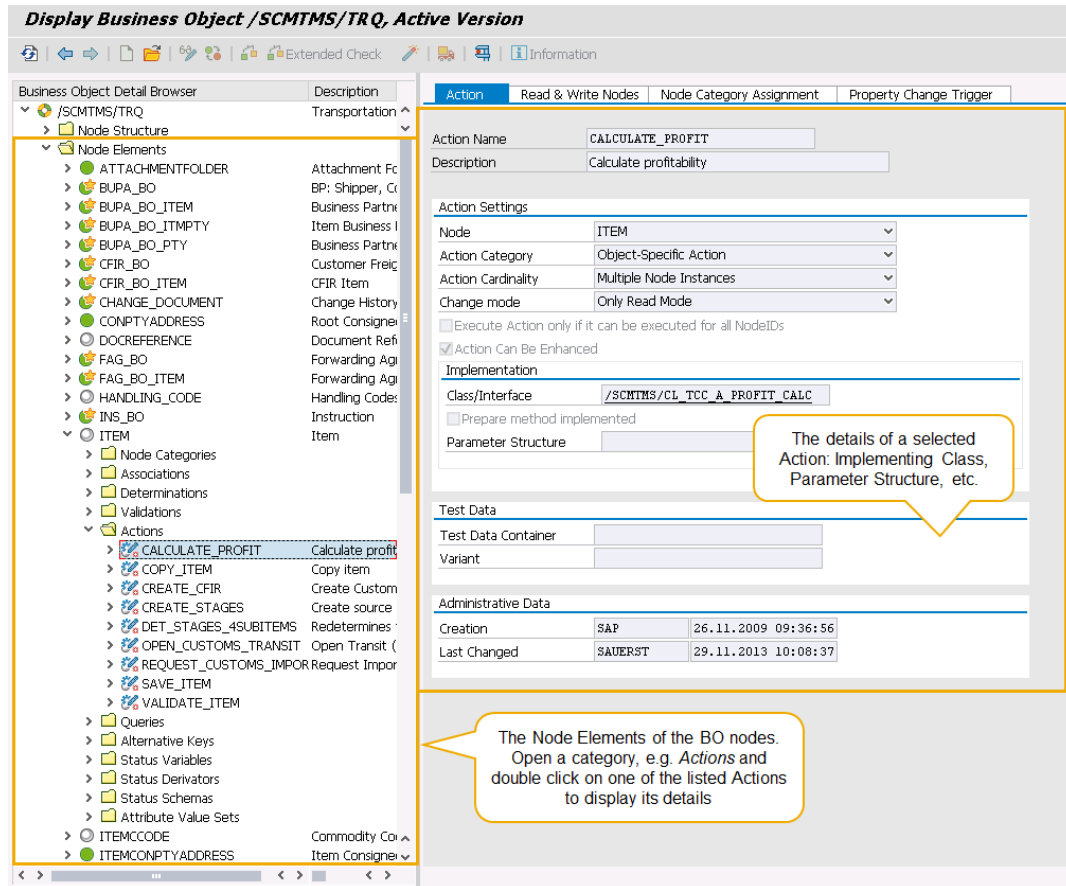
- **Data Structure:**

This DDIC structure contains the attributes of the node, representing the node data.

- **Data Structure (tr.):**

Contains the transient attributes of a node, i.e. attributes which do not get persisted but are only filled and used during runtime.

- Extension Include & Extension Include (tr.):**  
 Important for field extensions on a node is the Extension Include. With this include, all extension fields are added (via Append Structures) which are to be persisted. Extension fields which are only relevant at runtime and not relevant to be persisted are placed in the corresponding transient Extension Include.
- Database Table:**  
 Shows the database table where the persistent node information gets stored.



Picture: The Business Object Detail Browser - Node Elements.

When expanding the Node Elements, you can further navigate to a node and the elements assigned to it (e.g. Associations, Determinations, Validations, Actions and Queries as described in the previous sections). Moreover, the details for each of these elements can be displayed from here. For example, the details of an Action include a link to the implementing class of this Action and - if the Action has parameters - the corresponding parameter structure.

The details of the node elements like Actions, Validations, Determinations, etc. are the starting point to identify places in the coding where a specific functionality of interest is implemented. Within the implementing classes of the node elements, of course further classes and their methods are used to realize its functionality.

## 3.2 BOPF Consumer Implementation Basics

In this section, we give examples on how to implement *BOPF* consumers, i.e. how to use core services that allow creating, accessing and modifying business object instances. In section 2.3 we take a look at how to implement *BOPF* entities such as actions, determinations and validations (this can be also done with the *BOPF* Enhancement Workbench which is described in section 3.3).

### 3.2.1 Service Manager

A business object can be accessed via a so-called Service Manager. The following coding shows how to get an instance of the service manager for e.g. the business object Forwarding Order:

```
*&-----*
*& Report  ZREP_SRV_MGR
*&-----*
*& How to get a service manager instance and use it to access BOPF
*&-----*
REPORT  zrep_srv_mgr.

DATA: lo_srv_mgr TYPE REF TO /bobf/if_tra_service_manager.

* Get an instance of a service manager for e.g. BO TRQ
lo_srv_mgr = /bobf/cl_tra_serv_mgr_factory=>
              get_service_manager( /scmtms/if_trq_c=>sc_bo_key ).
```

Besides others, the service manager provides the following methods that can be used to access the corresponding business object that it was instantiated for:

Method	Description
QUERY	Search, execute a BO query.
RETRIEVE	Read data for a given set of node instance keys.
RETRIEVE_BY_ASSOCIATION	Read data via association.
DO_ACTION	Execute a given action of a BO node.
CONVERT_ALTERN_KEY	Convert an alternative key to the technical key.
MODIFY	Create, change and delete BO node instances.

The following coding examples and descriptions of the semantics of the corresponding method parameters illustrate the usage of the service manager methods to access *BOPF* business objects. We will add corresponding examples in a small demo report *ZREP\_BOPF\_DEMO\_1* step by step.

### 3.2.2 Query

The coding example shows how to call a *BOPF* query. To start a query, method *QUERY* of the service manager instance is used:

```
*&-----*
*& Report  ZREP_BOPF_DEMO_1
*&-----*
*& How to get a service manager instance and use it to access BOPF
*&-----*
REPORT  zrep_bopf_demo_1.

FIELD-SYMBOLS: <ls_root> TYPE /scmtms/s_trq_root_k,
                <ls_item> TYPE /scmtms/s_trq_item_k,
                <ls_link> TYPE /bobf/s_frw_key_link,
                <ls_loc>  TYPE /scmtms/s_bo_loc_root_k,
```

```

<ls_txc>  TYPE /bobf/s_txc_con_k,
<ls_msg>  TYPE /bobf/s_frw_message_k.

DATA: lo_srv_trq      TYPE REF TO /bobf/if_tra_service_manager,
      ls_selpar      TYPE /bobf/s_frw_query_selparam,
      lt_selpar      TYPE /bobf/t_frw_query_selparam,
      lo_message     TYPE REF TO /bobf/if_frw_message,
      ls_query_inf   TYPE /bobf/s_frw_query_info,
      lt_key         TYPE /bobf/t_frw_key,
      lt_root        TYPE /scmtms/t_trq_root_k,
      lt_failed_key  TYPE /bobf/t_frw_key,
      lt_item        TYPE /scmtms/t_trq_item_k,
      lt_link        TYPE /bobf/t_frw_key_link,
      lt_item_key    TYPE /bobf/t_frw_key,
      lt_target_key  TYPE /bobf/t_frw_key,
      lt_loc_root    TYPE /scmtms/t_bo_loc_root_k,
      lv_text_assoc_key TYPE /bobf/conf_key,
      lt_link_txctext TYPE /bobf/t_frw_key_link,
      lt_txc_text_key TYPE /bobf/t_frw_key,
      lv_text_node_key TYPE /bobf/conf_key,
      lv_content_node_key TYPE /bobf/conf_key,
      lv_content_assoc_key TYPE /bobf/conf_key,
      lt_txc_content TYPE /bobf/t_txc_con_k,
      lo_change      TYPE REF TO /bobf/if_tra_change,
      lr_action_param TYPE REF TO /scmtms/s_trq_a_confirm,
      lt_msg         TYPE /bobf/t_frw_message_k,
      lv_str         TYPE string,
      lo_msg         TYPE REF TO /bobf/cm_frw,
      lt_trq_id      TYPE /scmtms/t_trq_id,
      lt_trq_root_key TYPE /bobf/t_frw_key.

```

```

* Get an instance of a service manager for e.g. BO TRQ
lo_srv_trq = /bobf/cl_tra_serv_mgr_factory=>get_service_manager(
              /scmtms/if_trq_c=>sc_bo_key ).

```

BREAK-POINT.

```

* Set an example query parameter
ls_selpar-attribute_name = /scmtms/if_trq_c=>sc_query_attribute-root-
query_by_attributes-created_by.
ls_selpar-option        = 'EQ'.
ls_selpar-sign          = 'I'.
ls_selpar-low           = 'POLCH'.
APPEND ls_selpar TO lt_selpar.

```

BREAK-POINT.

```

* Use method QUERY of the service manager to start the query
lo_srv_trq->query(
  EXPORTING
    iv_query_key          = /scmtms/if_trq_c=>sc_query-root-
                           query_by_attributes
    it_selection_parameters = lt_selpar
  IMPORTING
    eo_message            = lo_message
    es_query_info         = ls_query_inf
    et_key                = lt_key ).

```

BREAK-POINT.



### 3.2.3 Retrieve

The coding example shows how to retrieve the data for the Root node keys that were found by the query in 3.2.2. Method *RETRIEVE* of the service manager instance is used:

```
...
BREAK-POINT.

* Use method RRETRIEVE to retrieve ROOT data
lo_srv_trq->retrieve(
  EXPORTING
    iv_node_key      = /scmtms/if_trq_c=>sc_node-root
    it_key            = lt_key
    iv_edit_mode      = /bobf/if_conf_c=>sc_edit_read_only
  IMPORTING
    eo_message        = lo_message
    et_data            = lt_root
    et_failed_key      = lt_failed_key ).

BREAK-POINT.
```

### 3.2.4 Retrieve By Association (Standard)

The coding example shows how to retrieve the data for the Item node keys that were found by the query in 3.2.2. Method *RETRIEVE\_BY\_ASSOCIATION* of the service manager instance is used with the composition association from Root to Item:

```
...
BREAK-POINT.

* Use method Retrieve by Association to retrieve ITEM node data
lo_srv_trq->retrieve_by_association(
  EXPORTING
    iv_node_key      = /scmtms/if_trq_c=>sc_node-root
    it_key            = lt_key
    iv_association    = /scmtms/if_trq_c=>sc_association-
root-
item
  iv_fill_data        = abap_true
  iv_edit_mode        = /bobf/if_conf_c=>sc_edit_read_only
  IMPORTING
    eo_message        = lo_message
    et_data            = lt_item
    et_key_link        = lt_link
    et_target_key      = lt_item_key
    et_failed_key      = lt_failed_key ).

BREAK-POINT.
```

### 3.2.5 Retrieve By Association (XBO)

The coding example shows how to retrieve the data for the Locations stored in the items whose Item node keys were retrieved in 3.2.4.

Again, method *RETRIEVE\_BY\_ASSOCIATION* of the service manager instance is used with the Cross BO association (*XBO*) from Item to Source Location Root that is defined on the item node of BO TRQ:

```

...
BREAK-POINT.

* Following XBO Association ITEM -> Location
lo_srv_trq->retrieve_by_association(
  EXPORTING
    iv_node_key      = /scmtms/if_trq_c=>sc_node-item
    it_key            = lt_item_key
    iv_association    = /scmtms/if_trq_c=>sc_association-item-
                      srcloc_root
    iv_fill_data      = abap_true
  IMPORTING
    eo_message        = lo_message
    et_data            = lt_loc_root
    et_key_link        = lt_link ).

BREAK-POINT.

```

### 3.2.6 Retrieve By Association (Dependent Objects)

The coding example shows how to retrieve the data from the dependent object TextCollection assigned to the Root node of BO TRQ. Besides method *RETRIEVE\_BY\_ASSOCIATION* of the service manager instance, this requires calling helper method *GET\_DO\_KEYS\_4\_RBA* of class */SCMTMS/CL\_COMMON\_HELPER* to map the TextCollection Meta Data node keys into TRQ runtime node keys:

```

...
BREAK-POINT.
* Retrieve by Association (To Dependent Object Nodes)
* Do RbA to ROOT TEXT Collection TEXT CONTENT node
* Get Text Collection ROOT keys
lo_srv_trq->retrieve_by_association(
  EXPORTING
    iv_node_key      = /scmtms/if_trq_c=>sc_node-root
    it_key            = lt_key
    iv_association    = /scmtms/if_trq_c=>sc_association-root-
                      textcollection
  IMPORTING
    eo_message        = lo_message
    et_key_link        = lt_link
    et_target_key      = lt_target_key ).

* Map TXC Meta model node keys into TRQ runtime node keys
* --> for all subnodes of DO ROOT we have to use this helper
* method to get the correct runtime node keys of the DO nodes

/scmtms/cl_common_helper=>get_do_keys_4_rba(
  EXPORTING
    iv_host_bo_key    = /scmtms/if_trq_c=>sc_bo_key
    "Host BO DO Representation node (TRQ, node TEXTCOLLECTION)
    iv_host_do_node_key = /scmtms/if_trq_c=>sc_node-textcollection
    "not needed here because source node of association is the DO ROOT
    "node for which we can use the TRQ constant
    * iv_do_node_key    = DO Node
    "DO Meta Model Association Key
    iv_do_assoc_key    = /bobf/if_txc_c=>sc_association-root-text
  IMPORTING
    "DO Runtime Model Association Key
    ev_assoc_key        = lv_text_assoc_key ).

```

```

lo_srv_trq->retrieve_by_association(
  EXPORTING
    iv_node_key          = /scmtms/if_trq_c=>sc_node-
                          textcollection
    it_key               = lt_target_key
    "DO runtime model association key
    iv_association       = lv_text_assoc_key

  IMPORTING
    eo_message           = lo_message
    et_key_link          = lt_link_txctext
    et_target_key        = lt_txc_text_key ).

* Map TXC Meta model node keys into TRQ runtime node keys
/scmtms/cl_common_helper=>get_do_keys_4_rba(
  EXPORTING
    iv_host_bo_key       = /scmtms/if_trq_c=>sc_bo_key
    iv_host_do_node_key = /scmtms/if_trq_c=>sc_node-textcollection
    "DO Meta Model Source Node Key
    iv_do_node_key       = /bobf/if_txc_c=>sc_node-text
  IMPORTING
    "DO Runtime Model Node Key
    ev_node_key          = lv_text_node_key ).

/scmtms/cl_common_helper=>get_do_keys_4_rba(
  EXPORTING
    iv_host_bo_key       = /scmtms/if_trq_c=>sc_bo_key
    iv_host_do_node_key = /scmtms/if_trq_c=>sc_node-textcollection
    "DO Meta Model Target Node key
    iv_do_node_key       = /bobf/if_txc_c=>sc_node-text_content
    "DO Meta Model Association Key
    iv_do_assoc_key      = /bobf/if_txc_c=>sc_association-text-
                          text_content
  IMPORTING
    "DO Runtime Model Node Key
    ev_node_key          = lv_content_node_key

    "DO Runtime Model Association Key
    ev_assoc_key         = lv_content_assoc_key ).

lo_srv_trq->retrieve_by_association(
  EXPORTING
    "DO runtime model source node key
    iv_node_key          = lv_text_node_key
    it_key               = lt_txc_text_key
    "DO runtime model association key
    iv_association       = lv_content_assoc_key
    iv_fill_data         = abap_true
  IMPORTING
    eo_message           = lo_messagect
    et_data              = lt_txc_content ).

```

### 3.2.7 Do Action (Standard)

The coding example shows how to start an action for a given set of TRQ instances represented by the corresponding Root node keys. The action *CONFIRM* of the TRQ Root node is called:

```

...
BREAK-POINT.

* Calling action CONFIRM of the TRQ Root node
lo_srv_trq->do_action(
  EXPORTING
    iv_act_key      = /scmtms/if_trq_c=>sc_action-root-confirm
    it_key          = lt_key
  * is_parameters   = Action Parameters if available & required
  IMPORTING
    eo_change       = lo_change
    eo_message      = lo_message
    et_failed_key   = lt_failed_key ).

BREAK-POINT.

```

### 3.2.8 Do Action (Action Parameters)

Again, the coding example shows how to start an action for a given set of TRQ instances represented by the corresponding Root node keys. In this example, the action *CONFIRM* of the TRQ Root node is called with some of the available action parameters:

```

...
BREAK-POINT.

* fill the action parameters
CREATE DATA lr_action_param.
* Carry out check
lr_action_param->no_check = abap_true.
lr_action_param->automatic = abap_false.

* Calling action CONFIRM of the TRQ Root node with parameters
lo_srv_trq->do_action(
  EXPORTING
    iv_act_key      = /scmtms/if_trq_c=>sc_action-root-confirm
    it_key          = lt_key
    is_parameters    = lr_action_param
  IMPORTING
    eo_change       = lo_change
    eo_message      = lo_message
    et_failed_key   = lt_failed_key ).

BREAK-POINT.

```

### 3.2.9 Convert Alternative Key

The coding example shows how to convert a list of TRQ IDs into the corresponding Root node keys. Method *CONVERT\_ALTERN\_KEY* of the service manager instance is used:

```

...
BREAK-POINT.

* Prepare a set of TRQ IDs
CLEAR lt_trq_id.
LOOP AT ls_root ASSIGNING <ls_root>.
  APPEND <ls_root>-trq_id TO lt_trq_id.
ENDLOOP.

```

```

* Convert IDs into BOPF keys
lo_srv_trq->convert_altern_key(
  EXPORTING
    iv_node_key    = /scmtms/if_trq_c=>sc_node-root
    iv_altkey_key  = /scmtms/if_trq_c=>sc_alternative_key-root-trq_id
    it_key         = lt_trq_id
  IMPORTING
    et_key         = lt_trq_root_key ).

BREAK-POINT.

```

### 3.2.10 Retrieve Property

The following coding example shows how to retrieve Property information of node elements. In this specific example the Service Manager method *RETRIEVE\_PROPERTY* is used to read the node attribute properties that are present at runtime. The concept of *BOPF* Node Element Properties is described in section 3.4.1 in more detail.

```

CALL METHOD lo_srv_trq->retrieve_property
  EXPORTING
    iv_node_key    = /scmtms/if_trq_c=>sc_node-root
    it_key         = lt_trq_root_key
    iv_node_attribute_property = abap_true
    it_node_attribute = lt_node_attribute
  IMPORTING
    eo_property    = lo_property
    eo_message     = lo_message.

BREAK-POINT.

```

### 3.2.11 Modify

While the coding examples of the previous sub sections demonstrated how to access *BOPF* business objects, we now take a look at how to create, update and delete BO (node) instances. For these purposes, method *MODIFY* of the service manager is used. To demonstrate different possibilities of the method, we create a second demo report *ZREP\_BOPF\_DEMO\_2* and add corresponding examples step by step.

```

*&-----*
*& Report  ZREP_BOPF_DEMO_2
*& How to get a service manager instance and use it to access BOPF.
*& How to create update and delete BO (node) instances
*& How to use a transaction manager to save changes.
*&-----*
REPORT  zrep_bopf_demo_2.

FIELD-SYMBOLS: <ls_root>      TYPE /scmtms/s_trq_root_k,
               <ls_trq_qdb> TYPE /scmtms/s_trq_q_result.

DATA:  lo_srv_trq      TYPE REF TO /bobf/if_tra_service_manager,
       lt_mod          TYPE /bobf/t_frw_modification,
       ls_mod          TYPE /bobf/s_frw_modification,
       lv_trq_new_key  TYPE /bobf/conf_key,
       lo_chg          TYPE REF TO /bobf/if_tra_change,
       lo_message      TYPE REF TO /bobf/if_frw_message,
       lo_msg_all      TYPE REF TO /bobf/if_frw_message,
       lo_tra          TYPE REF TO /bobf/if_tra_transaction_mgr,
       lv_rejected     TYPE abap_bool,
       lt_rej_bo_key   TYPE /bobf/t_frw_key2,
       ls_selpar       TYPE /bobf/s_frw_query_selparam,

```

```

lt_selpar          TYPE /bobf/t_frw_query_selparam,
lt_trq_qdb         TYPE /scmtms/t_trq_q_result.

* Get instance of service manager for TRQ
lo_srv_trq = /bobf/cl_tra_serv_mgr_factory=>get_service_manager(
              /scmtms/if_trq_c=>sc_bo_key ).

```

**Create:** As a first step, a new instance of business object TRQ (Forwarding Order) is created. First the modification table is set up to contain an entry for the creation (change mode is set to *CREATE*) of a new Root node instance. Then the data for the new Root node instance is assembled. Finally, the modification table is passed to method *MODIFY* of the service manager

```

...
BREAK-POINT.

*--- Creating a new TRQ instance ---*
ls_mod-node = /scmtms/if_trq_c=>sc_node-root.
ls_mod-key  = /bobf/cl_frw_factory=>get_new_key( ).
ls_mod-change_mode = /bobf/if_frw_c=>sc_modify_create.
CREATE DATA ls_mod-data TYPE /scmtms/s_trq_root_k.

ASSIGN ls_mod-data->* TO <ls_root>.
<ls_root>-trq_type = 'ZENH'.
APPEND ls_mod TO lt_mod.
lv_trq_new_key = ls_mod-key.

lo_srv_trq->modify(
  EXPORTING
    it_modification = lt_mod
  IMPORTING
    eo_change       = lo_chg
    eo_message      = lo_message ).

```

The next step shows how to instantiate a transaction manager which is used for persisting changes to the database. For this purpose, the *SAVE* method of the transaction manager is called.

```

...
BREAK-POINT.

* Save transaction to get data persisted (NO COMMIT WORK!)
lo_tra = /bobf/cl_tra_trans_mgr_factory=>get_transaction_manager( ).

* Call the SAVE method of the transaction manager
lo_tra->save(
  IMPORTING
    ev_rejected          = lv_rejected
    eo_change            = lo_chg
    eo_message           = lo_message
    et_rejecting_bo_key  = lt_rej_bo_key ).

```

**Update:** Now the newly created instance of the TRQ Root node is update with some additional data. The example code shows how to prepare the modification table for an update (the change mode will be set to *UPDATE*). Again, the modification table is then passed to method *MODIFY* of the service manager (and in this example the update is directly persisted by using the *SAVE* method of the transaction manager).



```

...
BREAK-POINT.

*--- Update the new instance with a Shipper ID ---*
CLEAR lt_mod.
ls_mod-node = /scmtms/if_trq_c=>sc_node-root.
ls_mod-key = lv_trq_new_key.
ls_mod-change_mode = /bobf/if_frw_c=>sc_modify_update.
CREATE DATA ls_mod-data TYPE /scmtms/s_trq_root_k.

ASSIGN ls_mod-data->* TO <ls_root>.
<ls_root>-shipper_id = 'B00_CAR002'.
APPEND /scmtms/if_trq_c=>sc_node_attribute-root-shipper_id
      TO ls_mod-changed_fields.
APPEND ls_mod TO lt_mod.

lo_srv_trq->modify(
  EXPORTING
    it_modification = lt_mod
  IMPORTING
    eo_change       = lo_chg
    eo_message      = lo_message ).

BREAK-POINT.

lo_tra->save(
  IMPORTING
    ev_rejected      = lv_rejected
    eo_change         = lo_chg
    eo_message        = lo_message
    et_rejecting_bo_key = lt_rej_bo_key ).

```

**Delete:** In the last step of the example report, we use a query to find TRQ instances that then will be deleted. Again, the modification table will be prepared for deleting a given TRQ instance (the change mode will be set to *DELETE*). Just like in the first steps, the modification table is then passed to method *MODIFY* of the service manager (and in this example the delete is directly persisted by using the *SAVE* method of the transaction manager).

```

...
BREAK-POINT.

* set an example query parameter
ls_selpar-attribute_name = /scmtms/if_trq_c=>sc_query_attribute-root-
query_by_attributes-trq_type.
ls_selpar-option         = 'EQ'.
ls_selpar-sign           = 'I'.
ls_selpar-low            = 'ZENH'.
APPEND ls_selpar TO lt_selpar.

* find a TRQ instance to be deleted
lo_srv_trq->query(
  EXPORTING
    iv_query_key       = /scmtms/if_trq_c=>sc_query-root-
qdb_query_by_attributes
    it_selection_parameters = lt_selpar
    iv_fill_data        = abap_true
  IMPORTING
    eo_message         = lo_message
    et_data            = lt_trq_qdb ).

BREAK-POINT.

```

```

* Delete 1st found instance
READ TABLE lt_trq_qdb ASSIGNING <ls_trq_qdb> INDEX 1.
CLEAR lt_mod.
ls_mod-node = /scmtms/if_trq_c=>sc_node-root.
ls_mod-key  = <ls_trq_qdb>-db_key.
ls_mod-change_mode = /bobf/if_frw_c=>sc_modify_delete.
APPEND ls_mod TO lt_mod.

lo_srv_trq->modify(
  EXPORTING
    it_modification = lt_mod
  IMPORTING
    eo_change       = lo_chg
    eo_message      = lo_message ).

* Call the SAVE method of the transaction manager
lo_tra->save(
  IMPORTING
    ev_rejected      = lv_rejected
    eo_change         = lo_chg
    eo_message        = lo_message
    et_rejecting_bo_key = lt_rej_bo_key ).

```

Note: When deleting the Root node of a business object instance, the *BOPF* Framework makes sure that all corresponding sub nodes of the business object instance will be also deleted, i.e. the complete business object instance will be deleted.

## 3.3 BOPF Enhancement Workbench

In the following section, we focus on detailed step-by-step descriptions on how to create enhancements using the *BOPF* Enhancement Workbench.

### 3.3.1 Overview

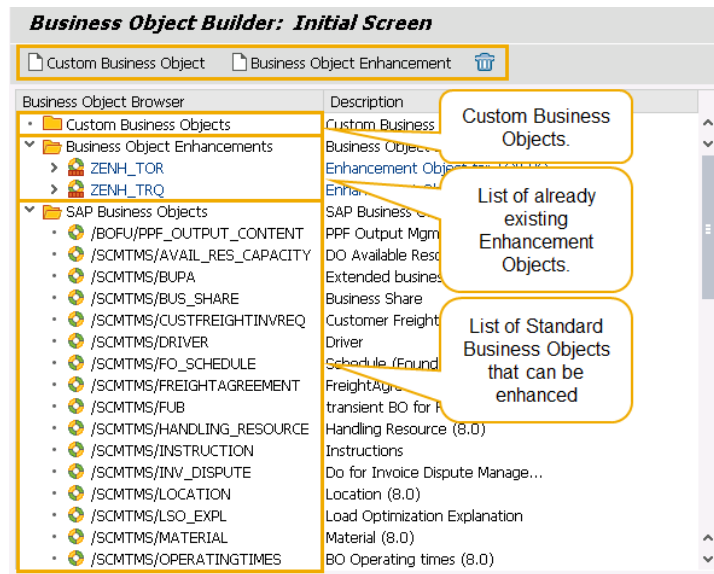
Since TM 8.0, the *BOPF* Enhancement Workbench is available to enhance the standard TM BOBF Business Objects. It can be used to create, change or delete enhancements of the standard TM BOBF Business Objects. Such enhancements again can be enhanced with the same tool, i.e. nested enhancements are also possible. The *BOPF* Enhancement Workbench supports the following enhancements:

Create, change or delete additional

- Subnodes.
- Actions and action enhancements.
- Determinations.
- Consistency and action validations.
- Queries.
- Customer/Partner-specific Business Objects → New as per NW 7.31.

The *BOPF* Enhancement Workbench now allows creating new customer/partner-specific Business Objects. Customers and partners can create their very own Business Objects that they can freely implement according to their requirements. Still Standard TM BOBF Business Objects must have been declared to be extensible by SAP Development. Only such Business Objects can be enhanced. The same applies to a Business Object's entities like nodes, actions, etc. They can only be enhanced if SAP Development has declared them to be extensible. If customer/partner-specific Business Objects are not declared to be extensible, others like implementation partners or SAP development cannot enhance them, i.e. nested enhancements are not possible. If this is required, declare them as extensible too.

The BOPF Enhancement Workbench is started with transaction `/BOBF/CUST_UI`.

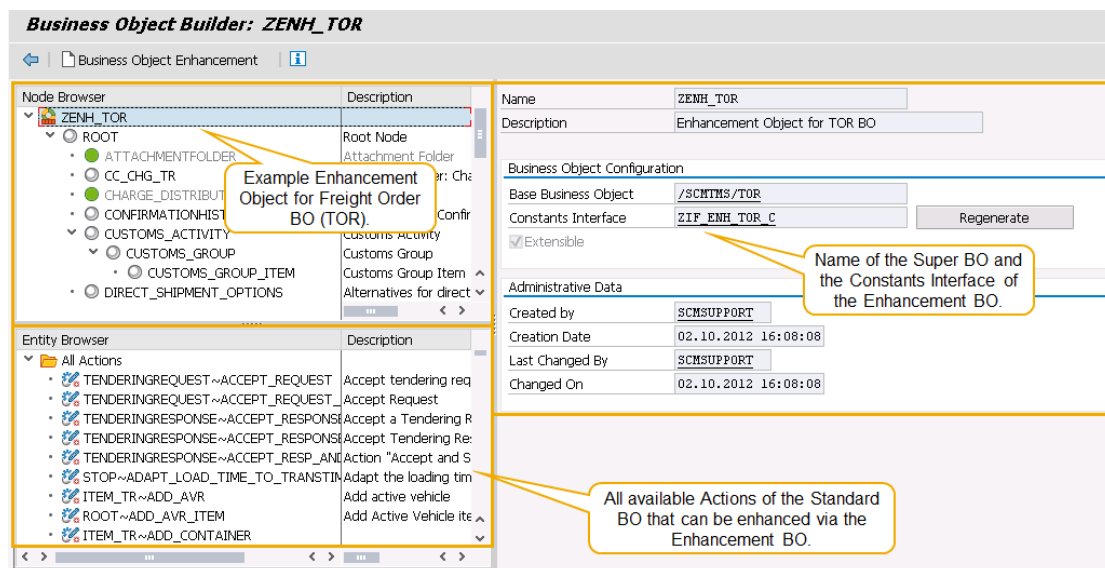


Picture: BOBF Enhancement Workbench Initial Screen.

On the initial screen, you can see the Business Objects of the Transportation Management Application that are allowed to be enhanced in general (see Business Objects in the picture above). Whenever you want to enhance one of the listed Standard Business Objects, the first step is to create a so-called Enhancement Object for this Business Object. For the Enhancement Object, the original Business Object represents the so called Super Business Object or base object.

Important to know is that this Enhancement Object does neither replace nor represent a copy of the standard Business Object. Instead it serves as a container for all enhancements that you add to the Business Object via the Enhancement Workbench. At runtime, still the standard Business Object functionality is being executed with the enhancements in addition.

A double click on one of the already existing Enhancement Objects will lead you to the corresponding details as shown in the following picture:



Picture: Details of an Enhancement Object.

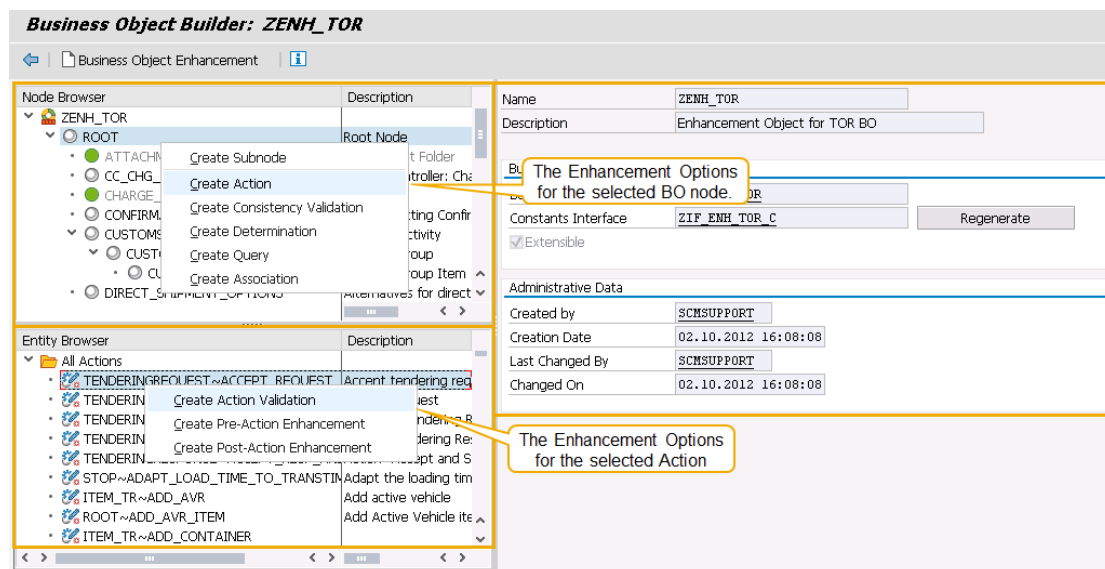
The specific example here shows an Enhancement Object for the Freight Order BO (TOR). The sections marked in the picture above show different entities of the Enhancement Object (double click on the very first line in the Node Browser).

The *Node Browser* section shows the node structure of the Enhancement Object which includes all nodes (i.e. the complete node hierarchy) of the super business object and all subnodes that might have been added to the business object via the Enhancement Workbench. All nodes which are not grayed out here are allowed to be enhanced. If a node is grayed out, it has been defined to be not extensible. This is especially the case for standard dependent objects like Address, Attachment Folder and Text Collection. On the other hand, dependent objects defined by Transportation Management itself can be enhanced, provided that development has enabled them correspondingly.

The *Entity Browser* section shows all actions available at the Enhancement Object. This includes all standard actions as well as actions that might have been added to the business object via the Enhancement Workbench (All actions are only shown when you have marked the Enhancement Object in this view. If you double click on one of the nodes, the Entity Browser will only contain those actions which belong to this specific node).

The details section on the right side shows the name of the Super Business Object associated with the selected Enhancement Object as well as the so called constants interface of the Enhancement Object. This interface will contain all constants which are generated for your enhancements.

A double click on one of the nodes (e.g. the Root Node) will show the following picture with details related to this specific node.



Picture: The node of an Enhancement Object with its details.

The picture above shows the possible enhancement options for the selected node (here: the Root Node of the Freight Order BO) in the *Node Browser*. Click the right mouse button to view the types of enhancements that can be created for the selected node.

In the *Entity Browser*, the available actions of the selected node are shown. You can select one of the listed actions and then click the right mouse button to see the types of enhancements that can be created for the selected action.

Moreover, in the details section, the node details (i.e. the data model) of the selected node is displayed, similar to those already mentioned in section 3.1.3 on the *BOBF* Modeling Tool. Here in the *BOBF* Enhancement Workbench you can see:

- **Persistent Structure:**  
This DDIC structure contains the attributes of the node, representing the node data.
- **Transient Structure:**  
Contains the transient attributes of a node, i.e. attributes which do not get persisted but are only filled and used during runtime.
- **Combined Structure & Combined Table Type:**  
This DDIC structure (table type) includes the data structure of a node. In addition it includes a fixed *BOPF* DDIC structure which contains the node instance key (*KEY*), the key of the direct parent node instance (*PARENT\_KEY*) as well as the key of the related business object instance (*ROOT\_KEY*).
- **Database Table:**  
Shows the database table where the persistent node information gets stored.
- **Extension Includes (persistent & transient):**  
Important for field extensions on a node is the Extension Include. With this include, all extension fields are added (via Append Structures) which are to be persisted. Extension fields which are only relevant at runtime and not relevant to be persisted are placed in the corresponding transient Extension Include.

By double click on the corresponding structures, tables or table types you can navigate to the details of these DDIC objects. Especially the Extension Includes are most important for field extensions. Here you add the customer / partner specific persistent and transient fields in corresponding append structures.

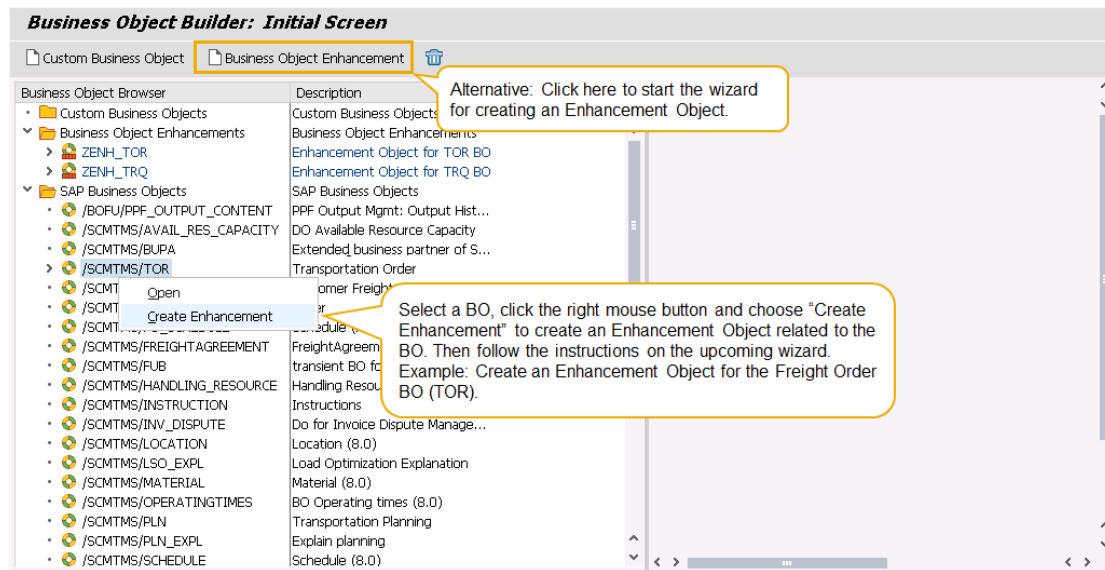
The procedure how to do this and how to create all other indicated enhancement options via the *BOPF* Enhancement Workbench will be described more detailed in the following sections.

### 3.3.2 First step: Creating an Enhancement Object

The first step to enhance a standard Transportation Management Business Object is to create a so called Enhancement Object for this Business Object. The Business Object associated with this Enhancement Object is also called the Super Business Object of an Enhancement Object.

It is important to know that this Enhancement Object does neither replace nor represent a copy of the standard Business Object. Instead it serves as a container for all enhancements that you add to the Business Object via the Enhancement Workbench. At runtime, still the standard Business Object functionality is being executed with the enhancements in addition. Moreover all coding corrections, adjustments of the BO Meta Model, etc. provided for the standard Business Objects and its entities will of course be present at any time.

- 1) Start the *BOPF* Enhancement Workbench (transaction */BOBF/CUST\_UI*) and select the Business Object to be enhanced. Let's assume we create an Enhancement Object for the Freight Order BO (its technical name is */SCMTMS/TOR*).
- 2) Click the right mouse button on the selected BO and select the option *Create Enhancement* in the upcoming popup menu.
- 3) A wizard will now guide you through the next steps for creating the Enhancement Object.



Picture: Creating an Enhancement Object.

- 4) On the first wizard screen click on button *Continue*.
- 5) On the next wizard screen you can see the name of the Super BO and the following list of fields ready for input:
  - **Enhancement Name:** The name of your enhancement.  
Example: *ENH\_TOR*
  - **Description:** A description of your enhancement.  
Example: *Enhancement Object for TOR BO*
  - **Namespace:** The namespace that your enhancement shall be associated with. It will be added at the beginning of the final technical name for your Enhancement Object.  
Example: *Z* (i.e. in this case the Customer Namespace).
  - **Prefix:** A prefix that will be added between the Namespace and the Enhancement Name in the final technical name of your enhancement. It is not a mandatory field and can be left empty if not required. In the following examples we will keep it just empty.
- 6) Click on button *Continue*. With the entries made before, the next wizard step will propose the name of your enhancement as well as the name of the constants interface that will be created for the new Enhancement Object. Example (with the given entries):

Technical Name : *ZENH\_TOR*  
 Constants Interface : *ZIF\_ENH\_TOR\_C*

On this screen you can also manually adjust the two fields.

- 7) Click on button *Continue* to get to the next wizard step. Here you define (yes/no) whether you allow your enhancement to be enhanced in further enhancements (i.e. nested enhancements are possible).
- 8) At any step of the wizard you can go back to all preceding steps again by clicking *Back* button. With this, adjusting the entered data is of course possible until you have completed the wizard. Click on button *Complete* to finally create your Enhancement Object with the entered specification.



The new Enhancement Object can now be used to create enhancements of the corresponding Super Business Object. The creation of the different types of enhancements on node and action level is described in the following sections.

### 3.3.3 General remarks on creating enhancements

Some general remarks that are valid for creating enhancements via the wizards provided by the BOPF Enhancement Workbench:

- **Implementing Classes:** Some types of enhancements require providing an implementing class. This class contains the business logic of the enhancement. In the wizards you define a class name and the system creates it automatically with the right interface assigned for the corresponding entity to be created after finishing the wizard. You must implement it of course manually.

The implementing class name should meet naming conventions. The wizard automatically suggests a class name that corresponds to the BOPF naming conventions. You can also provide already existing classes as implementing classes. They need to implement the corresponding BOPF Interface (e.g. `/BOBF/IF_FRW_ACTION` for actions or `/BOBF/IF_FRW_VALIDATION` for validations, etc.). The system does not overwrite the implementing class if it already exists when finalizing the wizard.

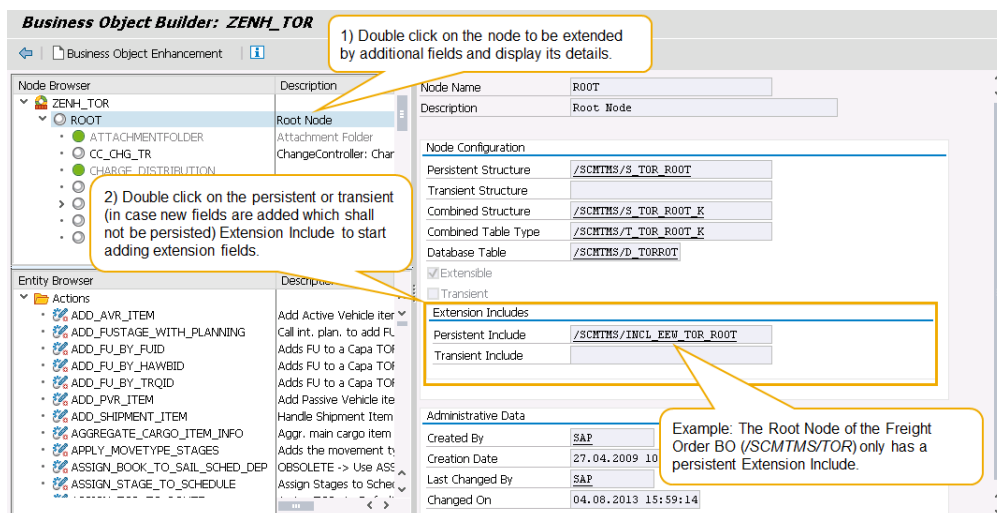
- The enhancement name should start with the namespace or prefix of the open enhancement (in our example this would be `"ZENH_"`). This ensures there is a clear separation between the entities of different enhancements (e.g. in case two or more different implementation partners want to separate their enhancements). The system automatically enters the value in the field for the enhancement name. You should add a meaningful enhancement name.
- When completing the wizard, further required objects will be generated automatically. The system adds the new enhancement to the enhancement object. It displays the new enhancement in the *Entity Browser* when you select the corresponding assigned node or action. As mentioned, in case an implementing class is required for the enhancement, it is generated and must be implemented manually afterwards. The constants interface of the enhancement object is regenerated and contains a unique constant identifying the new enhancement. In the enhancement constants interface you will only find constants for your enhancements. The constants for standard entities are located in the standard BO constants interface.
- Every enhancement created with the *BOPF Enhancement Workbench* can also be deleted again. For each create wizard a corresponding delete wizard is available which guides you through the relevant step and also checks the preconditions to be fulfilled for a deletion.
- Note that each Enhancement Object has its own Constants Interface. As you can see in the coding examples of section 3.2 the Node Elements of a BO are referred to via the BO's standard Constants Interface. Enhancement Node elements are not added to the standard Constants Interface but to the corresponding Enhancement Object Constants Interface. The latter one must be used to reference your Enhancement Node Elements in your coding.

### 3.3.4 Creating Field Extensions

Customers and Partners may require additional fields to be stored with the business objects, get them entered and displayed on the User Interface or get them transferred between external systems (e.g. ERP) and Transportation Management via corresponding services.

Creating field extensions on the Business Objects delivered with Transportation Management is the basis for these kinds of enhancements. Such field extensions can be created using the *BOPF* Enhancement Workbench.

- 1) Start the *BOPF* Enhancement Workbench (*/BOBF/CUST\_UI*) and select the Enhancement Object for the Business to be enhanced with additional fields. Let's assume we create a field extension for the Freight Order BO (Enhancement Object *ZENH\_TOR* created in section 3.3.2).
- 2) Double click on the node that shall be enhanced by additional fields (in our example the Root Node of the Freight Order BO). In the details of the node in the right area you can see the Extension Includes that will carry the additional fields. Field Extensions are always assigned to such an Extension Include.
  - Double click on the **Persistent Extension Include** to start adding fields that are to be persisted on the database.
  - Double click on the **Transient Extension Include** to start adding fields that are intended to be only used during runtime and do not get persisted on the database.

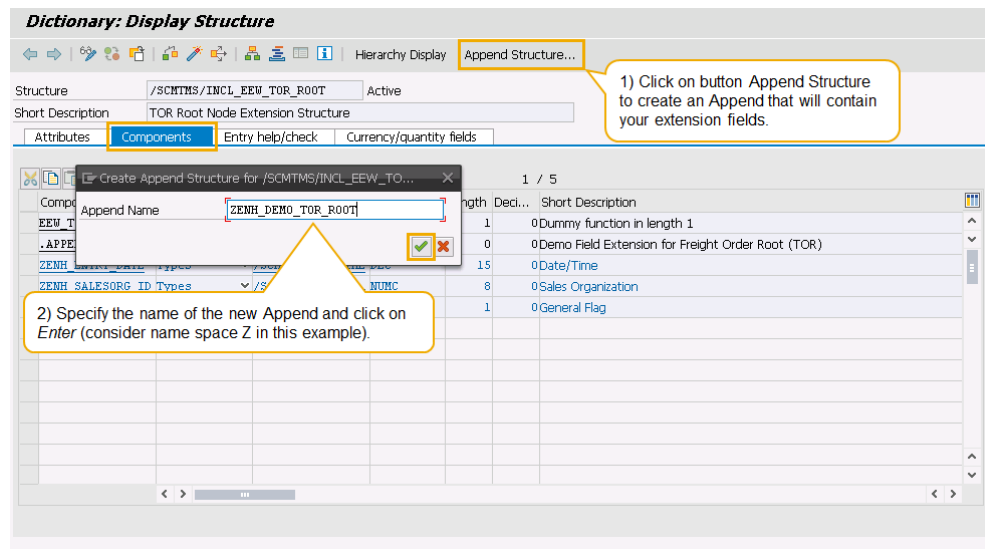


Picture: The Extension Includes of a BO node.

- 3) On the following screen you can see the DDIC Editor (analog to transaction SE11 for DDIC objects). Here click on the menu button *Append Structure...* to create a new Append for the chosen Extension Include.

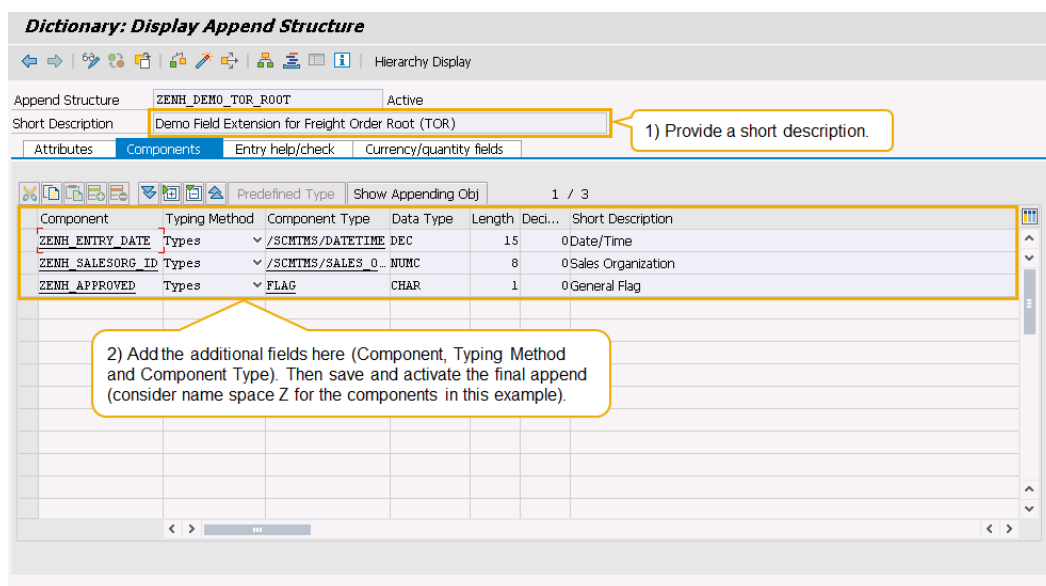
This append will contain your extension fields. Usually, you just need one such append to place all your extension fields. But it is also possible to create additional Appends for the Extension Include e.g. to separate extensions from different partners.

Example append: *ZENH\_DEMO\_TOR\_ROOT*



Picture: Creating an Append for the Extension Include.

- 4) Enter a short description for the Append and add the extension fields to be included in this Append (Component, Typing Method and Component Type). After the extension fields are correctly specified, save and activate the Append.



Picture: Specifying the extension fields in the new Append.

With the described four steps, the new extension fields are now part of the corresponding Node Structures, Table Types and the Database Table (provided that you have added the extension fields in the Persistent Extension Include). Both, transient and persistent extension fields are now ready to be used within further enhancements, e.g. in the business logic, the User Interface or in the context of services that send or receive corresponding information.

### 3.3.5 Creating Subnodes

The Enhancement Workbench allows extending a business object with additional nodes. Subnodes can be added via a corresponding wizard that guides you through the required steps.

- 1) Open the corresponding Enhancement Object and select the node that shall be extended with a new subnode. In the context menu of the node (click right mouse button) choose *Create Subnode* to start the wizard. Example:

Root Node of the Freight Order BO in the example Enhancement Object *ZENH\_TOR*.

- 2) The first step in the wizard is to specify the name for the new subnode and a description on the semantic and purpose of the new subnode. Example:

Node Name : *ZENH\_ROOT\_SUBNODE*  
Description : Demo Enh. TOR Root Subnode

Note: The name of the subnode must be unique in the business object and should start with the namespace of the used enhancement object. If no namespace has been entered, the node name must start with the prefix of the used enhancement (in our example this would be "*ZENH\_*"). This ensures that you have a clear separation between the nodes of different enhancements that belong to the same business object. The namespace (or prefix) value is automatically inserted in this field and must be completed with a meaningful node name.

- 3) In the second step you need to define whether this new subnode is itself extensible. Set the flag *Node is extensible* if you want to add additional enhancements to the new subnode (i.e. further subnodes, actions, determinations, etc.). If the flag is set, you can specify the names of a Persistent Extension Include and a Transient Extension Include to allow field extensions for the new subnode.

- (1) Enter the names of these includes. Example:

Persistent Extension Include : *ZENH\_INCL\_P\_TORSUBNODE*  
Transient Extension Include : *ZENH\_INCL\_T\_TORSUBNODE*

- (2) On the wizard screen double click on these structures to right away start creating the corresponding DDIC objects. You will be guided to the DDIC Editor where you can define the initial information for the Extension includes. Repeat the following steps for both Extension Includes:

- (3) Under menu path *Extras → Enhancement Category* choose enhancement category "*Can be enhanced (character-like or numeric)*".

- (4) Provide a short description.

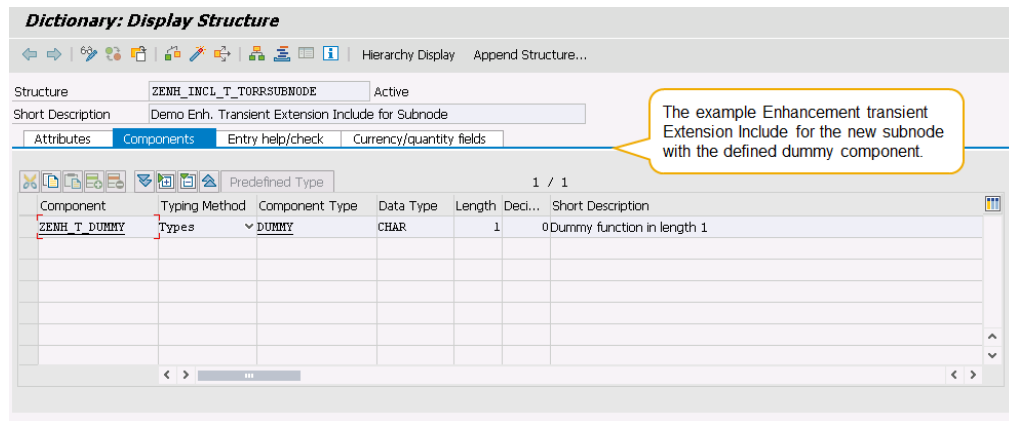
- (5) Create a dummy component (required for technical reasons). Example for the Persistent Extension Include:

Component	Typing Method	Component Type
ZENH_P_DUMMY	Types	DUMMY

Do the same for the Transient Extension Include (if you have defined one). Example for the Transient Extension Include:

Component	Typing Method	Component Type
ZENH_T_DUMMY	Types	DUMMY

- (6) Save and activate the corresponding Extension Include.



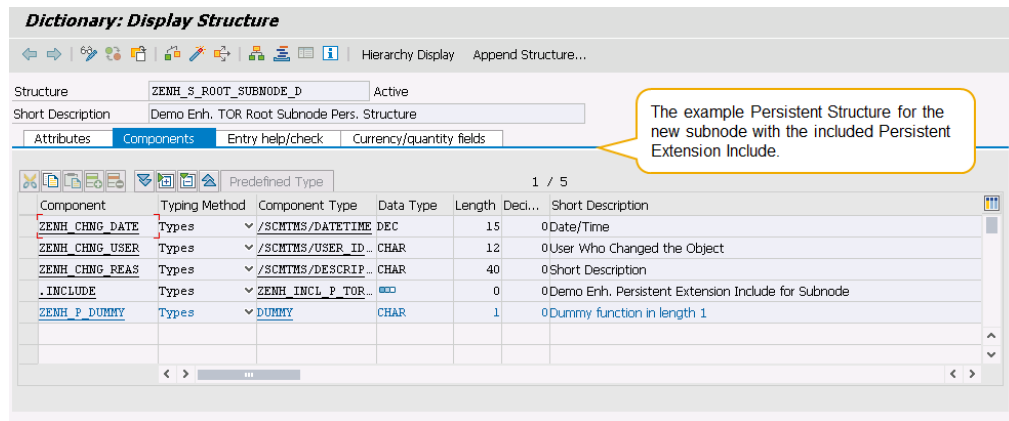
Picture: Example Enhancement Extension Include with dummy component.

- 4) The third step is to define and create the Persistent Structure and/or the Transient Structure for the subnode.

- (1) Enter the names for these structures. Example:

Persistent Structure : `ZENH_S_ROOT_SUBNODE_D`  
 Transient Structure : `ZENH_S_ROOT_SUBNODE_DT`

- (2) On the wizard screen double click on these structures to right away start creating the corresponding DDIC objects. You will be guided to the DDIC Editor where you can define the initial information for these structures.
    - (3) Under menu path *Extras* → *Enhancement Category* choose enhancement category “*Can be enhanced (character-like or numeric)*”.
    - (4) Provide a short description.
    - (5) In the Persistent Structure define the attributes that shall be part of the new subnode and get persisted on the database.
    - (6) At the end of the Persistent Structure include the Persistent Extension Include from step 3 to allow persistent field extensions for the new subnode.
    - (7) In the Transient Structure define the attributes that shall be part of the new subnode and will only be available at runtime (their content is created via Determinations at runtime).
    - (8) At the end of the Transient Structure include the Transient Extension Include from step 3 to allow transient field extensions for the new subnode.
    - (9) Save and activate the corresponding Extension Include.

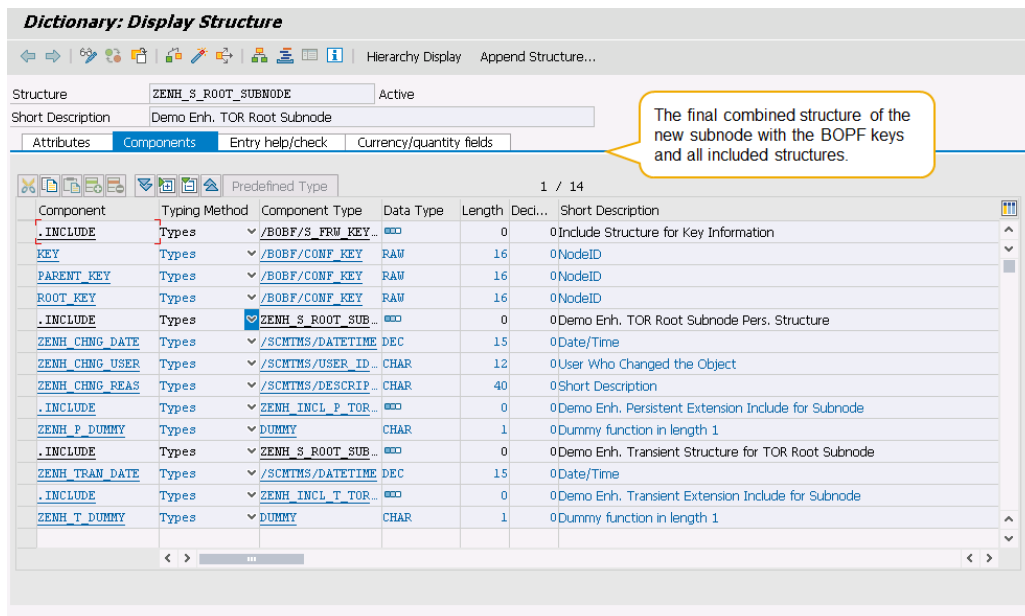


Picture: Example Persistent Structure.

- 5) In step four, the database types are defined. For this, the name of the Combined Structure, the Combined Table Type as well as the Database Table name is entered. Example:

Combined Structure : ZENH\_S\_ROOT\_SUBNODE  
 Combined Table Type : ZENH\_T\_ROOT\_SUBNODE  
 Database Table Name : ZENH\_D\_ROOT\_SUBN

The content of these three DDIC objects will be automatically generated by the system. The combined structure will contain the attributes of the persistent and transient structure from step 4 as well as BOBF-specific key attributes. Moreover, the combined structure will contain the Extension Includes with corresponding extension fields (if available). The combined table type and the database table will have the same structure like the combined structure.

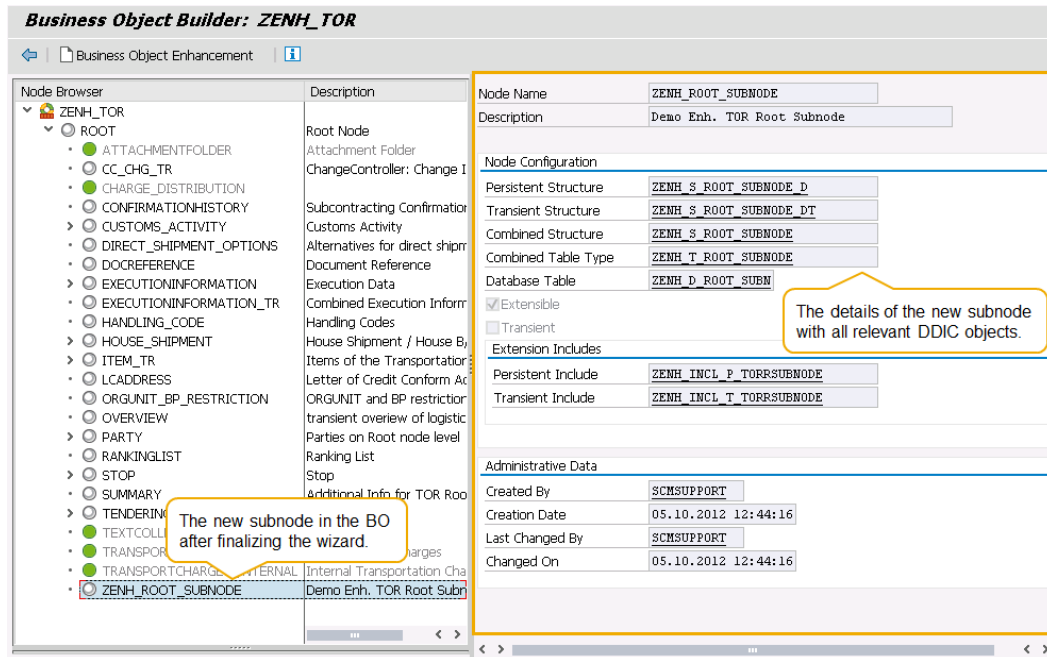


Picture: The final combined structure of the new subnode.

- 6) Click on button *Complete* to finalize the creation of the new subnode. With this step, further required objects for the subnode will be generated automatically. Afterwards, the subnode can be used like any other node of the enhanced business object. If you have declared it to be extensible, any enhancements for the new node are done the same way like for the standard nodes, i.e. you can add fields via field extensions, add actions, etc.



On completion of the subnode, also the constants interface of the enhancement is regenerated and contains a unique constant identifying the subnode (this constant is required for accessing the data of the node correctly).



Picture: The final new sub node.

### 3.3.6 Creating Actions

You can use an action to allow the explicit external triggering of business logic. Actions can be added to extensible standard nodes and new subnodes. The wizard for this task guides you through the following required steps.

- 1) Open the corresponding Enhancement Object and select the node that shall be extended with a new action. In the context menu of the node (click right mouse button) choose *Create Action* to start the wizard. Example:

Root Node of the Freight Order BO in the example Enhancement Object *ZENH\_TOR*.

- 2) The first step in the wizard is to specify the name for the new action and a description on the semantic and purpose of the new name. Example:

Action Name : *ZENH\_ROOT\_DEMO\_ACTION*

Description : Demo Enh. Action on TOR Root

- 3) In the second step you need to define the following information on the new action.

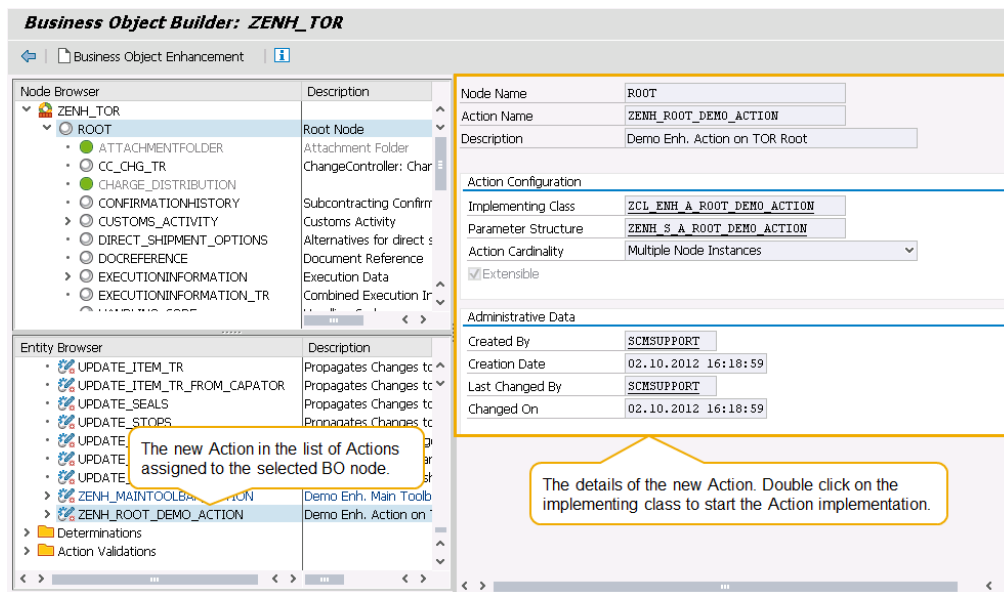
- (1) Implementing Class for example: *ZCL\_ENH\_A\_ROOT\_DEMO\_ACTION*.

The implementing class must implement interface */BOBF/IF\_FRW\_ACTION*.

- (2) Action Cardinality: Defines how many node instances the action can operate on during one action call. The following are the action cardinality types:

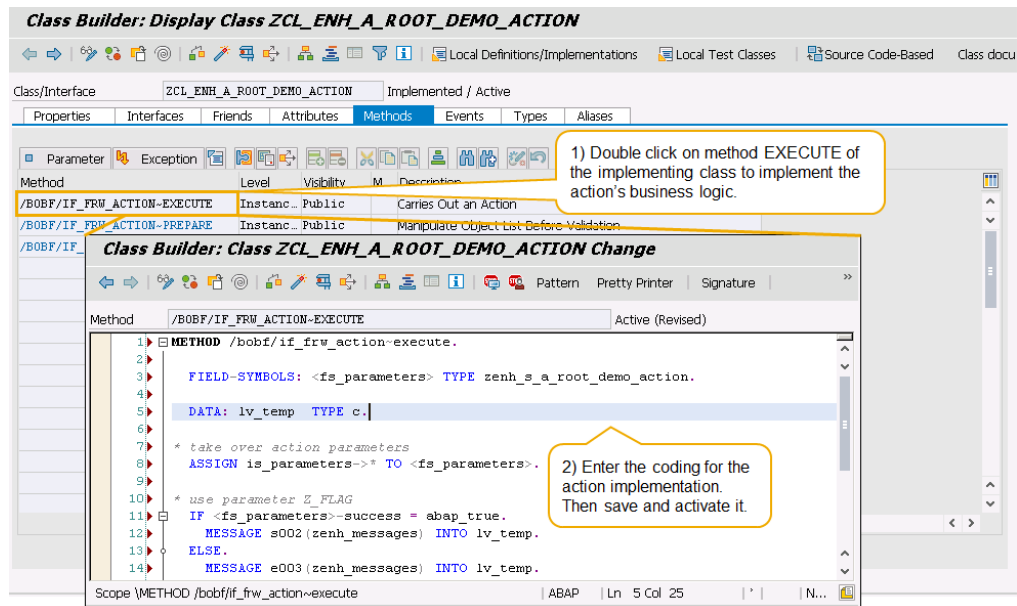
- Multiple Node Instances: Select if the action always operates on one or more node instances.
- Single Node Instance: Select if the action operates on exactly one single node instance for each call.

- Static Action (No Node Instances): Select if the action does not operate on any node instances.
- (3) Parameter Structure: Some actions need an additional importing parameter. Enter a name for the parameter structure and then start creating the structure by double-clicking the name. You actually get navigated to transaction *SE11* where you can simply define a DDIC structure (even a deep one) that represents the different required parameters for your action.
  - 4) On the next wizard screen you can specify whether the new action shall be extensible or not. Set the flag *Action can be enhanced* if you want to allow adding enhancements to the new action (i.e. adding Pre- and Post-Action Enhancements and Action Validations).
  - 5) Click on button *Complete* to finalize the creation of the new action.



Picture: The final new Action assigned to the respective node.

- 6) Finally, you need to implement your business logic in the Action's implementing class that you have specified in the previous steps. Double click on the implementing class in the action details to start the implementation.

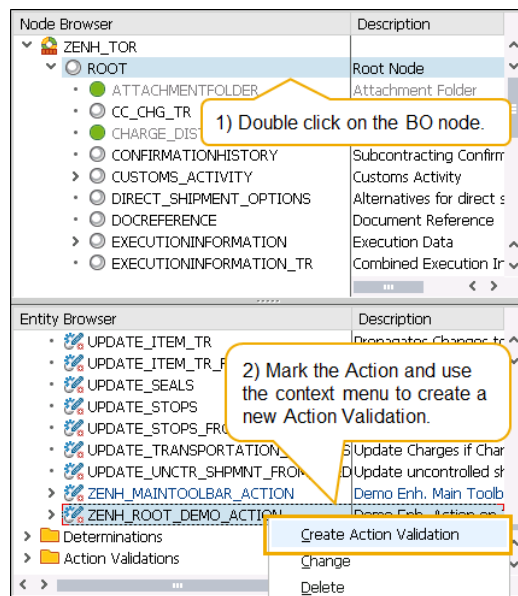


### 3.3.7 Creating Action Validations

An action validation is referred to a certain action. It contains checking logic which is automatically executed before the action is processed. It can be used to check if an action can be carried out. An action validation is carried out when an action is called, and before it is performed. If some validations fail, the action is not performed for the instances where the validation failed.

The BOBF Enhancement Workbench provides a corresponding wizard to create new action validations for extensible standard actions and enhancement actions.

- 1) Open the corresponding Enhancement Object and navigate to the Action that shall be extended with a new Action Validation. In the context menu of the Action (click right mouse button) choose *Create Action Validation* to start the wizard. Example:



Example: Action `ZENH_ROOT_DEMO_ACTION` created in section 3.3.6 (Root Node of the Freight Order BO in the example Enhancement Object `ZENH_TOR`).

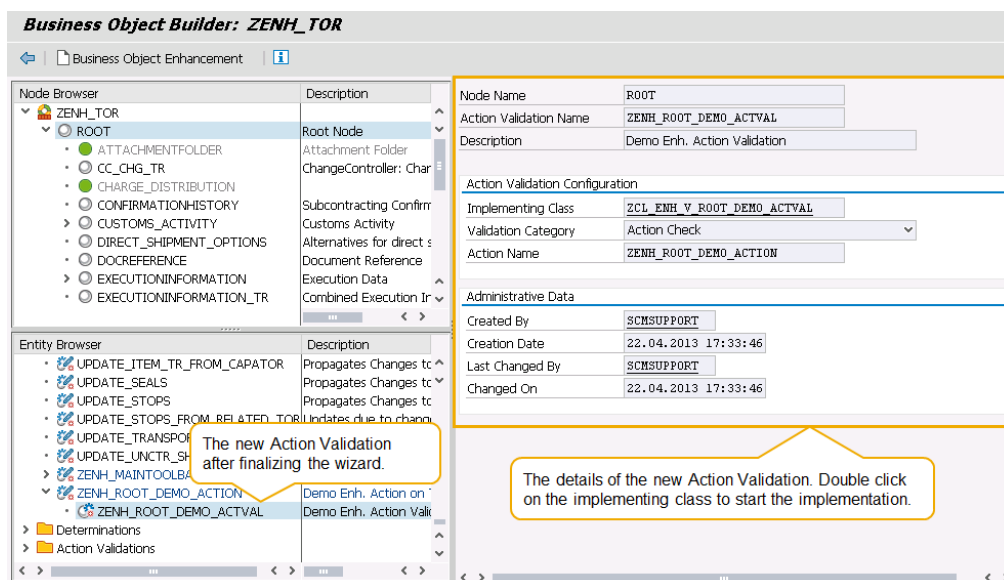
- 2) The first step in the wizard is to specify the name for the new action validation and a description on the semantic and purpose of this new entity. Example:

Validation Name : `ZENH_ROOT_DEMO_ACTVAL`  
Description : Demo Enh. Action Validation

- 3) In the second step you need to define the implementing class of the new action validation. Example: `ZCL_ENH_V_ROOT_DEMO_ACTVAL`.

The implementing class must implement interface `/BOBF/IF_FRW_VALIDATION`.

- 4) Click on button *Complete* to finalize the creation of the new action validation.
- 5) Finally, you need to implement your business logic in the action validation's implementing class that you have specified in the previous steps. Double click on the implementing class in the action validation details to start the implementation.



Picture: The final new Action Validation assigned to the corresponding action.

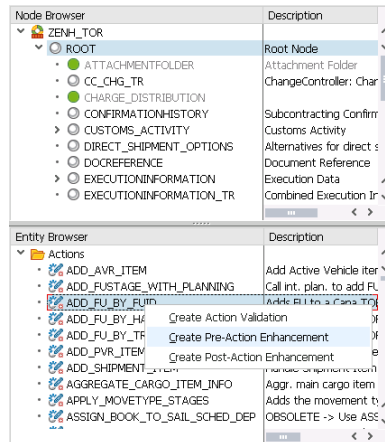
### 3.3.8 Creating Pre- and Post-Action Enhancements

A pre- or post-action enhancement can be used to extend the functionality of a certain action that is located in the base object (i.e. they are created for extensible standard actions only).

- A pre action enhancement is automatically executed by the BOPF framework, before a certain action of the base object is performed.
- A post action enhancement is automatically executed by the BOPF framework after a certain action of the base business object was performed.

If an importing parameter structure is maintained on the base action, this parameter is also handed over to the pre or post action enhancement and can be used in the implementation of the corresponding business logic. The BOPF Enhancement Workbench provides corresponding wizards for creating pre and post action enhancements. As an example, we describe the creation of a pre action enhancement (the procedure works analog for post action enhancements).

- 1) Open the corresponding Enhancement Object and navigate to the standard Action that shall be extended with a new Pre-Action Enhancement. In the context menu of the Action (click right mouse button) choose *Create Pre-Action Enhancement* to start the wizard.



Picture: Context menu of a Standard Action.

Example: Action *ASSIGN\_TSP* assigned to the Freight Order BO Root Node (in the example Enhancement Object *ZENH\_TOR*).

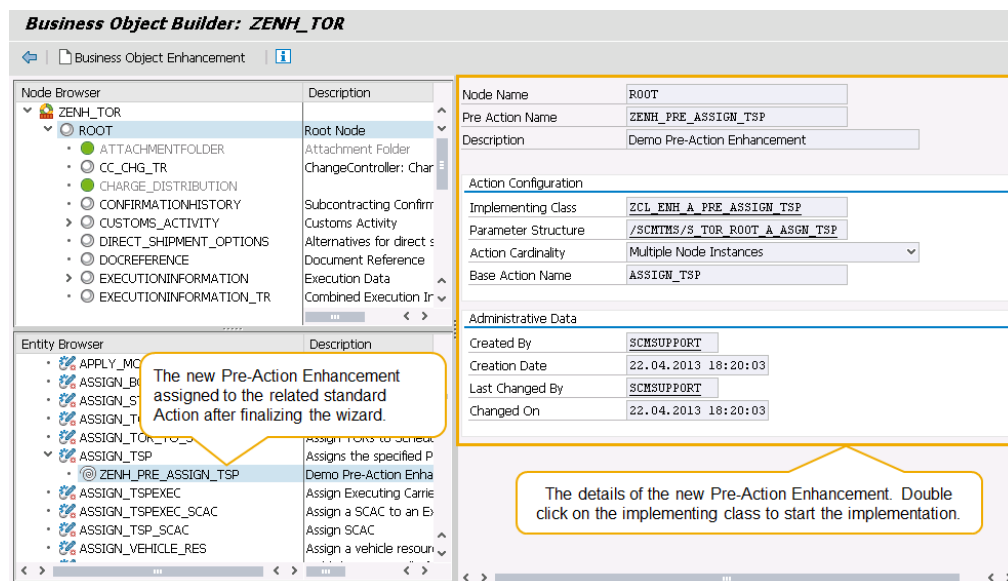
- 2) The first step in the wizard is to specify the name for the new Pre-Action enhancement and a description on the semantic and purpose of this new entity. Example:

Action Name : *ZENH\_PRE\_ASSIGN\_TSP*  
Description : Demo Pre-Action Enhancement

- 3) In the second step you need to define the implementing class of the new action validation. Example: *ZCL\_ENH\_A\_PRE\_ASSIGN\_TSP*.

The implementing class must implement the interface */BOBF/IF\_FRW\_ACTION*.

- 4) Click on button *Complete* to finalize the creation of the new Pre-Action enhancement.



Picture: The final new Pre-Action enhancement assigned to the respective action.

- 5) Finally, you need to implement your business logic in the consistency validation's implementing class that you have specified in the previous steps. Double click on the implementing class in the Pre-Action Enhancement details to start the implementation.

### 3.3.9 Creating Consistency Validations

Consistency validations can be used to check the consistency of a business object. It is possible to check whether or not a certain set of node instances of a certain node are consistent. The consistency validation implementation returns a set of failed keys identifying all handed over node instances that are inconsistent.

New enhancement consistency validations can be added to extensible standard nodes and new subnodes. The wizard for this task guides you through the following required steps.

- 1) Open the corresponding Enhancement Object and select the node that shall be extended with a new consistency validation. In the context menu of the node (click right mouse button) choose **Create Consistency Validation** to start the wizard. Example:

The new subnode `ZENH_ROOT_SUBNODE` of the Freight Order BO in the example Enhancement Object `ZENH_TOR` created in section 3.3.5.

- 2) The first step in the wizard is specifying the name for the new consistency validation and a description on the semantic and purpose of this new entity. Example:

Validation Name : `ZENH_DEMO_CONSVAL_BS`  
Description : Demo Enh. Cons. Validation Before Save

- 3) In the second step you need to define the implementing class of the new consistency validation. Example: `ZCL_ENH_V_DEMO_VAL_BS`.

The implementing class must implement interface `/BOBF/IF_FRW_VALIDATION`.

- 4) Maintain Request Nodes: A consistency validation is automatically executed as soon as one of the triggering conditions of its request nodes is fulfilled. In this wizard step, the request nodes and the corresponding triggering condition are defined.

On this screen, all nodes are shown that are connected to the assigned node by an association. To maintain a request node, select the request node checkbox and the appropriate triggering condition (Create, Update or Delete). Example:

Request Node	Node	Create	Update	Delete
Yes	<code>ZENH_ROOT_SUBNODE</code>	Yes	Yes	No
No	<code>ROOT</code>	No	No	No

With these settings, the consistency validation will be triggered when instances of node `ZENH_ROOT_SUBNODE` are created or updated. Assume we declared node `ROOT` as a request node and marked the triggering condition Update. In this case, the consistency validation would also be triggered when the Root node is updated.

Consistency Validation Configuration

Implementing Class: ZCL\_ENH\_V\_DEMO\_VAL\_BS

Validation Category: Consistency Check

Validation Impact: Return messages

Status Variable:

Request Nodes	Create	Update	Delete
<input checked="" type="checkbox"/> ZENH_ROOT_SUBNODE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> ROOT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Creation Date: 05.10.2012 13:07:05

Last Changed By: SCMSUPPORT

Changed On: 05.10.2012 13:23:51

Picture: Request Nodes and Triggering Conditions.

5) Maintain the Impact: The consistency validation shall indicate changed node instances that are inconsistent by the help of messages. You can prevent the system from saving the entire transaction if a changed instance fails the consistency validation, or fails to set a consistency status. You can maintain the type of reaction on inconsistent node instances as validation impacts in this wizard screen. Options:

- **Return messages:** This represents the default behavior of consistency validation. The validation implementation returns messages for inconsistent instances to the consumer.
- **Return messages and prevent saving:** Select this option for the validation impact if the inconsistency of a node instance must be solved before saving the transaction.
- **Return messages and set a consistency status:** If the base object (super object) contains a consistency status, this status can be influenced by a consistency validation. Choose the appropriate status variable. When a changed instance fails the consistency validation, this status is automatically set to inconsistent.

6) Click on button *Complete* to finalize the creation of the new consistency validation.

Business Object Builder: ZENH\_TOR

Business Object Enhancement

Node Browser

- ORGUNIT\_BP\_RESTRICTION
- OVERVIEW
- PARTY
- RANKINGLIST
- STOP
- SUMMARY
- TENDERING
- TEXTCOLLECTION
- TRANSPORTCHARGES
- TRANSPORTCHARGES\_INTERNAL
- ZENH\_ROOT\_SUBNODE

Entity Browser

- Determinations
- Consistency Validations
- Queries

Node Name: ZENH\_ROOT\_SUBNODE

Validation Name: ZENH\_DEMO\_CONSVAL\_BS

Description: Demo Enh. Cons. Validation Before Save

Consistency Validation Configuration

Implementing Class: ZCL\_ENH\_V\_DEMO\_VAL\_BS

Validation Category: Consistency Check

Validation Impact: Return messages

Status Variable:

Request Nodes	Create	Update	Delete
<input checked="" type="checkbox"/> ZENH_ROOT_SUBNODE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> ROOT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Administrative Data

Created By: SCMSUPPORT

Creation Date: 05.10.2012 13:07:05

Last Changed By: SCMSUPPORT

Changed On: 05.10.2012 13:23:51



Picture: The final new consistency validation assigned to the respective node.

- 7) Finally, you need to implement your business logic in the consistency validation's implementing class that you have specified in the previous steps. Double click on the implementing class in the consistency validation details to start the implementation.

Note: Validations are always executed after Determinations (see also next section), i.e. first the Determinations compute new data or derive it from existing data. After execution of the Determinations the then current data is checked for consistency by Validations.

### 3.3.10 Creating Determinations

A determination is mainly used to compute data that can be derived from the values of other attributes. The determined attributes and the determining attributes of the triggering condition can belong to the same node or to different nodes. There are also values that do not depend on any other value but still have to be determined automatically on the creation or modification of a node instance, for example IDs.

A determination is assigned to a business object node. It describes internal changing business logic on the business object. A determination is automatically executed by the BOPF as soon as the BOPF triggering condition is fulfilled. This triggering condition is checked by the framework at different points in the transaction, depending on the pattern of the determination. For each determination, it is necessary to specify the changes that build the triggering condition. Changes can include creating, updating, deleting, or loading node instances.

As soon as the framework checks the trigger conditions of determinations and there is more than one determination to be executed, the dependencies of the determinations are considered. With the help of a determination dependency, a determination can be maintained either as a predecessor or a successor of another determination.

The four supported determination patterns (in the following referenced with A, B, C and D) are:

#### A) Derive dependent data immediately after modification (After Modify):

The trigger condition of the determination is evaluated at the end of each modification. A modification roundtrip is defined as one single modification core service call from the consumer to the framework. The call contains arbitrary creations, updates, or deletions of node instances. Additionally, the trigger condition is checked after each action core service execution.

The pattern shall be used if creating, updating, or deleting of node instances leads to unforeseen errors. These errors are handled during the same roundtrip. If there is no need to react immediately on the modification, and the handling of the side effect is very time consuming, we recommend you use the Derive dependent data before saving determination pattern instead.

Example: As soon as a new ITEM node instance of the Freight Order business object is added, the changed total quantity on the Root Node (header level) must be immediately recalculated in order to show the new quantity on the consumer's user interface.

#### B) Derive dependent data before saving (Before Save):

The trigger condition is checked as soon as the consumer saves the whole transaction. If the save of the transaction fails, these determinations could run multiple times.

In contrast to the "Derive dependent data immediately after modification" pattern, the framework evaluates all changes done so far in the current transaction to check the trigger condition. Because this evaluation only takes place at the save phase of the transaction, this pattern is recommended for time consuming determinations.

Example: The data of each invoice must additionally be stored in a XML file. A determination is configured to extract the XML code from each changed invoice. Because this is very time consuming the determination does not run immediately after each change of an invoice. Instead, it runs once before saving for all invoices changed during the current transaction.

### C) Fill transient attributes of persistent nodes (After Loading):

The determination is automatically executed before the consumer accesses a transient node attribute of the assigned node for the first time. This allows you to initially derive the values of the attribute. In addition, these determinations are executed after each modification of a node instance. This allows you to recalculate the transient field if its derivation source attribute has been changed by the modification.

These determinations are used to derive the values of transient attributes of a node.

Example: The volume of a certain item of a Freight Order can be derived out of the length, width, and height of this item unit, and the quantity. The volume attribute is a transient attribute of the item node, and its value can be derived as soon as an item is loaded from the database. Therefore, you can use a determination that calculates and fills the volume at this point in time.

### D) Derive instances of transient nodes (Before Retrieve):

The determination is executed before the consumer accesses the assigned transient node of the determination and allows the creation, update or deletion of transient node instances.

These determinations are used to create and update instances of transient nodes. Because the determinations are executed before each access to their assigned transient node, they must ensure that the requested instances are in a consistent state.

Example: The Forwarding Settlement business object does not store payment options. A payment option node is added as transient node buffering detailed information that is located in another system.

### E) Create Properties:

The determination is used to create dynamic properties of business object node instances at runtime. It overrides the static properties of BO node instances provided that the default properties of the node defined at design time are not defined as final.

Example: For a Forwarding Order a Forwarding Settlement Document is created via an Action. A status attribute is set on the Root Node of the Forwarding Order indicating that Settlement has taken place already for the Forwarding Order. Consequently, the determination sets the *UPDATE\_ENABLED* attribute of the ITEM node from True to False indicating that the node cannot be updated anymore.

New enhancement determinations can be added to extensible standard nodes and new subnodes. The wizard for this task guides you through the following required steps.

- 1) Open the corresponding Enhancement Object and select the node that shall be extended with a new determination. In the context menu of the node (click right mouse button) choose *Create Determination* to start the wizard. Example:

The new subnode *ZENH\_ROOT\_SUBNODE* of the Freight Order BO in the example Enhancement Object *ZENH\_TOR* created in section 3.3.5.

- 2) The first step in the wizard is to specify the name for the new determination and a description on the semantic and purpose of this new entity. Example:

Determination Name : *ZENH\_DEMO\_DET\_AM*  
Description : Demo Enh. Determination After Modify

- 3) In the second step you need to define the implementing class of the new determination. Example: `ZCL_ENH_D_DEMO_DET_AM`.

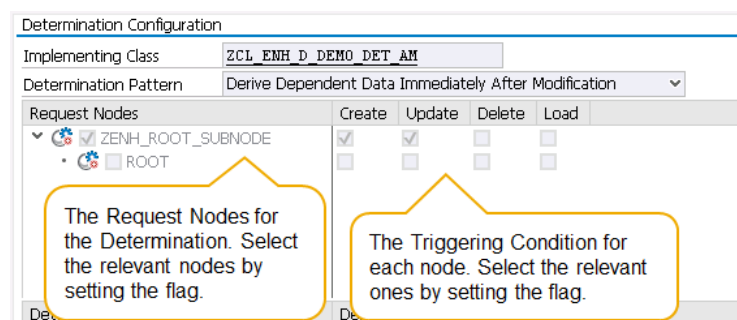
The implementing class must implement interface `/BOBF/IF_FRW_DETERMINATION`.

- 4) Maintain the determination pattern: In this step you can choose one of the determination patterns described before via a radio button.
- 5) Maintain Request Nodes: This is only required and done for determination patterns A and B). A determination is automatically executed as soon as one of the triggering conditions of its request nodes is fulfilled. In this wizard step, the request nodes and the corresponding triggering condition are defined.

On this screen, all nodes are shown that are connected to the assigned node by an association. To maintain a request node, select the request node checkbox and the appropriate triggering condition (Create, Update or Delete). Example:

Request Node	Node	Create	Update	Delete
Yes	ZENH_ROOT_SUBNODE	Yes	Yes	No
No	ROOT	No	No	No

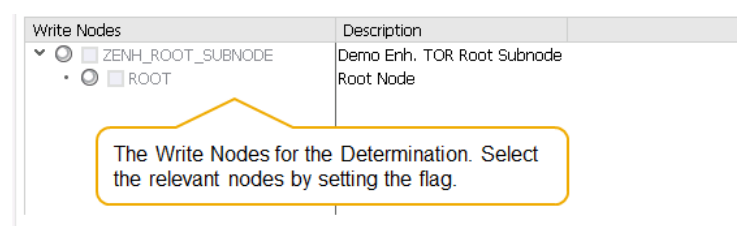
With these settings, the determination will be triggered when instances of node `ZENH_SUBNODE` are created or updated. Assume we declared node `ROOT` as a request node and marked the triggering condition Update. In this case, the determination would also be triggered when the Root node is updated.



Picture: Request Nodes and Triggering Conditions.

- 6) Maintain Transient Node: This is only required and done for determination pattern D. Select the transient node whose instances shall be modified by the determination.
- 7) Maintain Write Nodes: Select all the nodes whose instances are created or modified by the determination. The write nodes ensure that the related BO nodes get locked for writing before the determination is executed. Example:

Write Node	Node
Yes	ZENH_SUBNODE
No	ROOT



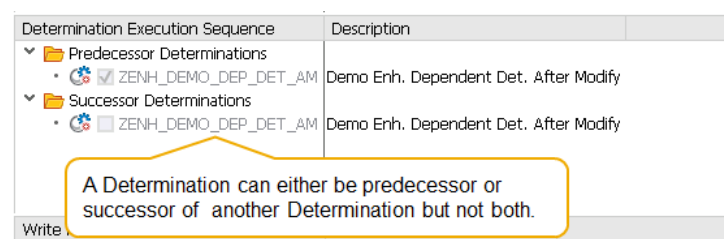
Picture: Write Nodes for a Determination

#### 8) Maintain Determination Dependencies:

As mentioned before, enhancement determinations are only executed after all standard determinations have been executed to ensure that the standard business logic is not disrupted and leads to inconsistencies. Nevertheless, you can define dependencies between enhancement determinations with the same determination pattern (this can only be the case for pattern A and B), i.e. this step will only come up in the wizard in case there is more than one determination of the same pattern present.

On the wizard screen select the determinations which must be processed before or after the determination currently being created or configured. The triggering condition of the determination must also be fulfilled to get executed.

For each determination you can define if it is either executed before (predecessor) or after the current determination (successor). This allows defining an execution sequence of the enhancement determinations.



Picture: Defining the execution sequence of a determination.

In the picture above determination `ZENH_DEMO_DET_AM` gets assigned determination `ZENH_DEMO_DEP_DET_AM` as a predecessor determination. This second determination follows the same determination pattern (`AM` = After Modify; otherwise it would not be listed as an option for an Execution Sequence definition in this example). Moreover the `BOPF` framework will then automatically define determination `ZENH_DEMO_DET_AM` as a successor determination for `ZENH_DEMO_DEP_DET_AM`.

- 8) Click on button *Complete* to finalize the creation of the new determination.
- 9) Finally, you need to implement your business logic in the determination's implementing class that you have specified in the previous steps. Double click on the implementing class in the determination details to start the implementation.

**Note:** To make sure that the `BOPF` framework does not slowdown in its performance you should follow the rule “as much as necessary and as few as possible additional Determinations”. Instead of adding multiple Determinations to a BO node make sure that you group your determinations by triggering condition and transactional point in time (execution time point), i.e. everything that has the same triggering condition and the same transactional point in time shall be implemented in just one additional determination. This prevents the `BOPF` Framework from handling too many single Determinations (can cause quite some overhead) at runtime and therefore helps to prevent increased runtimes of the enhanced BO. You can find a few more details on this topic in section 3.4.

### 3.3.11 Creating Queries

Queries are the initial point of access to business objects. They allow performing searches on a business object to receive the keys or the data of specific or all node instances. Each query has an associated data structure representing the search criteria. To allow different search criteria at runtime, the consumer may hand over this query data type with discrete search values. A query returns the queried keys (along with the related data if required) of the business object node instances that match the provided search criteria.

The `BOBF` Enhancement Workbench allows creating new queries of three different types:

- **Node Attribute Query:**

The search parameters are equal to the assigned node of the query. The search criteria can consist of value comparisons and value ranges on the attributes of the nodes. The query returns the instances of the assigned node whose attributes match the provided search parameters.

Node attribute queries are recommended for all cases that do not need complex query logic. In contrast to custom queries, node attribute queries are only modeled and therefore do not have to be implemented (they are realized in a generic way).

- **Custom Query:**

Queries of this type execute application specific logic. In contrast to the node attribute query, this logic is realized in an implementing class that is assigned to the query. A Custom Query can have an arbitrary data type structure that is handed over by the consumer to the query implementation at runtime. This data type represents the search criteria of this query.

- **Custom Query (Generic Result Query):**

In addition it can also (but must not necessarily) have an arbitrary result table type and an associated result type (represents the structure of the result). A custom query of this type is also referred to as Generic Result Query. As per *NetWeaver 7.31 Service Pack 06* (also for *NetWeaver 7.02 Service Pack 11*) and above the `BOPF` Enhancement Workbench allows customers and partners creating their own Generic Result Queries. So the Enhancement Workbench now enables e.g. assigning a query to the `ROOT` node which returns instances or keys of e.g. the `ITEM` node.

In general Custom Queries are used if the recommended node attribute queries do not fulfill the requirements, e.g. if specific query parameters must be handed over, or if the query logic is more complex than simply comparing attribute values.

Query Type	Input/ Search Criteria	Output / Result	Implementing Class
Node Attribute Query	Data Type with attributes from the BO node that the query is assigned to.	A list of keys of the BO node instances that match the search criteria (and the node data of these instances).	Not required.
Custom Query	Arbitrary Data Type with attributes that represent the search criteria	A list of keys of those BO node instances that match the search criteria (and the node data of these instances)	Required.
Custom Query (Generic Result Query)	Arbitrary Data Type with attributes that represent the search criteria	An arbitrary result data type and related table type representing the result data	Required.

The table above shows a summarized classification of the different queries. As mentioned, new enhancement queries of the listed types can be added to extensible standard nodes and new sub nodes. The wizard for this task guides you through the following required steps.

- 1) Open the corresponding Enhancement Object and select the node that the new query shall be assigned to. In the context menu of the node (click right mouse button) choose *Create Query* to start the wizard. Example:

The new subnode *ZENH\_ROOT\_SUBNODE* of the Freight Order BO in the example Enhancement Object *ZENH\_TOR* created in section 3.3.5.

- 2) The first step in the wizard is to specify the name for the new query and a description on the semantic and purpose of this new entity. Examples:

For a Node Attribute Query:

Query Name : *ZENH\_DEMO\_NA\_QUERY*

Description : Demo Enh. Node Attribute Query

For a Custom Query:

Query Name : *ZENH\_DEMO\_CUST\_QUERY*

Description : Demo Enh. Custom Query

For a Custom Query (Generic Result Query):

Query Name : *ZENH\_DEMO\_GENRES\_QUERY*

Description : Demo Enh. Generic Result Query

- 3) Choose the Query Type. If you choose type *Node Attribute Query* finalize the creation of the new query with step 6. In case of type *Custom Query* continue with step 4.
- 4) Specify the implementing class and a query data type for the new query (only required in case of Custom Queries). Example:

Implementing Class : *ZCL\_ENH\_Q\_DEMO\_CUST\_QUERY*

Data Type : *ZENH\_S\_Q\_DEMO\_CUST\_QUERY*

The implementing class must implement interface */BOBF/IF\_FRW\_QUERY*. When finalizing the wizard, this interface will automatically be assigned to the specified class.

The Data Type represents the attributes that are available to be used as search criteria. When creating the example Data Type *ZENH\_S\_Q\_DEMO\_CUST\_QUERY* define e.g. the following subset of fields (the listed ones represent a subset of the attributes defined on the node that the new Query gets assigned to).



Component	Typing Method	Component Type
TOR_ID	Types	/SCMTMS/TOR_ID
TOR_CAT	Types	/SCMTMS/TOR_CATEGORY
TOR_TYPE	Types	/SCMTMS/TOR_TYPE
...	...	...

An alternative in this example Custom Query would be simply reusing the complete Node Structure of the node that the query will be assigned to or of course any arbitrary structure. In contrast a Node Attribute Query automatically reuses the Node Structure, i.e. it provides all the available node attributes as search criteria).

If you do not specify a result type and result table type you can already start implementing the query in the implementing class that you specified. In this case the result of the query is represented by the structure of the node that the query is assigned to (i.e. keys of found node instances along with the corresponding data if required).

#### 5) In case you create a Generic Result Query:

As mentioned, this type of query allows an arbitrary data type as well as an arbitrary result type (and result table type) and is e.g. the basis for any Personal Object Work List (POWL) in SAP TM (see also section 6).

In addition to the data type mentioned above, the wizard allows specifying a result type and result table type that represent an arbitrary result structure of the query instead of a fixed one. So the result is not necessarily returning data of the related node but simply a table of result type data records (e.g. merged data from different BO Nodes) that were found based on the provided data type search criteria.

Most important when defining the result type: The first attribute must be *DB\_KEY* of type */BOBF/CONF\_KEY* (Query *QDB\_ROOT\_TENDERING\_BY\_ELEMENTS* defined on the ROOT Node of BO */SCMTMS/TOR* represents an example for such a Query). This means that the first attribute of the result structure is a node instance key of the node that the Generic Result Query is assigned to. The rest of the result structure attributes is arbitrary. Example:

Result Data Type : *ZENH\_S\_Q\_DEMO\_GENRES\_QUERY\_R*  
Result Table Type : *ZENH\_T\_Q\_DEMO\_GENRES\_QUERY\_R*

Attributes of the result data type:

Component	Typing Method	Component Type
DB_KEY	Types	/BOBF/CONF_KEY
CREATED_ON	Types	/BOFU/TSTMP_CREATION_TIME
TOR_ID	Types	/SCMTMS/TOR_ID
TOR_CAT	Types	/SCMTMS/TOR_CATEGORY
TOR_TYPE	Types	/SCMTMS/TOR_TYPE
...	...	...

Use the result data type as the line type for the mentioned result table type.

- 6) Click on button *Complete* to finalize the creation of the new query.
- 7) Implementation of the query logic is required in case of a Custom Query. Double click on the implementing class in the query details to start the implementation.

SAP TM provides a Query Super class */SCMTMS/CL\_Q\_SUPERCLASS* that can be reused for any Query implementation. It is recommended to let your Query implementations inherit from this super class as it provides some basic reuse methods (that can be overwritten) and enables an enhancement concept for existing standard



Queries. The BOBF Enhancement Workbench does not support enhancing existing standard Queries. How to enhance existing standard queries is described in section 6.1.

### 3.3.12 Creating custom Business Objects

As per NW 7.31 Service Pack 06 (also for NW 7.02 Service Pack 11) the BOPF Enhancement Workbench allows customers / partners to create their own Business Objects that they can fully implement on their own according to their requirements. In general all the functions introduced in the previous sections which allow creating enhancements for Standard BOs are also used to create the node hierarchy and structure as well as all relevant node elements like queries, actions, determinations and validations.

As for the other functions, a wizard supports you in creating a new custom Business Object. The following steps illustrate an example:

- 1) In the Enhancement Workbench click on button *Custom Business Object* (F2).
- 2) The first step in the wizard is to specify a name and a description for the new BO. Example:

Namespace : Z  
 Prefix : CUST  
 Name : DEMO\_BO  
 Description : Demo Custom Business Object

In the next step you specify the name for the Constants Interface of the new Business Object. You can click on button *Propose Name* to let the wizard automatically propose a name for this interfaces that follows the BOPF naming conventions. Example:

Constants Interface : ZIF\_CUST\_DEMO\_BO\_C

- 3) The very first node of a Business Object is always the Root Node. In this step we specify the name of the Root Node (it is recommended to always choose *ROOT* as the name for this node) and a corresponding description. Moreover we need to specify the persistent data structure (and the transient if required) that contain the attributes of the node. The later on required combined structure that besides the data attributes also contains the technical key attributes *KEY*, *PARENT\_KEY* and *ROOT\_KEY* is generated automatically when finalizing the wizard. Example:

Root Node Name : Root  
 Root Node Description : Demo Custom BO Root Node  
 Persistent Structure : ZSCUST\_ROOT\_D  
 Transient Structure : ZSCUST\_ROOT\_DT

- 4) Double click on structure *ZSCUST\_ROOT\_D* to create the persistent structure of the Root Node. This will guide you to transaction *SE11* where you can specify this structure. Example:

Demo Custom BO Persistent Node Structure		
Component	Typing Method	Component Type
TOR_ID	Types	/SCMTMS/TOR_ID
TOR_CAT	Types	/SCMTMS/TOR_CATEGORY
TOR_TYPE	Types	/SCMTMS/TOR_TYPE
.INCLUDE	Types	/BOFU/S_ADMIN_DATA

As Enhancement Category for such structures always choose option *Can be enhanced (character-type or numeric)*. Save and activate the structure.

- 5) Just like the persistent structure, the transient structure is created. Example:

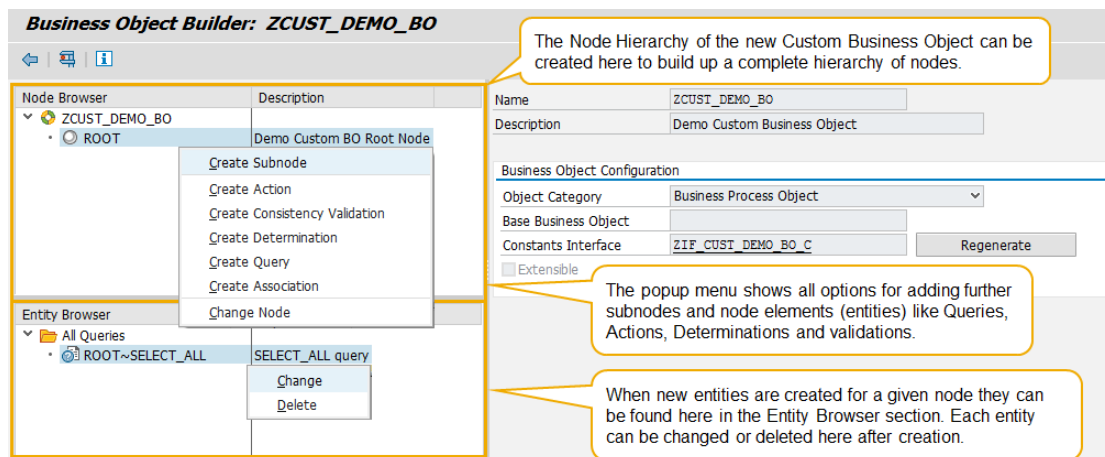
Short Description	Demo Custom BO Transient Node Structure	
Component	Typing Method	Component Type
RUNTIME_STATUS	Types	FLAG
LAST_CHANGED	Types	/SCMTMS/DATETIME

As Enhancement Category again choose option *Can be enhanced (character-type or numeric)*. Save and activate the structure.

- 6) The next step is to specify names for the Combined Structure, the related Combined Table Type and the Database Table that will contain the persisted Root Node data. These structures and table types are derived from the structures specified in step 5) and 6) and will be automatically created / generated by the wizard. Example:

Combined Structure : ZSCUST\_ROOT\_K  
 Combined Table Type : ZTCUST\_ROOT\_K  
 Database Table : ZDCUST\_ROOT

- 7) Now we can finalize the wizard by clicking on button *Complete*. This will then generate all other required objects for the new Custom Business Object like the combined structures, the Database Table for the first Node ROOT and the Constants Interface.



Picture: The new Custom BO after finalizing the wizard.

Custom Business Objects are accessed just like any other BOPF BO, i.e. using a Service Manager and a Transaction Manager. Moreover you can also find and edit them in transaction */BOBF/CONF\_UI*. This allows adding e.g. cross BO associations to Standard BOs and a few other elements that are not (yet) supported by the Business Object Builder within the Enhancement Workbench.

### 3.4 Advanced BOPF Topics

In this section, a few further topics are described that you should know about when developing applications with BOPF. Especially the section on performance should be reviewed and the recommendations kept in mind to prevent some serious performance issues. The majority of performance issues detected in BOPF are actually not caused by the framework itself but rather by coding using BOPF in a critical or suboptimal way.

#### 3.4.1 Properties

Each node of a BOPF BO model has a corresponding node that carries related property information for the entities *Node* (i.e. for the node itself), *Node Attributes*, *Actions*, *Action Parameters*, *Associations* and *Association Parameters*. These properties are e.g. used to control the behavior of a node attribute on the UI. For this, the application can retrieve the current property values and react on it. If e.g. a node attribute has its property *Read-Only* set it is not ready for input on the UI. Or when property *Mandatory* is set, the attribute must be provided with a corresponding value and is not allowed to have no value.

The following properties are available and can be set (information is stored in a node's property node that usually has the same name like the node but with the postfix *\_PROPERTY* assigned to it:

Property	Comment
Enabled	If set, the entity can be used. If not set, it cannot be used at all.
Read-Only	The entity can only be displayed but not changed.
Mandatory	The entity must have a value. Two semantics are distinguished: <ul style="list-style-type: none"> <li>• Must be filled during the initial create or update modification call.</li> <li>• Must be filled the latest until finalize phase.</li> </ul>
Create-Enabled	New instances of the node can be created/ Instances can be created by the help of an association.
Update-Enabled	Existing instances of the node can be changed.
Delete-Enabled	Existing instances of the node can be deleted.

These properties can have a static or dynamic character. Static Properties are in general defined during design time, i.e. when modeling the respective BO entity. While Static Properties can be overruled by dynamic properties at runtime ("default property values") Final Static Properties are final and cannot be changed at runtime.

Dynamic properties can be changed at runtime. Lock-dependent dynamic properties are e.g. implicitly set to *Read-Only* in case the requesting User does not have a lock on the corresponding entity. Application specific dynamic properties are set by so called Property Determinations, e.g. a determination registered on *AFTER\_MODIFY* that sets a node attribute on *Read-Only* if another attribute has a certain value. The already mentioned Property Node carry the dynamic properties for instances of the related node at runtime (i.e. they are transient nodes).

#### Static Properties

The static properties of a node are associated with so called Node Category. A Node Category can be used to bundle node instances that have an identical data structure but a different semantic / behavior. The default Node Category is available for each BO node and has the same name like the related node. During the modeling process, multiple Node Categories can be defined.

Example: Different Node Categories for node *Item* might be provided, one that represents e.g. Freight Order items with associated costs and another one that represents items which are free of charge.

In the SAP TM standard, this BOPF modeling concept has been rarely used. Instead the SAP Standard nodes carry specific node attributes that define the semantic of a node instance with

corresponding values (e.g. *TRQ\_TYPE* and *TRQ\_CAT* on the Root Node of the TRQ BO). Nevertheless, the static properties of a node are maintained via the corresponding Node Category, i.e. in most cases here via the standard node category. The static properties are defined at design time. The following picture shows an example in transaction */BOBF/CONF\_UI* where you can find the Node Categories in the list of Node Elements.

**Display Business Object /SCMTMS/TOR, Active Version**

Business Object Detail Browser | Description | Node Category | Associations | Actions | Determinations | Validations | **Attribute Properties**

The static properties of a Node Category can be found here.

The Node Categories for the Root Node of e.g. the TOR BO. In this example, only the standard Node Category is defined (by default).

Node Attribute Properties: ROOT

Exception	Enabled	Enabled (Default)	Final	Read-Only	Read-Only (Default)	Final	Mandatory	Mandatory (Default)	Final
CB_MTR_COUNTRY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CHANGED_BY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CHANGED_ON	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CMPLC_CHK_ENABLE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CM_ID	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
COLOAD_IND	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
COMMPYID	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
COMMPY_KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
COMPLIANCE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CONFIRMATION	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CONSIGNEEID	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CONSIGNEE_KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CONSOL_TYPE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Example: The attributes of *CHANGED\_BY* and *CHANGED\_ON* have the final static property Read-Only, i.e. these fields can only be displayed and the property can not be overruled by dynamic property changes (content is actually provided by a Determination).

Picture: The static properties of Node Attributes.

## Dynamic Properties

Dynamic Properties can be defined for various BOPF entities. In the table below you can see which entity can have or can make use of which dynamic property (not all entities can use all dynamic properties).

Entity	Property					
	Enabled	Read-Only	Mandatory	Create-Enabled	Update-Enabled	Delete-Enabled
Node					Available	Available
Node Attribute	Available	Available	Available			
Association	Available			Available		
Association Parameter	Available					
Action	Available					
Action Parameter	Available					

*Enabled* entities are available for requests. An entity that is not enabled cannot be used. Node Attributes can be *Read-Only* or *Mandatory*. A mandatory Node Attribute must be provided with a value before Save, i.e. a mandatory attribute must not necessarily have a value at the time when the carrying node is created.

An Association can be *Create-Enabled*. For example, when creating Sub Node instances (e.g. for an Item Node) for a given Root Node instance, the Composition Association Item (pointing from the Root Node to the Item Node) must be provided to the modification table. When the dynamic property *Create-Enabled* of this Composition Association is set to false, the creation of Item Node instances will not be possible.

With the properties *Update-Enabled* and *Delete-Enabled* for a node you can define whether instances of the node are allowed to be updated or deleted. But keep in mind that dynamic properties are transient and not persisted, i.e. they are set and only present at runtime in contrast to the static properties.

In the configuration (design time) of a node you can set a flag *Sub Tree Properties Used* that allows propagating the dynamic properties *Create-Enabled*, *Update-Enabled* and *Delete-Enabled* to the sub tree of the node. Such sub tree properties cannot be overwritten by other dynamic properties.

## Handling Dynamic Properties

Dynamic properties are determined by Property Determinations. Such determinations are assigned for each node for which dynamic properties are used. They have the corresponding property Node defined as the request node and do not have a triggering condition. The determination is registered on transactional point in time *Before Retrieve*. The implementing class for the determination will contain the code to set the required dynamic properties.

BOPF provides helper class `/BOBF/CL_LIB_H_SET_PROPERTY` that provides methods to set dynamic properties. They can be used in your own implementations. The following example uses method `SET_ATTRIBUTE_MANDATORY`:

```
* Instanciate the helper class
DATA: lo_set_property TYPE REF TO /bobf/cl_lib_h_set_property,
      ls_node_inst_key TYPE /bobf/conf_key,
      lt_node_inst_key TYPE /bobf/t_frw_key2.

CREATE OBJECT lo_set_property
EXPORTING
  is_context = is_ctx
```

```

        io_modify = io_modify.

* Get list of node instance keys

* Set Node Attribute LABELTXT of the TOR Root Node as a
* mandatory attribute for each relevant node instance
LOOP AT lt_node_inst_key INTO ls_node_inst_key.
...
    CALL METHOD lo_set_property->set_attribute_mandatory
    EXPORTING
        iv_attribute_name = /scmtms/if_tor_c=>
                                sc_node_attribute-root-labeltxt
        iv_key              = ls_node_inst_key
        iv_value            = abap_true.
...
ENDLOOP.
...

```

Assume you implement method *Execute* of a Property Determination for the TOR Root Node and want to set the Root Node attribute *LABELTXT* as a mandatory field. In the above-mentioned example, the helper class is instantiated with the parameters *IS\_CONTEXT* and *IO\_MODIFY* provided by the interface of method *Execute* (of the Determination Interface), i.e. the created object will know the same context like your determination.

You can then loop over the node instance keys for which the determination is running and set the dynamic attribute property to *Mandatory* for mentioned attribute. Take a look at the BOBF helper class to find further such setter methods for dynamic properties of other entities. They are called in the same described way (with different parameters depending on the entity).

With such a determination, you can e.g. implement a logic that switches a customer / partner specific extension fields to mandatory depending on the value of another attribute (e.g. the status of a document, etc.).

### 3.4.2 Message Concept

BOPF uses a message object for transporting messages through the call stack that might have been issued during the execution of e.g. an Action, a Determination, a Validation or a Query. The interface of each of these BOPF entities provides a corresponding parameter *EO\_MESSAGE* which is a reference to interface */BOBF/IF\_FRW\_MESSAGE*. In this exporting parameter, you can find the message object instances that have been issued during execution of the related BOPF entity.

The concept is based on class-based messages, i.e. you can create your own message classes that should inherit from the BOPF standard class */BOBF/CM\_FRM*. You can then instantiate a BOPF message of such a class and attach it to the message object as shown in this example:

```

DATA co_message TYPE REF TO /bobf/if_frw_message.
CREATE OBJECT lm_applog TYPE /scmtms/cm_applog_msg
EXPORTING
    symptom              = /scmtms/cm_applog_msg=>sc_symptom_log_ctx
    ms_origin_location   = ls_location      (key of affected entity)
    Severity              = ls_msg-msgty    (severity of the message)
    textid               = ls_t100         (key for message text)
    mv_attr1             = ls_msg-msgv1
    mv_attr2             = ls_msg-msgv2
    mv_attr3             = ls_msg-msgv3
    mv_attr4             = ls_msg-msgv4
    mv_detlevel          = lv_detlevel
    mv_probclass         = lv_probclass

```

```

ms_context      = is_context
mv_bopf_location = iv_bopf_location_key.

```

You can then add this message object instance to parameter *EO\_MESSAGE* by using its method *ADD\_CM* as follows:

```
co_message->add_cm( io_message = lm_applog ).
```

Note that multiple Message Object instances can exist at runtime that independently store messages. In the example above we have actually created a new message object *CO\_MESSAGE* besides the mentioned *EO\_MESSAGE*. If you want to pass any message of e.g. an Action through the call stack you need to assign your message instances to parameter *EO\_MESSAGE* of course.

As you can see in this example, it actually takes quite some lines of coding to e.g. just pass a simple error message. Moreover, for support and maintenance purposes this approach is not suited as it is not possible to properly use the *Where-Used-Functionality* to find out where a certain message has been issued. It is therefore recommended to always use the concept for issuing messages as done in the following simple example of an Action implementation:

```

FIELD-SYMBOLS: <fs_parameters> TYPE zenh_s_a_root_demo_action.

DATA: lv_temp TYPE c.

* take over action parameters
ASSIGN is_parameters->* TO <fs_parameters>.

* use parameter Z_FLAG
IF <fs_parameters>-success = abap_true.
  MESSAGE s002(zenh_messages) INTO lv_temp.
ELSE.
  MESSAGE e003(zenh_messages) INTO lv_temp.
ENDIF.

CALL METHOD /scmtms/cl_common_helper=>msg_helper_add_symmsg(
  EXPORTING
    iv_key      = /scmtms/if_tor_c=>sc_bo_key
    iv_node_key = /scmtms/if_tor_c=>sc_node-root
  CHANGING
    co_message = eo_message ).

```

In this example the “classical” ABAP instruction *MESSAGE* is used to issue e.g. a success or an error message that was defined in a “classical” message class with transaction *SE91*. The message is issued into a variable of type C (yes, that’s right, a character with length 1). This ensures that the message is not actually displayed somewhere but nevertheless it will fill the corresponding fields *SY-MSGTY*, *SY-MSGNO*, etc. i.e. all relevant *SY-MSG\** fields.

Helper class */SCMTMS/CL\_COMMON\_HELPER*, method *MSG\_HELPER\_ADD\_SYMSG* now allows the direct transformation of the *SY-MSG\** fields into a corresponding BOPF message object instance that can be added e.g. to the Action Interface Parameter *EO\_MESSAGE*.

In this example the coding is much easier to understand, it does not comprise that many lines and the *Where-Used-Functionality* will definitely find all the places in coding where a message is issued. The concept reduces the implementation effort, keeps the coding readable and enables support and maintainability. In customer/partner specific implementations this should be the standard approach for issuing messages that need to be passed through the call stack (all the way up to be displayed on the UI).



### 3.4.3 Performance in the context of BOPF

As mentioned in the introduction the majority of performance issues detected in BOPF are actually not caused by the framework itself but rather by coding using BOPF in a critical or suboptimal way. In this section, a few use cases are listed that you should keep in mind when using the BOPF Framework for your own development or enhancements.

#### Mass Data Processing

The BOPF Framework enables mass data processing. Developers must implement all their BOPF related coding in a way that the mass data processing capabilities are utilized, i.e. in any case you need to make sure that you exclusively use mass calls to not run into performance issues with your implementation. Ideally the number of a BOPF method call is completely independent of the number of node instances that are to be processed with the call. Let's take a look at an example to illustrate this:

A **wrong** BOPF call that is not mass enabled:

```
LOOP AT it_key INTO ls_key.
  CLEAR lt_key.
  APPEND ls_key TO lt_key.

  io_read->retrieve(
    EXPORTING
      iv_node = if_constant=>sc_node
      it_key = lt_key
    IMPORTING et_data = lt_node ).

  READ TABLE lt_node INDEX 1 INTO ls_node. checksy-subrc.
...
ENDLOOP.
```

In this example the *Retrieve* is called directly in a loop, sequentially and separately for each of the node instance keys contained in internal table *IT\_KEY*, i.e. for one node instance at a time. Each node instance is then further processed before the next node instance is read. Here the BOPF Framework is overloaded with unnecessary calls. This approach dramatically slows down the performance and **must be prevented**, not only for *Retrieve* calls like in this example but also for *RetrieveByAssociation*, *Query* and *Action* calls.

The **right** way to implement a mass enabled call:

```
io_read->retrieve(
  EXPORTING
    iv_node = if_constant=>sc_node
    it_key = lt_key
  IMPORTING et_data = lt_node ).

LOOP AT lt_node_data ASSIGNING <fs_node_data>.
...
ENDLOOP.
```

In this revised example the *Retrieve* is first called for all node instance keys in *IT\_KEY* before the returned node data is used for further processing. So the basic principle to prevent performance issues is to first execute a *Retrieve*, a *RetrieveByAssociation*, a *Query*, an *Action* or a *Convert* (Convert Alternative Key) with all relevant node instance keys (that are passed to the corresponding method by table *IT\_KEY*) and only afterwards process returned data in a loop if required.

#### Determinations and Validations

Assume you have added e.g. an additional node to a Standard BO and need to add business logic to this node by implementing Determinations and Validations. Remember that these

BOPF entities are not called explicitly in the coding but are triggered and executed by the BOPF Framework automatically, depending on the transactional point in time and the triggering condition that they are registered for.

General rules for creating new Determinations and Validations are:

- As much Determinations and Validations as necessary as few as possible. This will prevent overloading the BOPF Framework with too many calls. Each call creates a certain overhead in the framework that sums up to quite some runtime and memory consumption with the number of Determinations and Validations that are executed by the framework.
- To keep the number of Determinations and Validations as small as possible group them by their assigned transactional point in time, i.e. if possible only one Determination and Validation per transactional point in time. In the majority of cases the triggering condition of e.g. a Determination registered on the same transactional point in time is the same so they can be grouped.

Example: Instead of implementing separate Determinations for each node attribute and register them on transactional point in time *After Modify* a single Determination should be implemented that contains the determination of all relevant node attribute.

- This approach will prevent the BOPF Framework from getting overloaded by too many Determination calls for a specific transactional point in time. Only in specific use cases it may be required to have an additional Determination registered to the same transactional point in time. This approach has been followed in the SAP TM standard implementation since TM 8.0 based on performance tests. Exceptions are e.g. the usage of the BOPF library determinations that can be reused for drawing document numbers or the administrative data of a node (*DET\_DRAW\_NUMBER* and *DET\_ADMIN\_DATA*).
- Before implementing a second Determination or Validation with the same transactional point in time you should check other options first.
- You should only implement exactly one Consistency and one Action Validation per BO node which will further help to reduce the number of Determination and Validation calls. Do not implement separate Consistency Validations for each aspect of the check logic required for a node and do not implement a separate Action Validation for each Action assigned to the corresponding node.

## Property Determinations

Dynamic Properties must be determined only by corresponding Property Determinations that are registered on *Before Retrieve*. Such a determination must not change any data except that of the property node available for the related node. It is not required to delete old dynamic properties before creating new ones. Existing entries available in the buffer will automatically be updated.

### 3.4.4 Status & Action Management (Consistency Groups)

For handling statuses of Business Object instances the reuse component Status & Action Management (SAM) is still in use in the TM 9.x release as it has been in predecessor releases.

Starting with SAP TM 8.0 the Status Management has been also realized manually by adding corresponding status attributes to BO nodes and Determinations that contain the logic to determine the correct status depending on the current BO data. This “manual” approach has the advantage that customers and partners can use the BOPF enhancement technologies presented so far to add further status attributes for their very own purposes.

So for customers and partners it is recommended to consider this as the first approach. Adding a new attribute to a BO node representing a status and adding a new determination with the required logic to set the new status attribute can be simply done by the means provided with the Enhancement Workbench. Nevertheless, there might be use cases where the usage of SAM helps getting to the required solution. The first use case shows how a Consistency Group can be configured and used to set a status. The second use case described how to use a Consistency Group to prevent saving a transaction in case of errors.

## Consistency Group for setting a status value

The first use case is based on a real-world scenario where a customer wanted a validation to update a certain new status attribute. Well, as we learned, actually only determinations can change data while validations only check data based on a given logic and issue e.g. an error message that is then displayed on the UI. But we will see how this can be accomplished.

The customer has added a new status attribute and assigned a new determination to the Item Node of the Forwarding Order Business Object (technical name `/SCMTMS/TRQ`). At runtime, the determination resets the new status attribute to its initial value (e.g. *Check is Pending*).

When the Forwarding Order is saved, a validation shall be executed which executes a call to an external system where the item data is checked for certain criteria. Depending on the check result for an item, the new status attribute shall be set to the corresponding value *Inconsistent* or *Consistent*.

In case of a business document created from scratch the external check will be executed for all items with the effect that saving takes a bit more time. But if item data is then later on changed, the validation is used to only execute the external check again for those items which have the status *Check Pending* or *Inconsistent*. So, in this case we need to get the validation to trigger an update of the status attribute. This can be achieved by assigning a validation to a Consistency Group.

The following steps describe how to setup and implement an example. Of course, you should create a corresponding Enhancement Object for BO `/SCMTMS/TRQ` (if not yet done up until here → see section 3.3.2)

### 1) Creating an Extension Field to represent the new Status.

First of all create an extension field that will represent the new status on the Item node of the Forwarding Order BO (`/SCMTMS/TRQ`). How to do this in detail is described in section 3.3.4. Create the following example:

Create the Append Structure `ZENH_TRQ_ITEM` in the (persistent) Extension Include of the Item node and place the following example status attribute there:

Attribute	Typing Method	Component Type
<code>ZENH_CONS_STATUS</code>	Types	<code>/SCMTMS/CONSISTENCY_STATUS</code>

Make sure to define Enhancement Category *Can be enhanced (character-type or numeric)* for the new Append. Then save and activate the Append Structure.

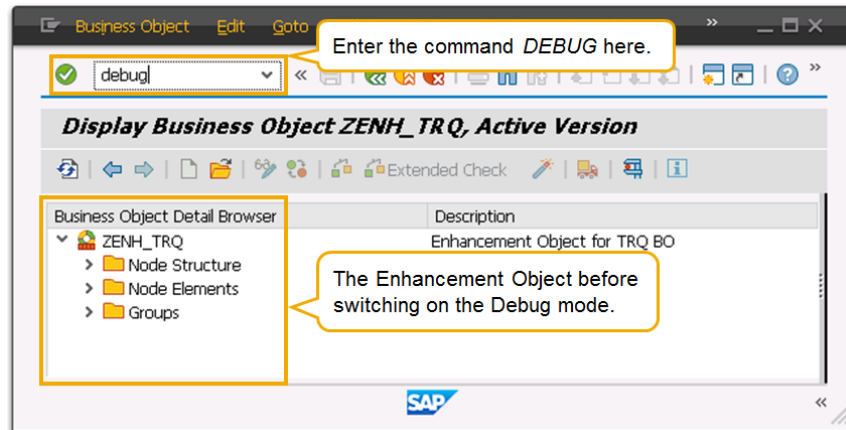
### 2) Creating a new Action for setting the new Status.

In this step we create a new *Action* that will be used to trigger the setting of the new status. Actually, this Action will not contain an implementation with corresponding code but is rather a modelled Action that only serves as a trigger in the context of Status & Action Management. As this kind of Action cannot be created via the Enhancement Workbench, we use a workaround that nevertheless allows you adding the Action as an element to the Enhancement Object. Create the Action as follows:

- Start transaction `/BOBF/CONF_UI` and use the following path in the Business Object Browser to navigate to your Enhancement Object: *Transportable Business Objects* →

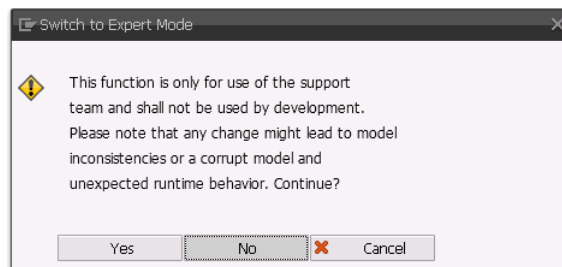
*Business Process Objects* → */SCMTMS/TRQ* → *Sub Business Objects* → (your Enhancement Object).

- Double click on your Enhancement Object which is now displayed in the Business Object Details Browser.
- Now the “trick” to allow adding elements to the Enhancement Object: In the command field enter *DEBUG* (and hit *ENTER*).



Picture: Switching the Enh. Object into Debug Mode.

- As you can see on the following popup, this debug mode is actually only intended for support purposes and should not be used in general. Nevertheless, it allows adding enhancement elements to the Enhancement Object that are not (yet) supported with the Enhancement Workbench directly. All enhancement elements added for this example will be assigned to the Enhancement Object, i.e. here we only use features that keep the standard BO modification free. But as stated in the popup warning message:

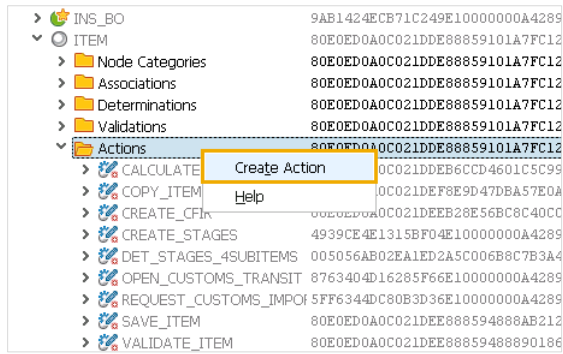


Picture: Warning before switching into Debug Mode.

**Note that changes are possible in this debug mode that can lead to inconsistencies, a corrupt model and unexpected runtime behavior. So use it with the required care.**

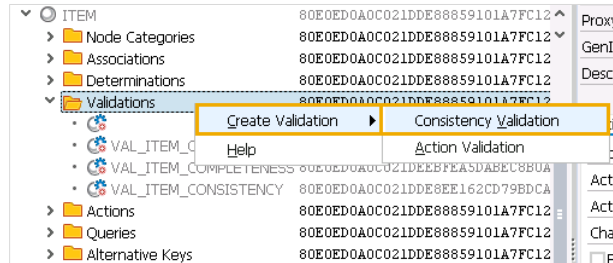
Click on button *Yes* to continue and switch into Debug Mode for the Enhancement Object. Moreover, switch into Change Mode (*Ctrl+F1*).

- Now create the intended Action by navigating along the following path in the Enhancement Object: *Node Elements* → *ITEM* → *Actions*. Right mouse click on *Actions* will bring up a popup menu. Choose option *Create Action*.



### 3) Creating a new Validation.

In the next step we create the Validation by navigating along the following path in the Enhancement Object: *Node Elements* → *ITEM* → *Validations*. Right mouse click on *Validations* will bring up a popup menu. Choose option *Create Validation* → *Consistency Validation*.



Picture: Creating a Validation.

- Create a Validation with the following attributes on the initial screen of the guided procedure. All other Action attributes are filled automatically in this use case:

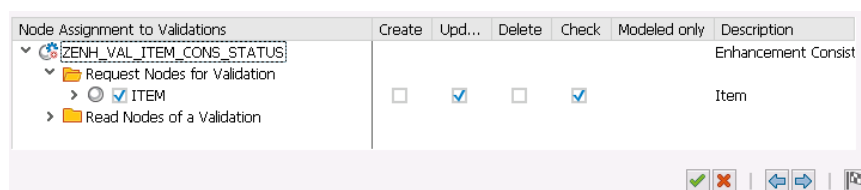
Attribute	Value/Content	Comment
Validation Name	ZENH_VAL_ITEM_CONS_STATUS	The name of the Validation.
Description	Enhancement Consistency Validation	Validation Description.
Node	ITEM	Node that the Validation is assigned to.
Validation Category	Consistency Check	The Validation represents a Consistency Check
Class/Interface	ZCL_ENH_V_ITEM_CONS	The implementing class for the Validation.
Change Mode	Exclusive Write Mode	Node Instances are locked exclusively in this case.
Action Can Be Enhanced	Yes	Action can be enhanced if required.

Picture: Guided Procedure for creating the Validation.

Navigate to the next screen of the guided procedure and define the following Request Nodes for the Validation:

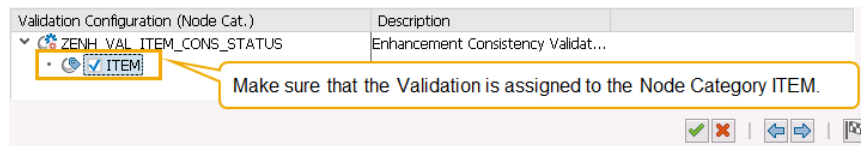
Node	Create	Update	Delete	Check
Item	No	Yes	No	Yes

With this setup, we have defined that the Item node triggers the Validation execution if the trigger condition *Create* or *Check* is fulfilled, i.e. it will be executed when Item node instances are updated/changed or the Item Node content is checked.



Picture: Node Assignment to Validation.

Navigate to the next screen of the guided procedure and assign the Validation to Node Category ITEM.



Picture: Assigning the Validation to Node Category ITEM.

- Click on the Finalize button to create the Validation with the given attributes. You can then find the new Validation in the Node Elements of the Item Node in this example.

Double click on your Validation to display its details. Make sure that you are in Change Mode. Then double click on the implementing class name to create this class and the implementation for the Validation. Some simple example coding for the *Execute* method of the Validation could look as follows:

```
METHOD /bobf/if_frw_validation~execute.
* Declarations
  FIELD-SYMBOLS: <fs_trq_items> TYPE /scmtms/s_trq_item_k.

  DATA: lt_trq_items TYPE /scmtms/t_trq_item_k,
        ls_failed_key TYPE /bobf/s_frw_key.

  IF sy-uname = 'DEMOUSER'.
    " Get the Item Data
    io_read->retrieve(
      EXPORTING
        iv_node      = /scmtms/if_trq_c=>sc_node-item
        it_key        = it_key
        iv_fill_data  = abap_true
      IMPORTING
        et_data       = lt_trq_items ).

    LOOP AT lt_trq_items ASSIGNING <fs_trq_items>.
      " If the Tare Weight Value is > 2000
      " then throw a failed key
      IF <fs_trq_items>-pkgun_wei_val > 2000.
        CLEAR ls_failed_key.
        ls_failed_key-key = <fs_trq_items>-key.
        APPEND ls_failed_key TO et_failed_key.
      ENDIF.
    ENDLOOP.
  ENDIF.

ENDMETHOD.
```

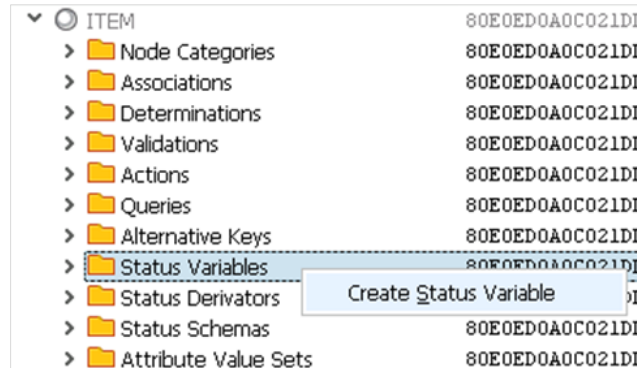
With this coding the Validation will check the value of the Tare Weight for a Forwarding Order Item whether it is larger than 2.000. If so it will return a Failed Key for such an Item instance and set its new status field from *01 Check Pending* (or *[blank]*) to *02 Check Inconsistent*. Otherwise the new status field will be set to *Check Consistent*.

- Save and activate the implementing class and then also the Enhancement Object.



#### 4) Creating a new Status Variable.

Create a Status Variable by navigating along the following path in the Enhancement Object: *Node Elements* → *ITEM* → *Status Variables*. Right mouse click on *Status Variables* will bring up a popup menu. Choose option *Create Status Variable*.



Picture: Creating a Status Variable.

- Create a Status Variable with the following attributes:

Attribute	Value/Content	Comment
Status Variable Name	ZENH_ITEM_CONS	The name of the Status Variable.
Description	Enhancement Item Cons. Status Var.	Status Variable Description.
Node	ITEM	Node that the Validation is assigned to.
Status Variable Category	Consistency Status (3 states)	Consistency status with 3 states (check pending, consistent and inconsistent).
Related Attribute Name	ZENH_CONS_STATUS	The attribute on the given Node that will hold the status value (in this example this is the extension field that we added in step 1).

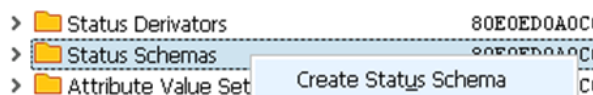
Status Variable Name	ZENH_ITEM_CONS
Proxy Var. Name	
Description	Enhancement Item Cons. Status Var.
<b>Status Variable Settings</b>	
Node	ITEM
Status Var. categ.	Consistency Status (3 states)
Related Attrib. Name	ZENH_CONS_STATUS

Picture: Specifying the Status Variable.

- Save and activate the Enhancement Object. You can then find the new Status Variable in the Node Elements of the Item Node in this example.

#### 5) Creating a new Status Schema.

Now create a Status Schema by navigating along the following path in the Enhancement Object: *Node Elements* → *ITEM* → *Status Schemas*. Right mouse click on *Status Schemas* will bring up a popup menu. Choose option *Create Status Schema*.



Picture: Creating a Status Variable.

- Create a Status Schema with the following attributes:

Attribute	Value/Content	Comment
Status Schema Name	ZENH_ISS	The name of the Status Schema (make sure it's not longer than 10 characters!).
Description	Enhancement Status Schema for Item Node.	Status Schema Description.
Node	ITEM	Node that the Validation is assigned to.

Picture: Specifying the Status Schema.

- Save and activate the Enhancement Object. You can then find the new Status Schema in the Node Elements of the Item Node in this example. The Status Schema requires further details to be configured. This is described in the next step.

#### 6) Specifying the details of the new Status Schema.

The details of the Status Schema can be specified and maintained in the View Cluster /BSAM/VC\_STM via transaction SM34.

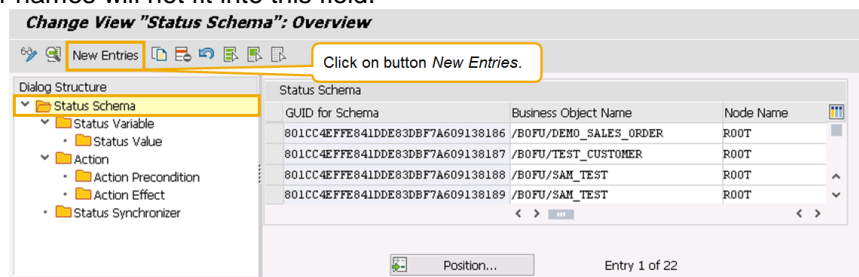
Picture: Entry Screen of transaction SM34.

- In the first step create an entry for the Status Schema in the View Cluster with the following table. But before we can create this entry, we need to determine the GUID of the Status Schema that was created in step 5. This GUID can be found as follows:
  - Start Transaction *SE16* and enter */BOBF/STA\_SCHEMA* as Table Name and hit *Enter* (this table contains the BOBF BO Meta Data for Status Schemas).
  - On the initial screen enter the name of your Enhancement Object (in this example *ZENH\_TRQ*) in field *NAME* and the name of the Status Schema created in step 5 (in this example *ZENH\_ISS*) in field *SCHEMA\_NAME* → press *F8*.
  - You should now see the entry for your Status Schema as represented on the database. Copy the GUID in field *STA\_SCHEMA\_KEY* and keep it for creating the first entry in the View Cluster */BSAM/VC\_STM*.

In the Dialog Structure of the View Cluster maintenance screen click on *Status Schema* and then click on button *New Entries*. Enter the following values in the related input fields and save the entry:

Attribute	Value/Content	Comment
Schema GUID	(GUID of the Schema created for the Enhancement Object → see remarks above)	The GUID of the Status Schema that was created and assigned to the Item Node via the Enhancement Object.
Node Name	ITEM	The node that the Status Schema is assigned to.
Status Schema	ZENH_ISS	The name of the Status Schema
Schema Use	B	= BOBF Status & Action Management Schema
Describing Text	Enhancement Status Schema	Description.

Remark: In step 5 it was mentioned to restrict the length of the Status Schema name to a maximum of 10 characters (see table with data for the Status Schema). The field for the Status Schema in this step 6 is (currently) restricted to 10 characters, i.e. longer names will not fit into this field.



Picture: Maintenance screen for View Cluster /BSAM/VC\_STM.

Schema GUID	0050563F05D11EE4B7F1B2261EA0CBEC
Status Schema	
BO Name	/SCMTMS/TRQ
Node Name	ITEM
Status Schema	ZENH_ISS
Schema Use	B
Describing Text	Enhancement Status Schema
Schema Use Desc	BOBF Same Adapter
Created By	
Created On	
Changed By	
Changed On	

Picture: Entry for Status Schema.

- In the Dialog Structure of the View Cluster maintenance screen click on *Status Variable* and then click on button *New Entries*. Enter the following values in the related input fields and save the entry:

Attribute	Value/Content	Comment
Status Variable	ZENH_ITEM_CONS	The name of the used Status Variable.
Describing Text	Enh. Item Cons. Status Var.	Description.

Status Value	Describing Text	Initial Status	Finalization Status
[blank]	Undefined	No	No
01	Check Pending	Yes	No
02	Check Inconsistent	No	No
03	Check Consistent	No	No

Picture: Entry for Status Variable.

- In the Dialog Structure of the View Cluster maintenance screen click on *Status Value* and then click on button *New Entries*. Enter the following values in the related input fields (in the given sequence) and save the entries:

Status Value	Describing Text	Initial Status	Finalization Status
[blank]	Undefined	No	No
01	Check Pending	Yes	No
02	Check Inconsistent	No	No
03	Check Consistent	No	No

Remark: Remember that a Status Variable with 3 states was chosen for this example but we actually maintained 4 Status Values in this step. In the table above the first entry represents a [blank] value. With this value present, the status change will also work for existing Item Node instances that did not have the new status field before. So we do not necessarily have to create e.g. a Determination (or even run a DB Table migration) that at the beginning sets the initial status to *01 Check Pending* for all instances of the Item Node. The value [blank] is so to say handled just like value *01 Check Pending*.

Click on button *Next Entry* (or *Previous Entry*) to display/enter the next Status Value.

Enter the data for each Status Value in this section.

Status Value	Describing Text	Initial Status	Finalization Status
01	Check pending	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Picture: Entries for Status Variables.

Finally you should see the entered values as shown in the following picture.

Status Value			
SVal	Describing Text	Initial St	Final. Sts
01	Check pending	<input type="checkbox"/>	<input type="checkbox"/>
02	Check inconsistent	<input checked="" type="checkbox"/>	<input type="checkbox"/>
03	Check consistent	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Picture: The final Status Values.

- In the Dialog Structure of the View Cluster maintenance screen click on *Action* and then click on button *New Entries*. Enter the following values in the related input fields and save the entry.

Attribute	Value/Content	Comment
Action	ZENH_SET_CONS_STATUS	The name of the used Action to trigger the setting of the status.
Describing Text	Enh. Action for setting status	Description.
Target Status Variable	ZENH_ITEM_CONS	The Status Variable that shall be influenced by the given Action.

**Dialog Structure**

- Status Schema
  - Status Variable
    - Status Value
  - Action
    - Action Precondition
    - Action Effect
  - Status Synchronizer

Schema GUID: 0050563F05D11EE4B7F1B2261EA0CBEC  
BO Name: /SCMTMS/TRQ  
Node Name: ITEM  
Status Schema: ZENH\_ISS  
Schema Description: Enhancement Status Schema  
Schema Use: B  
Schema Use Desc: BOPF Same Adapter  


---

Action: ZENH\_SET\_CONS\_STATUS  


---

Action:

Describing Text: Enh. Action for setting status  
Target Status Var: ZENH\_ITEM\_CONS  
Target Variable Desc: Enh. Item Cons. Status Var.

Picture: Entry for Action.

- In the Dialog Structure of the View Cluster maintenance screen click on *Action Precondition* and then click on button *New Entries*. Enter the following values in the related input fields (in the given sequence) and save the entries:

Source Status Variable	Status Value	Precondition Type
ZENH_ITEM_CONS	01	Enable
ZENH_ITEM_CONS	02	Enable
ZENH_ITEM_CONS	03	Enable

**New Entries: Details of Added Entries**

Click on button *Next Entry* (or *Previous Entry*) to display/enter the next Action Precondition.

Enter the data for each Action Precondition in this section.

Dialog Structure	Schema GUID
Status Schema	Action ZENH_SET_CONS_STATUS
Status Variable	BO Name /SCMTMS/TRQ
Status Value	Node Name ITEM
Action	Status Schema ZENH_ISS
Action Precondition	Schema Use B
Action Effect	Schema Description Enhancement Status Schema
Status Synchronizer	Action Description Enh. Action for setting status
	Target Status Var ZENH_ITEM_CONS
	Schema Use Desc BOPF Same Adapter
	Source Status Var ZENH_ITEM_CONS
	Status Value 03
	Action Precondition
	Source Variable Desc Enh. Item Cons. Status Var.
	Value Description Check consistent
	Precondition Type Enable

Picture: Entries for Action Precondition.

- In the Dialog Structure of the View Cluster maintenance screen click on *Action Effect* and then click on button *New Entries*. Enter the following values in the related input fields (in the given sequence) and save the entries:

Status Value
01
02
03

**New Entries: Details of Added Entries**

Click on button *Next Entry* (or *Previous Entry*) to display/enter the next Action Effect.

Enter the data for each Action Effect in this section.

Dialog Structure	Schema GUID
Status Schema	Action ZENH_SET_CONS_STATUS
Status Variable	BO Name /SCMTMS/TRQ
Status Value	Node Name ITEM
Action	Status Schema ZENH_ISS
Action Precondition	Schema Use B
Action Effect	Schema Description Enhancement Status Schema
Status Synchronizer	Action Description Enh. Action for setting status
	Target Status Var ZENH_ITEM_CONS
	Describing Text BOPF Same Adapter
	Status Value 01
	Action Effect
	Value Description Check pending

Picture: Entries for Effect.

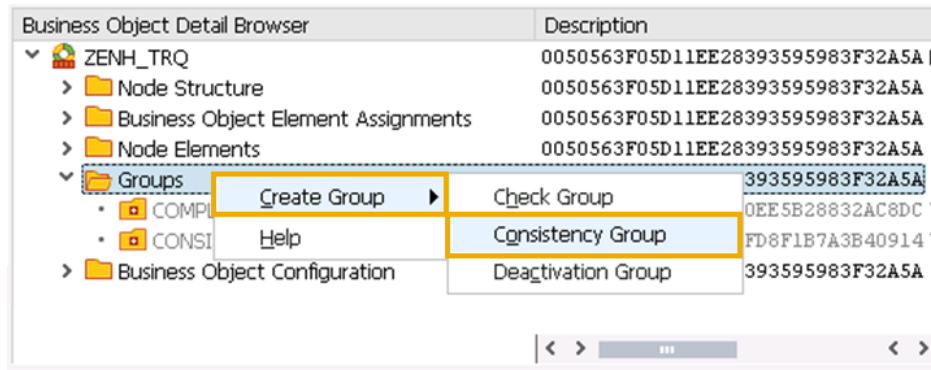
- Assigning the new Status Schema to the relevant Node Category.

This step is mentioned here explicitly as it actually would require the modification of a BO Meta Data Table which contains the information about the Node Categories. In the example, the new Status Schema was assigned to the Item Node of BO /SCMTMS/TRQ. In standard SAP TM this node has only a single Node Category *ITEM*. To prevent any modification for assigning the new Status Schema to this Node Category, make sure that [note 2145188](#) is implemented in your SAP TM System. This note contains an adjustment that works as follows: If the Node only has one Node Category and also has only one Status Schema assigned, then this Status Schema will be used as the default Schema at runtime.

## 8) Creating a new Consistency Group.

Navigate back to the Enhancement Object as described in step 2, switch on the Debug Mode. Moreover, switch into Change Mode (*Ctrl+F1*).

Create a Consistency Group by a Right mouse click on Groups in the Business Object Details Browser. In the popup menu choose option *Create Group* → *Consistency Group*.



Picture: Creating a Consistency Group.

- On the first screen of the guided procedure enter the following data to specify the new Consistency Group:

Attribute	Value/Content	Comment
Group	ZENH_CONS_GROUP	The name of the Consistency Group.
Description	Enhancement Consistency Group	Description.
Group Category	Consistency Group	A Consistency Group can be used to set a Consistency Status depending of the result of validations assigned to this Group.
Node	ITEM	The BO Node that the Group is assigned to.
Action	ZENH_SET_CONS_STATUS	The Action that triggers the Status Change.
Status Variable	ZENH_ITEM_CONS	The Status Variable that represents the new status in this example.

- Navigate to the next screen of the guided procedure and assign the Validation created in step 3 to the new Consistency Group. Navigate along the following path: *ZENH\_CONS\_GROUP* → *Validations*.

You can find the Validation *ZENH\_VAL\_ITEM\_CONS\_STATUS* in the list of Validations. Make sure that this new Validation is assigned to the Consistency Group by setting the flag in front of the Validation name.

- Click on the Finalize button to create the Consistency Group with the given attributes. You can then find the new Consistency Group in the Groups section of the Enhancement Object in this example.
- Save and activate the Enhancement Object.

## 9) Test the example.

- Open an existing Forwarding Order, change the Tare Weight value for an item of this Forwarding Order from 1.700 to e.g. 2.500 and hit *Enter*. First of all the status is set to status value *01 Check Pending* in this example.



The screenshot shows the 'Details: CONTAINER 10' screen with the 'Container Details' tab selected. The 'General Data' section on the left includes fields for Item Type, Description, Container from Forwarding, Resource, Equipment Group/Type from..., Container from Freight Do..., Equipment Group/Type from..., Shipper-Owned Container, and Length/Width/Height. The 'Ordered Quantities' section on the right shows Quantity (1), Gross Weight (14.450), Gross Volume, Net Weight (12.750), Tare Weight (1.700), and Number of TEU (1). A callout box points to the 'Demo Enhancement Group for Status Change Demo' field, which is set to '01'. Another callout box points to the 'Tare Weight' field, stating 'The Tare Weight of a selected container item with 1.700 KG.'

Picture: Status 01 Check Pending.

Remark: In the picture above you can see the new status value displayed which is not part of the standard User Interface. Once you have worked through section 5 on User Interface Enhancements, it would be a nice exercise for you to customize your User Interface the same way to make it display the additional Item Status on the Item Details screen of the Forwarding Order UI as shown above.

- Now click on button *Save* to save the Forwarding Order and check the status of the selected item again. The status value now has changed from *01 Check Pending* to *02 Check Inconsistent*.

The screenshot shows the 'Details: CONTAINER 10' screen with the 'Container Details' tab selected. The 'General Data' section on the left includes fields for Item Type, Description, Container from Forwarding, Resource, Equipment Group/Type from..., Container from Freight Do..., Equipment Group/Type from..., Shipper-Owned Container, and Length/Width/Height. The 'Ordered Quantities' section on the right shows Quantity (1), Gross Weight (15.250), Gross Volume, Net Weight (12.750), Tare Weight (2.500), and Number of TEU (1). A callout box points to the 'Demo Enhancement Group for Status Change Demo' field, which is set to '02'. Another callout box points to the 'Tare Weight' field, stating 'The adjusted Tare Weight of the selected container item with 2.500 KG.'

Picture: Status 02 Check Inconsistent.

- Adjust the Tare Weight value from 2.500 to e.g. 1.500, save the Forwarding Order again and check the status of the selected item. This time the status value has changed from *02 Check Inconsistent* to *03 Check Consistent*.

The screenshot shows the 'Details: CONTAINER 10' screen with the 'Container Details' tab selected. The 'General Data' section on the left includes fields for Item Type, Description, Container from Forwarding, Resource, Equipment Group/Type from..., Container from Freight Do..., Equipment Group/Type from..., Shipper-Owned Container, and Length/Width/Height. The 'Ordered Quantities' section on the right shows Quantity (1), Gross Weight (14.250), Gross Volume, Net Weight (12.750), Tare Weight (1.500), and Number of TEU (1). A callout box points to the 'Demo Enhancement Group for Status Change Demo' field, which is set to '03'. Another callout box points to the 'Tare Weight' field, stating 'The adjusted Tare Weight of the selected container item with 1.500 KG.'

Picture: Status 03 Check Consistent.

## Consistency Group for preventing Save

The first use case makes use of a Consistency Group and the other involved elements to set a status field on the Forwarding Order Item Node depending on a Validation result. This behavior of the Consistency Group was achieved by assigning it a Node, an Action and a Status Variable (see step 8 of the first use case).

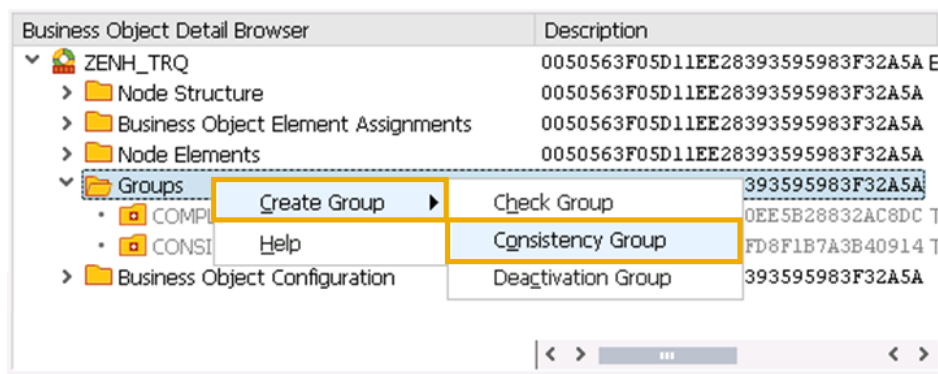
The second use case is based on the first. In this case not only the status shall be set but in case of the Validation returning Failed Keys (i.e. errors or inconsistencies) the Save of the transaction shall be prevented. It is a quite common use case that a transaction shall not be Save-enabled in case of errors or inconsistencies.

Just like for the first use case the following steps describe how to setup and implement an example. Access the required Enhancement Object as described in step 2 of the first example, i.e. in Debug Mode and Change Mode.

### 1) Creating a new Consistency Group.

Navigate to the Enhancement Object and switch on the Debug Mode. Moreover, switch into Change Mode (*Ctrl+F1*).

Create a Consistency Group by a Right mouse click on Groups in the Business Object Details Browser. In the popup menu choose option *Create Group* → *Consistency Group*.



Picture: Creating a Consistency Group.

- On the first screen of the guided procedure enter the following data to specify the new Consistency Group:

Attribute	Value/Content	Comment
Group	ZENH_CONS_GROUP_SAVE	The name of the Consistency Group.
Description	Enh. Cons. Group to prevent Save	Description.
Group Category	Consistency Group	A Consistency Group can be used to set a Consistency Status depending of the result of validations assigned to this Group.
Node	[blank → nothing chosen]	No specific BO node in this use case
Action	[blank → nothing chosen]	No specific Action in this use case.
Status Variable	[blank → nothing chosen]	No Status Variable in this use case as this Consistency Group shall serve as a means to prevent Save.

Picture: Consistency Group for preventing Save.

- Navigate to the next screen of the guided procedure and assign the Validation created in step 3 of the first use case to the new Consistency Group. Navigate along the following path: **ZENH\_CONS\_GROUP** → **Validations**.

You can find the Validation **ZENH\_VAL\_ITEM\_CONS\_STATUS** in the list of Validations. Make sure that this Validation is assigned to the Consistency Group by setting the flag in front of the Validation name.

Picture: Assigning Validations to the Consistency Group.

- Click on the Finalize button to create the Consistency Group with the given attributes. You can then find the new Consistency Group in the Groups section of the Enhancement Object in this example.
- Save and activate the Enhancement Object.

## 2) Creating a new Consistency Group.

- Open an existing Forwarding Order, change the Tare Weight value for an item of this Forwarding Order from e.g. 2.500 and hit **Enter**. First of all the status is set to status value **01 Check Pending** in this example.
- Now click on button **Save** to save the Forwarding Order. With the first use case implemented, the status value now has changed from **01 Check Pending** to **02 Check Inconsistent**. Moreover, the **Save** is now prevented as you can see in the message log of the Forwarding Order User Interface.

Picture: The status is set to Check Inconsistent and Save is prevented.

### 3.4.5 Change Document Adapter Enhancements

For many types of Business Documents used in SAP TM it is possible or even might be required to track changes. This allows finding out what has changed (e.g. an attribute from value A to value B) and when.

In the Document Types (Customizing) for SAP TM Business Documents like Forwarding Order, Freight Order and others you can define that changes shall be tracked.

Picture: Example Forwarding Order Type with activated tracking of Document Changes.

For the purpose of logging changes to a Business Object, this object is defined as a so called Change Document Object (CDO). Such a CDO contains the information about the Database Tables that hold the data for the Business Object. Changes to a Business Object are then saved via the related CDO. In standard, the following Change Document Objects are available (delivered with SAP TM 9.3):

Change Document Object (CDO)	BOPF BO	Description
/SCMTMS/CD_CFIR	/SCMTMS/CUSTFREIGHTINVREQ	Forwarding Settlement Document
/SCMTMS/CD_FAG	/SCMTMS/FREIGHTAGREEMENT	Freight Agreement
/SCMTMS/CD_RATE	/SCMTMS/TC_RATES	Transp. Charges Rates
/SCMTMS/CD_SCAL	/SCMTMS/TC_SCALE	Transp. Charges Scales
/SCMTMS/CD_SFIR	/SCMTMS/SUPPFREIGHTINVREQ	Freight Settlement Document
/SCMTMS/CD_TARF	/SCMTMS/TC_TARIFF	Transp. Charges Tariff
/SCMTMS/CD_TCCS	/SCMTMS/TCC_TRNSP_CHRG	Transp. Charges Calculation Sheet
/SCMTMS/CD_TOR	/SCMTMS/TOR	Freight Oder / Freight Unit / Booking
/SCMTMS/CD_TRQ	/SCMTMS/TRQ	Forwarding Order
/SCMTMS/CD_TTMP	/SCMTMS/TENDERINGTEMPLATE	Tendering Profile

The Change Document Object mentioned in this table can be seen in transaction SCDO\_NEW. The following example is again a real customer enhancement use case. You will see how to enable tracking changes for data that is stored on a customer-specific Enhancement Node for the Forwarding Order BO (/SCMTMS/TRQ) (Remark: If you are not yet familiar with User Interface Enhancements, you should consider working through section 5 first before continuing with this enhancement use case).

Visit <http://help.sap.com> to navigate to the detailed documentation for Change Documents, their setup and concepts. You can find this documentation under the BC Extended Application Function Library.

Remark: You should – as always – keep in mind the performance of your system / application, i.e. with the Change Document functionality active you add a certain additional amount of runtime and memory consumption of course. So, Change Documents should be only considered where really required. They might e.g. help finding errors in certain processes but should be switched off again after having solved the root causes of these errors if they are not actually needed for any other purposes.

- 1) Create a new Subnode for the Forwarding Order (/SCMTMS/TRQ) as described in section 3.3.5. If not yet available, you first need to create a corresponding Enhancement Object for standard BO /SCMTMS/TRQ that you can then assign the new Subnode to. Use the following details in the Wizard for creating Subnodes in the Enhancement Workbench:

Attribute	Value/Content	Comment
Node Name	ZENH_TRQRT_SUB	The name of the new Subnode.
Description	Demo Enh. TRQ Root Subnode	Description.
Node is extensible	Yes	It shall be allowed to enhance the new Subnode.
Persistent Extension Include	ZENH_INCL_P_TRQRTSUB	Just specify the required Dummy field in this structure.
Persistent Structure	ZENH_S_TRQRT_SUB_D	Name of the persistent structure. The following table shows the attributes to be placed there.
Transient Structure	ZENH_S_TRQRT_SUB_DT	Name of the transient structure. The following table shows the attributes to be placed there.

For the Persistent Structure define the following attributes:

Component	Typing Method	Component Type
ZENH_CHNG_DATE	Types	/SCMTMS/DATETIME
ZENH_CHNG_USER	Types	/SCMTMS/USER_ID_CH
ZENH_CHNG_REAS	Types	/SCMTMS/DESCRIPTION
.INCLUDE	Types	ZENH_INCL_P_TRQRTSUB

For the Transient Structure define the following attributes:

Component	Typing Method	Component Type
ZENH_TRAN_DATE	Types	/SCMTMS/DATETIME
.INCLUDE	Types	ZENH_INCL_T_TRQRTSUB

For the Combined Structure, the Combined Table Type and the Database Table use the following names:

Attribute	Value/Content	Comment
Combined Structure	ZENH_S_TRQRT_SUB	The name of the Combined Structure.
Combined Table Type	ZENH_T_TRQRT_SUB	The name of the Combined Table Type.
Database Table	ZENH_D_TRQRTS	The name of the Database Table that will hold the data of the new Subnode.

Finish the Wizard for creating the Subnode with these details by clicking on button *Complete* and continue with step 2.

- 2) In section 5.4.3 you can find a step-by-step example how to add a new tab strip to the User Interface to represent the data of our new Subnode. A few things are of course different here compared to the example in section 5.4.3.



- Similar to the steps described in section 5.4.3 create a new Component Configuration for a List UIBB with the details in the table below. Note that this Component Configuration is a new object with the customer/partner as the owner (it's not customizing):

Attribute	Value	Comment
Component Name	FPM_LIST_UIBB_ATS	The new configuration shall represent a list which will contain data from the new BO subnode.
Configuration ID	ZENH_WDCC_TRQSUBNODE_LIST	This will be the new configuration to be integrated in the Forwarding Order UI.

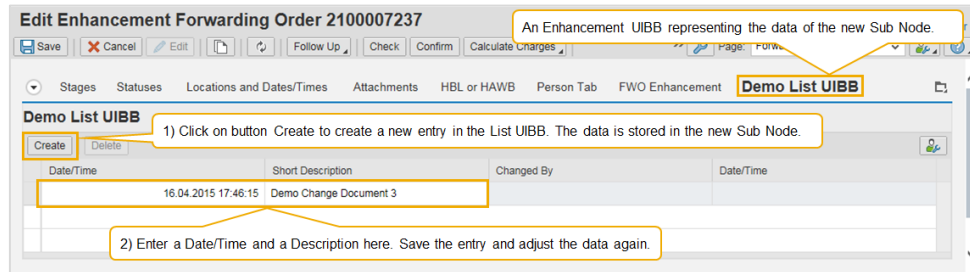
- In step 4, use Feeder Class /BOFU/CL\_FBI\_GUIBB\_LIST\_ATS for the new List UIBB configuration.
- In step 5, define the following Feeder Class Parameters for the new List UIBB configuration: As Business Object choose /SCMTMS/TRQ and as Node choose the new subnode ZENH\_TRQRT\_SUB that was created before.
- In step 6, place all the attributes of the subnode on the List UIBB and define the attributes ZENH\_CHNG\_DATE and ZENH\_CHNG\_REAS as input fields while keeping the others output only fields. Moreover, add the standard buttons Create and Delete to the toolbar of the List UIBB.
- Save the new Component Configuration for the List UIBB
- Add the new List UIBB to the Forwarding Order UI. To do so, navigate to the Component Customizing of Component /SCMTMS/WDCC\_FWD\_ORDER as described in step 8. Note that adding your List UIBB to the standard Forwarding Order UI and wiring it with this UI is subject to customizing that you create with this step.

Add your List UIBB to the Main Page, SECTION\_1 of the Forwarding Order UI (component /SCMTMS/WDCC\_FWD\_ORDER).

- As described in steps 9 - 11, create a Wire between the initial screen of the Forwarding Order UI and your new Component Configuration for the List UIBB with the following details (Again: This Wire is subject to customizing the standard Forwarding Order UI)

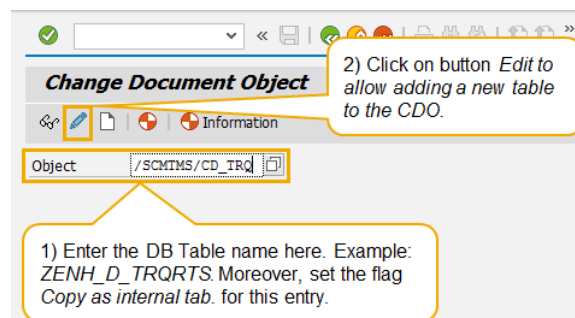
Attribute	Value	Comment
Component	FPM_LIST_UIBB_ATS	The generic List UIBB provided by FPM, i.e. the target component of the wire.
Configuration ID	ZENH_WDCC_TRQSUBNODE_LIST	The example configuration for the new List UIBB, i.e. the target configuration of the wire.
Instance ID		
Source Component	FPM_FORM_UIBB_GL2	The source component of the wire. In this example it is a Form UIBB.
Source Configuration Name	/SCMTMS/WDCC_TRQ_INITSCREEN	The source configuration of the wire. In this example, it is the configuration for the initial screen form of the Forwarding Order UI.
Source Node Association	ZENH_TRQRT_SUB	The association that is defined between the wire source and target node. In this example: The composition association between the Forwarding Order (TRQ) Root node and our new subnode ZENH_TRQRT_SUB.
Port Type	Collection	
Port Identifier	CO	
Connector Class	/BOFU/CL_FBI_CONNECTOR	Provides basic functions to connect FPM, FBI and BOBF.

- Save your Component Configuration Customizing and test the new UI Enhancement. The following picture shows the result of the described UI Enhancement steps. With this UI in place we can now create changes to the data stored in the new subnode. You will later on see how these changes get recorded in the context of a Change Document Object.



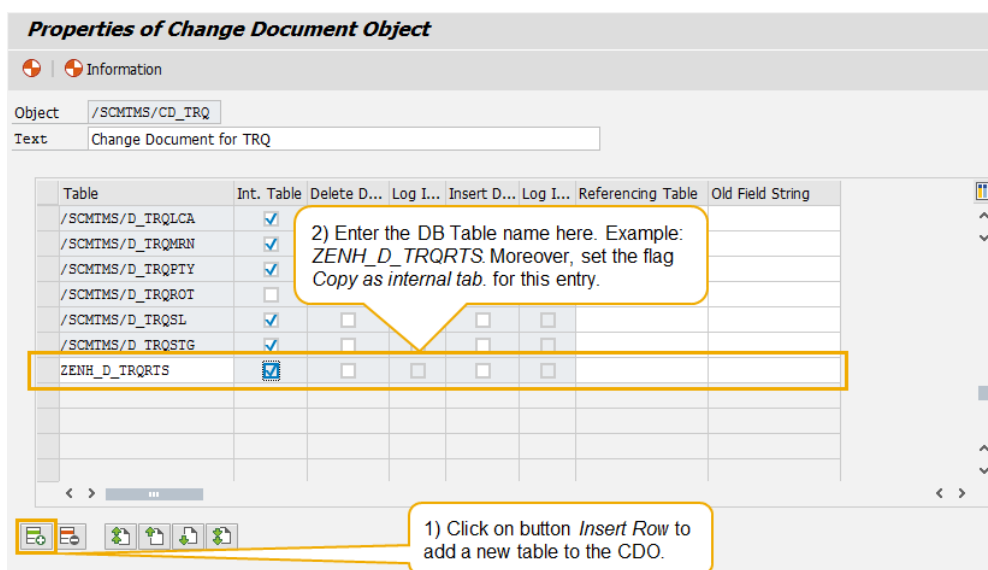
Picture: The new Subnode represented on the Forwarding Order UI.

- 3) Start transaction `SCDO_NEW` to search for and display the Change Document Object `/SCMTMS/TRQ`.



Picture: Searching for Change Document Objects.

On the following screen you can then see the list of Database Tables that are assigned to this CDO and the properties for each table that determine different aspects of the behavior in case of changes. Changes to the content of the assigned tables will be recorded and saved in the context of this Change Document Object.



Picture: Database Tables assigned to a Change Document Object



- 4) Click on button *Insert Row* to add the Database Table that represents the data of the subnode added in step 1. Add the name *ZENH\_D\_TRQRTS* in column *Table* and set the flag *Int. Table* for the same entry. Finally click on button *Save* (*Ctrl. + S*).
- 5) In the next step you need to (re-) generate the update program for the Change Document Object. To do this, click on button *Generate change document object* (*F5*). This will first of all bring up a popup that allows specifying a few generation parameters. Finally, click on button *Generate*.

Picture: Parameters for generating the update program for the CDO.

After the generation of the update program for the Change Document Object, the system displays a summary of the generation process (what has been changed and what remained unchanged). When displaying a CDO you can always display the latest Generation Information by clicking on button *Generation information* (*F6*) (see picture in step 3). A part of the Generation Information for this example is shown in the following picture:

**Change Document Object: Generation Information**

To navigate directly to an object, double-click a line

Example: Here you can see the entry in the Generation Information that was created for the additional Database Table that was added to this CDO.

Object	Report Name	Area	Description	Value
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Dictionary Structures	Structure	/SCMTMS/TRQD_TRQSL
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Structure	Structure	/SCMTMS/TRQD_TRQSTG
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Structure	Structure	/SCMTMS/TRQZENH_D_TRQRTS
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	Date	16.04.2015
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	Time	17:46:21
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	User	POLCH
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	Package	/SCMTMS/TRQ
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	Source System	C2B
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	Prefix for DDIC Objects	/SCMTMS/TRQ
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	Application Area	CD
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	Error Number	600
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	Special Text Handling	Active
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	DATA Generation for ABAP OO	Inactive
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Generation	Update Flag from TCDRP	1
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Include Programs	TOP (Call Data Declarations)	/SCMTMS/FCD_TRQCDT
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Include Programs	Globally Valid Data Declarations	/SCMTMS/FCD_TRQCDF
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Include Programs	Object-Specific Data Declarations	/SCMTMS/FCD_TRQCDV
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Include Programs	Call Update Function Module	/SCMTMS/FCD_TRQCDC
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Update Function Module	Package	/SCMTMS/TRQ
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Update Function Module	Name	/SCMTMS/CD_TRQ_WRITE_DOCUMENT
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Update Function Module	Function Group	/SCMTMS/TRQ_CD
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Update Function Module	Package	/SCMTMS/TRQ
/SCMTMS/CD_TRQ	/SCMTMS/CD_TRQ	Update Function Module	Source System	C2B

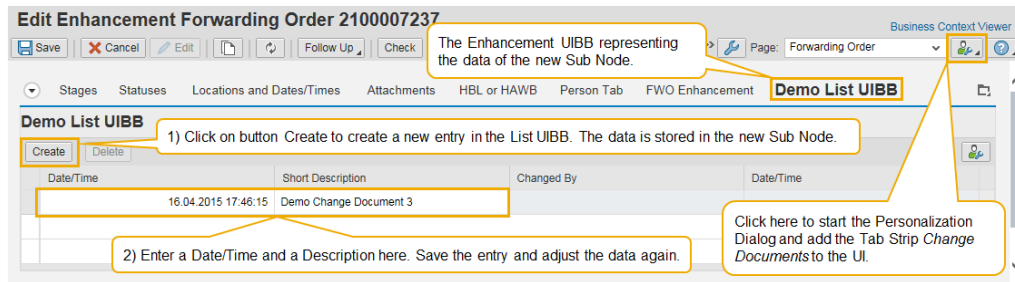
General data of the last generation.

The Includes of the update program.

The used Update Function Module.

Picture: CDO Generation Information after the generation process.

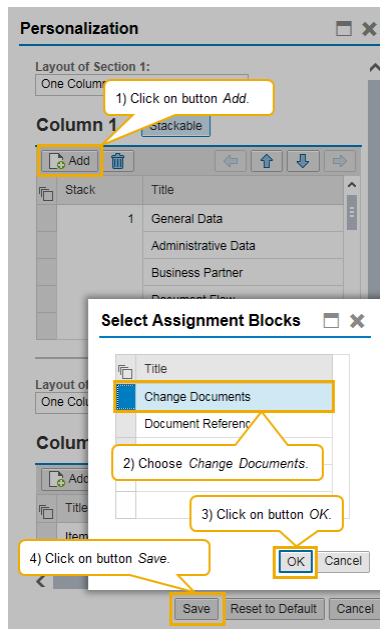
- 6) Now you can test this enhancement use case. Open an existing Forwarding Order, and navigate to the new Tab Strip that was added to allow displaying and creating data for the new Enhancement Node.



Picture: Entering data for the new subnode on the enhanced UI.

- Click on button *Create* in the toolbar of the List UIBB to create a new entry. Enter a Date/Time and a description in the open input fields.
  - Save the document.
  - Click on button *Edit* to switch back into Edit Mode. Change the data that you have created before and save the document again, i.e. we “enforce” a few changes that are recorded in the context of CDO /SCMTMS/CD\_TRQ.
- 7) Display the created Change Documents on the Forwarding Order UI.

If you do not see the Tab Strip *Change Documents* in the standard UI, first of all make it visible by clicking on button *Personalize* in the header toolbar of the UI. On the popup execute the following steps: Click on button *Add* and select Assignment Block *Change Documents* from the list. Now click on button *OK* in the list and finally on button *Save*.



Picture: Adding the *Change Document* Tab Strip to the UI.

- 8) The Change Documents incl. the ones for the new Enhancement Node displayed on the Forwarding Order UI looks as follows:

Change	Node	ID	Node Description	Change Indicator	Short Description	Field Name	New Value	Old Value	Change	Chan	Time	Transaction Code
2056420	ROOT_TRANSPO...	Total Sum	ChargeElement	Insert (I)	KEY	KEY			POLCH	15.04...	11:11:57	
2056420	ROOT_TRANSPO...	Charge Root	Root	Insert (I)	KEY	KEY			POLCH	15.04...	11:11:57	
2056420	ITEM	10	Item	Insert (I)	KEY	KEY			POLCH	15.04...	11:11:57	
2056420	ROOT	2100007237	Root	Insert (I)	KEY	KEY			POLCH	15.04...	11:11:57	
2056420	STAGE	10	Stage	Insert (I)	KEY	KEY			POLCH	15.04...	11:11:57	
2056836	ZENH_TRQRT_SUB	005056AC01921ED...		Insert (I)	DateTime	ZENH_CHNG_DATE	16.04.2015 16:12:00		POLCH	16.04...	16:12:25	
2056850	ZENH_TRQRT_SUB	005056AC01921ED...		Update (U)	DateTime	ZENH_CHNG_DATE	16.04.2015 16:12:15	16.04.2015 16:12:00	POLCH	16.04...	17:44:15	
2056854	ZENH_TRQRT_SUB	005056AC01921ED...		Update (U)	DateTime	ZENH_CHNG_DATE	16.04.2015 17:46:15	16.04.2015 16:12:15	POLCH	16.04...	17:47:06	
2056858	ZENH_TRQRT_SUB	005056AC01921ED...		Update (U)	DateTime	ZENH_CHNG_DATE	24.04.2015 17:46:15	16.04.2015 17:46:15	POLCH	24.04...	15:17:31	

Picture: The Change Documents displayed on the UI

On the Tab Strip Change Documents you can now see all available changes that have been made to a Forwarding Order, not only for data stored in standard Database Tables but also for Database Tables of Enhancement Nodes. You can see there the node name (e.g. `ZENH_TRQRT_SUB`), the Change Document Number, the name of the changed field, old and new value, date and time of the change, etc.

Some further remarks:

As mentioned at the beginning of this example for enhancing an existing Change Document Object, this is a real customer enhancement use case. But you should keep in mind that the CDOs delivered with the standard are SAP objects, i.e. depending on the setup of your system, it might be not allowed to change that kind of objects as you change an SAP object.

Alternatively, you could create your very own customer- or partner-specific Change Document Object for a Standard Business Object and assign it all the Database Tables that the standard CDO has assigned plus the Database Tables for Enhancement nodes.

But how does the system (better: The BOPF BO) then know which CDO to use at runtime? Well, that is a matter of customizing. In the IMG (transaction `SPRO`) use the following path: *Cross-Application Components → Reusable Objects and Functions for BOPF Environment*.

Business Object	Change doc. object	Change Document Class
/SCMTMS/TCCS	/SCMTMS/CD_TCCS	/SCMTMS/CL_TCCS_CDO_CB
/SCMTMS/TCC_TRNSP_CHRG	/SCMTMS/CD_TCC	/SCMTMS/CL_TCC_CDO_CB
/SCMTMS/TC_RATES	/SCMTMS/CD_RATE	/SCMTMS/CL_RATE_CDO_CB
/SCMTMS/TC_SCALE	/SCMTMS/CD_SCAL	/SCMTMS/CL_SCALE_CDO_CB
/SCMTMS/TENDERINGTEMPLATE	/SCMTMS/CD_TTMP	/SCMTMS/CL_TTEMP_CDO_CB
/SCMTMS/TOR	/SCMTMS/CD_TOR	/SCMTMS/CL_TOR_CHANGELOG CA
/SCMTMS/TRQ	/SCMTMS/CD_TRQ	/SCMTMS/CL_TRQ_CDO_CB
DATA REPLICATION FILTER	DRF_FILTER_CD	CL_DRF_FILTER_CDO_CALLBACK
ZK_OPEN_ACCESS		/BOFU/CL_CDO_DEMO_CUSTOMER
ZIPTCL_PAACT		

Picture: Assignment of BOPF BO to CDO.

Here you can see the assignment of e.g. the technical BOPF BO `/SCMTMS/TRQ` to the Change Document Object `/SCMTMS/CD_TRQ` (which we have enhanced in the example above) and the Change Document Class `/SCMTMS/CL_TRQ_CDO_CB`. The mentioned class is a good example for implementing your very own implementing class for your own CDO. It always should inherit from the super class `/SCMTMS/CL_GEN_CDO_CB`.

Again it might be not allowed to change that kind of objects in your system as you change an SAP object.

## 4 Techniques for Enhancing the Business Logic

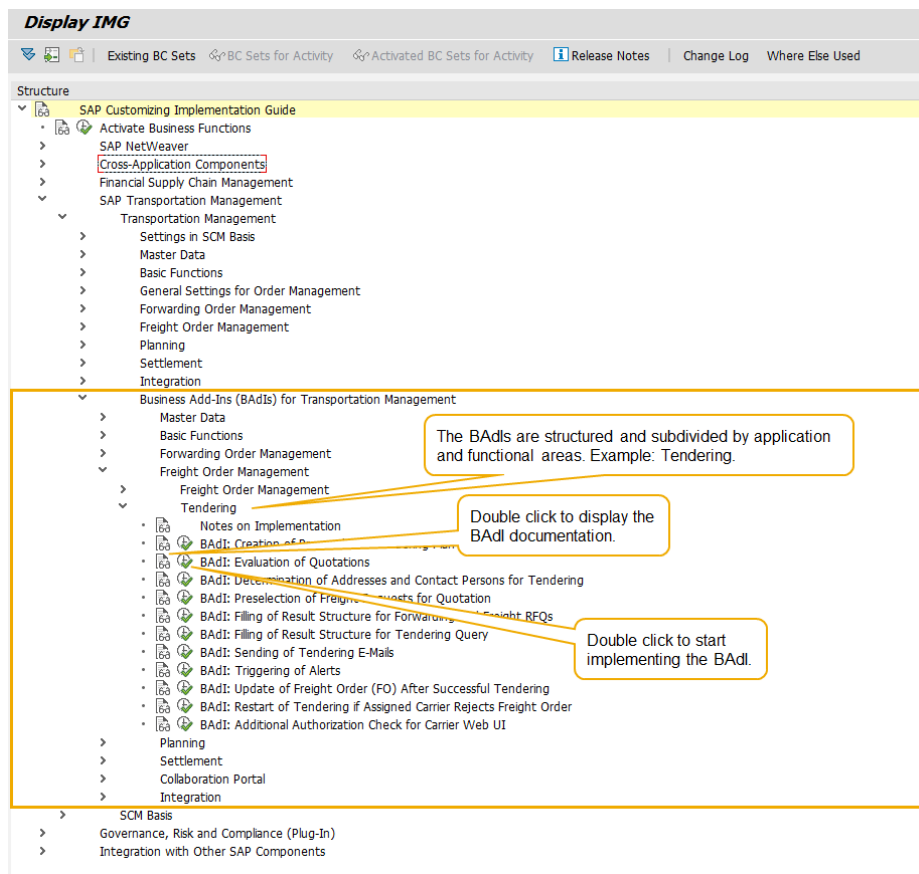
### 4.1 BAdIs

The Business Add-In (BAI) concept is SAP's object-oriented plug-in concept for ABAP. BAdIs are a mechanism for planned extensibility. Planned means that the developer of the standard software already anticipates that others may want to change or enhance the standard behavior at certain points in the application. BAdIs are used to plug in custom behavior either in an additive way or by replacing the standard behavior.

#### 4.1.1 Where and how to find BAdIs related to TM

In the meantime, in SAP TM more than 170 BAdIs are available in all application areas. They can be found in the IMG (transaction *SPRO*) under the following path:

*SAP Transportation Management* → *Transportation Management* → *Business Add-Ins (BAdIs) for Transportation Management*.



Picture: TM BAdIs in the IMG.

Alternatively, you can use transaction *SE18* and use the F4-Help in field *BAdI Name* to search for BAdIs with name */SCMTMS/\**.

#### 4.1.2 Implementing a BAdI

A BAdI implementation can be started directly from the IMG. As an alternative, transaction *SE19* can be used to either edit existing enhancement implementations or create new ones. Besides the initial screen, the other steps to implement a BAdI with *SE19* are the same as the procedure starting from the IMG.

Picture: Defining the required Enhancement Implementation.

When starting from the IMG, just click on the second icon next to the BAdI name (see picture in section 4.1.1). This allows also the navigation to existing implementations of the corresponding BAdI.

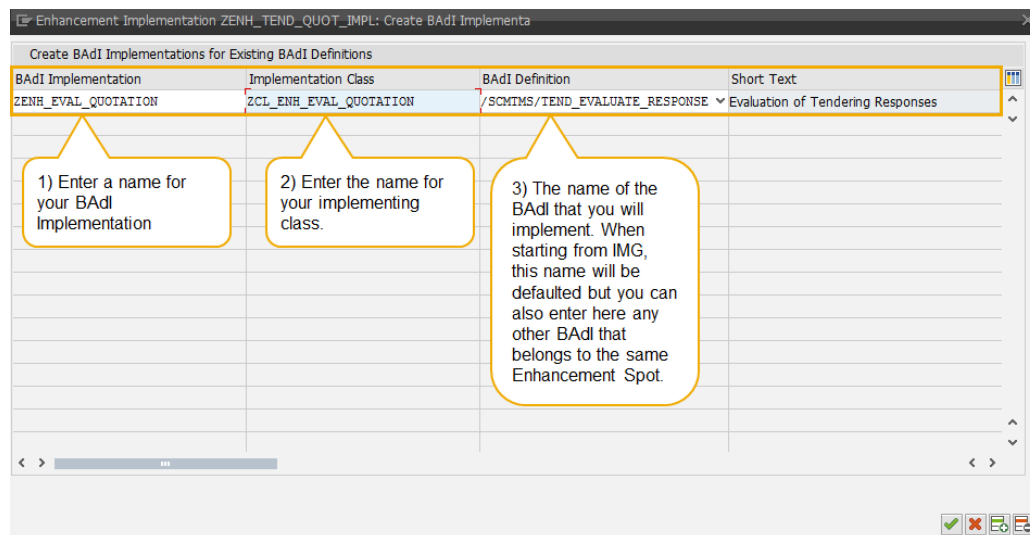
- 1) When starting a BAdI Implementation directly from the IMG, the first step is to specify an Enhancement Implementation and a Short Text for it. This Enhancement Implementation serves as a container for your implementation steps done in the following.

Picture: Defining the required Enhancement Implementation.

- 2) On the next popup, choose a package where you store your implementation and click on the Save button.

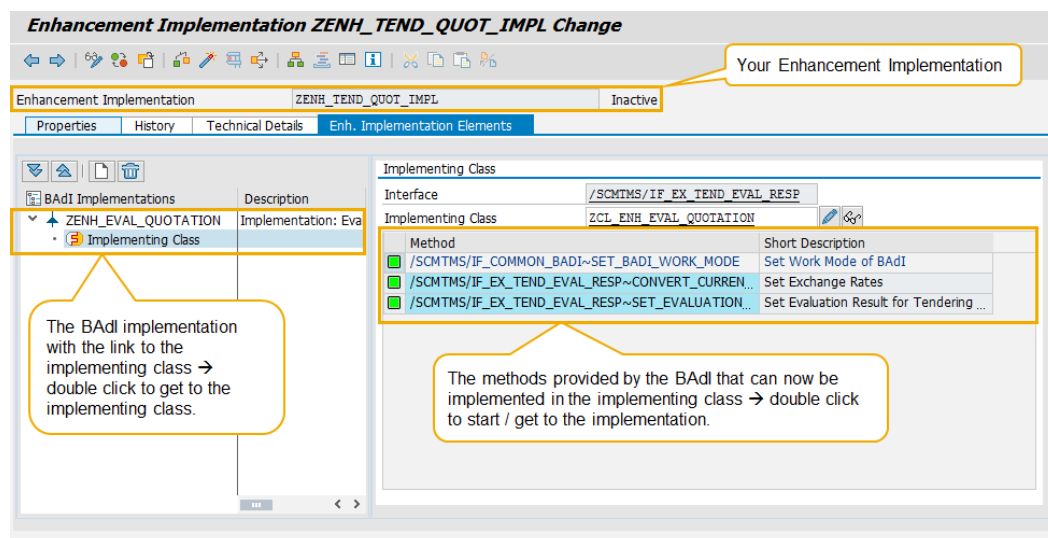
Picture: Defining the required Enhancement Implementation.

On the following screen you need to enter a BAdI Implementation and an Implementation Class. If you started your implementation from the IMG, the correct BAdI Definition is already defaulted. Optionally you can also choose another or additional BAdI Definitions here that belong to the same Enhancement Spot. In this example you can choose from the group of BAdIs that belong to the Enhancement Spot /SCMTMS/TEND which contains all Tendering related BAdIs.



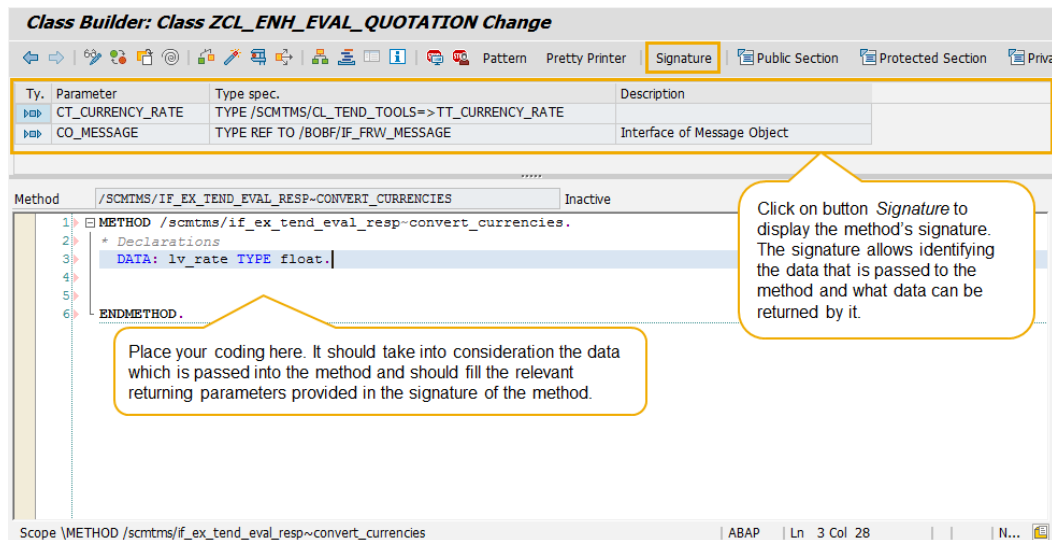
Picture: Specifying a BAdI Implementation and the implementing Class.

- 3) A BAdI can provide one or more methods that serve different purposes. So the next step is to navigate to the implementing class to get the required methods implemented.



Picture: Navigating to the implementing class of a BAdI.

- 4) Finally create an implementation for desired BAdI methods. Within this step it is helpful to display the signature of the method to see what data it receives and what data it is able to return as a result. The displayed signature allows direct navigation to the DDIC objects used for its parameters.



Picture: Implementing a selected BAdI method.

Finally, after having implemented the required BAdI methods, the implementation needs to be activated. This comprises activating the method implementations as well as activating the Enhancement Implementation (see also picture in step 3 where the Enhancement Implementation is still inactive). To get your implementation up and running, both need to be active!



## 4.2 Process Controller Strategies

The Process Controller Framework (PCF) allows the flexible definition of application processes. This is accomplished by defining a process as a sequence of methods which represent the single process steps. Such a sequence of methods is called a Strategy.

Using the PCF means the definition of an application process as a sequence of methods (a Strategy) in customizing by SAP, partners and customers. Pieces of functionality can be packed into a method, which can then be included and used in a strategy. Partners and Customers can define their own methods and combine them either to completely new strategies or they can include their own methods in SAP standard strategies, i.e. they also can enhance them with their customer specific functionality.

The following table with predefined services provides an overview of different application areas and functionalities within Transportation Management that make use of the Process Controller Framework:

Service	Description / Purpose
COPY_CONTR	Copy Control
DDD_DET	Distance and Duration Determination
GEO_DET	Geo Coordinate Determination
GEO_ROUTE	Geo Route Determination
RG_DYNAMIC	Dynamic Routing Guide
RG_FIX	Fixed Routing Guide
TM_DG	TM Dangerous Goods
TM_GT_GRP	Customs Groups
TM_INVOICE	TM Invoicing
TM_TSPS	TM Carrier Selection
TOR_CHACO	TM Change Controller for Changes (Freight Order)
TOR_CREATE	TM Change Controller for Creation (Freight Order)
TOR_DELETE	TM Change Controller for Deletion (Freight Order)
TOR_DSO	TM Direct Shipment Options (Freight Order)
TOR_SAVE	TM Change Controller for Saving (Freight Order)
VSR	TM VSR Optimizer
VSR_INTER	TM VSR Interactive Planning
...	...

### 4.2.1 Relevant parts of the Process Controller

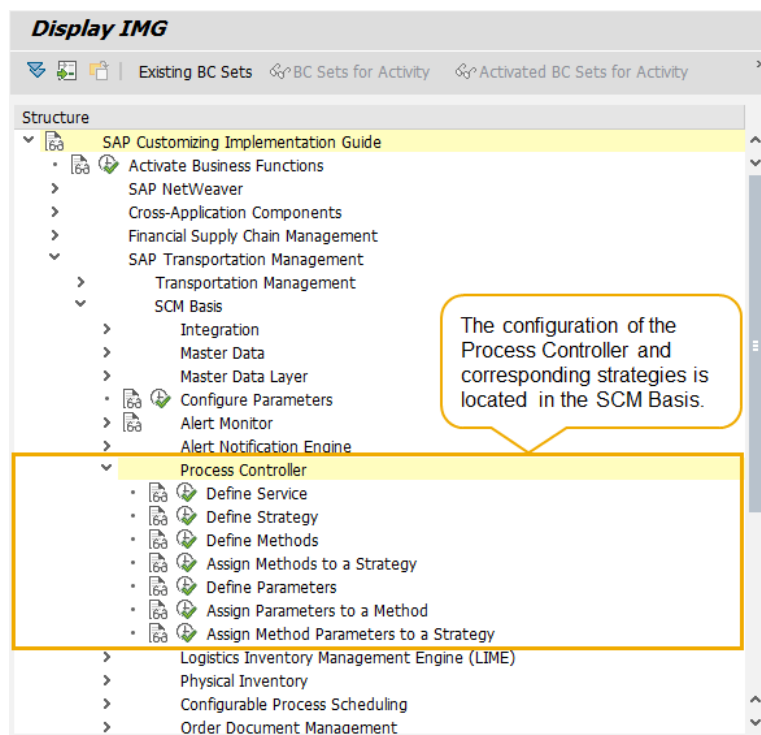
The Process Controller Framework comprises the following entities:

- Service:**  
A service is the definition of strategies and methods working on the same process. It is used to clearly separate the maintenance.
- Method:**  
All methods share the same interface of an object for providing parameters and a list of mutually independent requests. Some methods comprise functionality themselves, whereas others mainly encapsulate existing functionality (like EH&S check within RGE). There is a method pool for each process, and each method pool contains all methods which the application process requires in order to define its strategies.
- Strategy:**  
A strategy serializes a selection of methods from the method pool. It brings them in the sequence in which they shall be executed by the Controller.

- Parameter:**  
 Parameters influence the behavior of the respective methods, but do not change the sequence in which the methods are executed. The Controller passes them on to each method through its interface. A parameter may not only have different values in different methods, but also in the same method as part of different strategies, or even in the same method of the same strategy as part of different requests.
- Request:**  
 The requests represent the various business demands (e.g. determination of a best route) on an application process. The Controller processes the requests by passing them on to the strategy methods through their interfaces. The requests may further allow storing possible solutions.
- Application Class:**  
 The methods in the pool of the Controller are provided by so-called application classes. Method execution by the Controller works (dynamically) since the definition of a new method includes the name of the providing application class. The results produced by a method are either passed on to all succeeding methods with the corresponding requests or they are stored externally, if the calling application, but not succeeding methods take advantage of these results.

#### 4.2.2 Setting up a Process Controller Strategy

The configuration of Process Controller Strategies is done in the IMG (Transaction *SPRO*), located under the path shown below:



Picture: Configuration of PCF in IMG.

The path in IMG (Transaction *SPRO*) is: *SAP Transportation Management* → *SCM Basis* → *Process Controller*. From here all required steps for defining a strategy can be triggered.

As an example for the configuration of a Process Controller Strategy and the implementation of an example method, we create a new strategy for the SAP Standard Service *TOR\_SAVE*. In the application area Freight Order Management you can assign a Save Strategy to a Freight Order Type which allows executing follow-on functions when saving Freight Orders.

- 1) The first step is to create a new strategy belonging to the service *TOR\_SAVE*. In the mentioned IMG path click on *Define Strategy* and on the following screen on *New Entries* (you could also mark an existing strategy and copy it with a new name).

Strategy	Service	Description	Created by	Created on	Changed
ZENH_TORSV	TOR_SAVE	Enh. Demo: New TOR_SAVE Strategy			
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				

Picture: Creating a new strategy.

- 2) In the next step, we will define a class that provides a new method to be part of the method pool for the used service *TOR\_SAVE*. The example method shall do a Charge Calculation for a Freight Order. For the definition of the class and its implementation, transaction *SE24* is used (for demo purposes, you should use the customer namespace and save as local object).

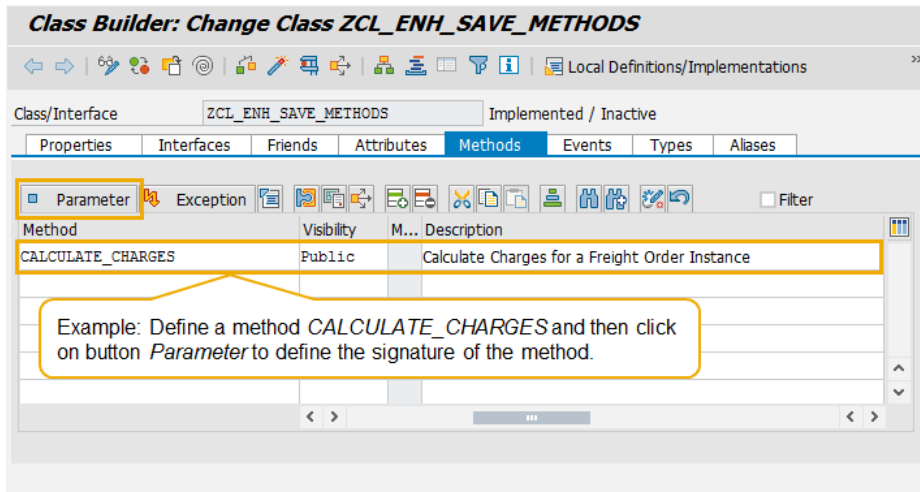
Class Browser

Object type: ZCL\_ENH\_SAVE\_METHODS

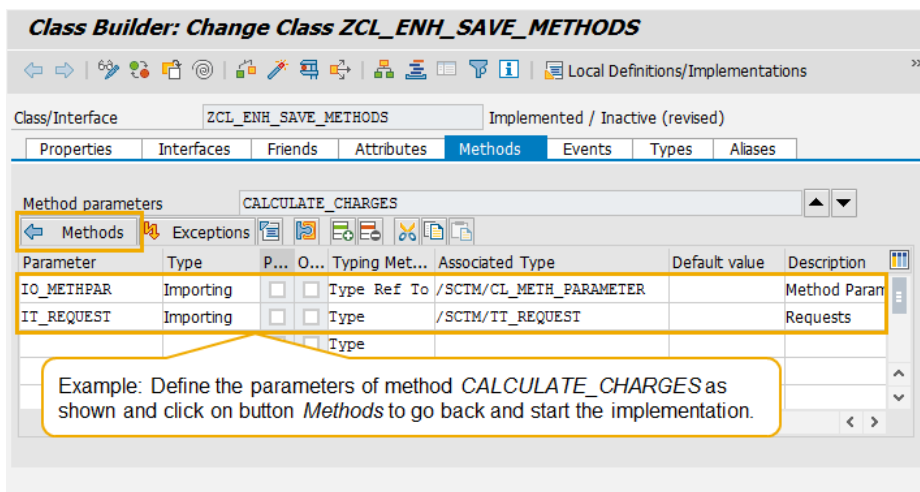
Buttons: Display, Change, Create

Picture: Creating a new class with transaction SE24.

- 3) Within the class we define a method `CALCULATE_CHARGES` which has to provide specific parameters so that the Process Controller Framework can execute it with the relevant data. After the parameters have been defined, the implementation of the method can be started by double clicking on the methods name in the methods overview. When the class implementation is complete, save and activate it.



Picture: Defining a method with transaction SE24.



Picture: Defining method parameters with transaction SE24.

At runtime, the example method receives Process Controller Requests and executes the action `CALC_TRANSPORTATION_CHARGES` of all instances of the business object TOR (e.g. Freight Orders) provided with each request. The example coding for the method looks as follows:

```
METHOD calculate_charges.

  DATA: lo_request      TYPE REF TO /sctm/cl_request,
        lo_tor_save_request TYPE REF TO /scmtms/cl_chaco_request,
        lt_failed_key    TYPE /bobf/t_frwr_key,
        lo_message       TYPE REF TO /bobf/if_frwr_message.

  LOOP AT it_request INTO lo_request.
    lo_tor_save_request = /scmtms/cl_tor_helper_chaco=>cast_request
    ( lo_request ).
    CHECK lo_tor_save_request IS BOUND.
```

```

*****
**** calc. the charges for the uncanceled and unfinalized insta
nces
*****
***
CALL METHOD
  lo_tor_save_request->mo_tor_srvmgr->do_action(
    EXPORTING
      iv_act_key      = /scmtms/if_tor_c=>sc_action-root-
                        calc_transportation_charges
      it_key           = lo_tor_save_request->mt_tor_key_active
    IMPORTING
      eo_message       = lo_message
      et_failed_key    = lt_failed_key ).

  APPEND LINES OF lt_failed_key TO lo_tor_save_request->
                                mt_failed_key.

* add messages to change controller request
  /scmtms/cl_common_helper=>msg_helper_add_mo(
    EXPORTING
      io_new_message = lo_message
    CHANGING
      co_message      = lo_tor_save_request-
>mo_message ).
  ENDLOOP.

ENDMETHOD.

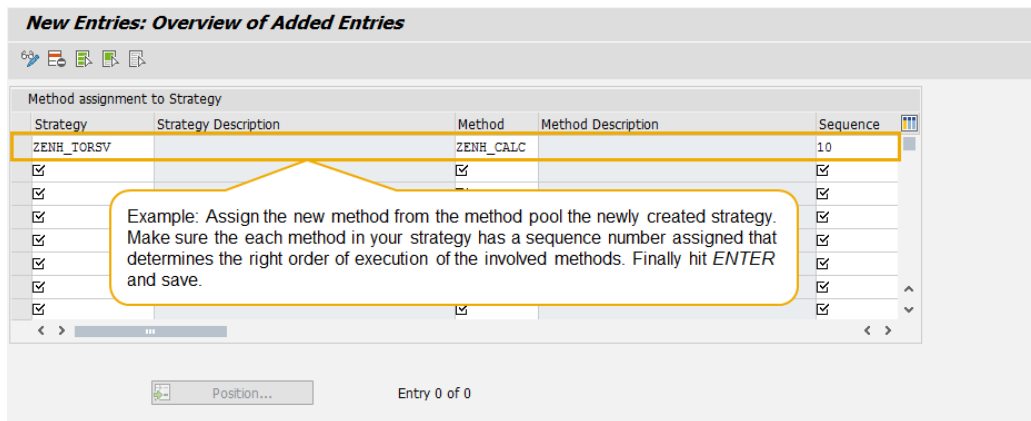
```

- 4) Now you can assign the new method to the method pool of the *TOR\_SAVE* service. This is again done in the IMG. In the IMG path click on *Define Methods* and on the following screen on *New Entries*.

Method	Service	Description	MethType	Class/Interface	Interface Component
ZENH_CALC	TOR_SAVE	Calculate Charges for FO	Basic Method	ZCL_ENH_SAVE_METHODS	CALCULATE_CHARGES

Picture: Assigning the method to the method pool of service *TOR\_SAVE*.

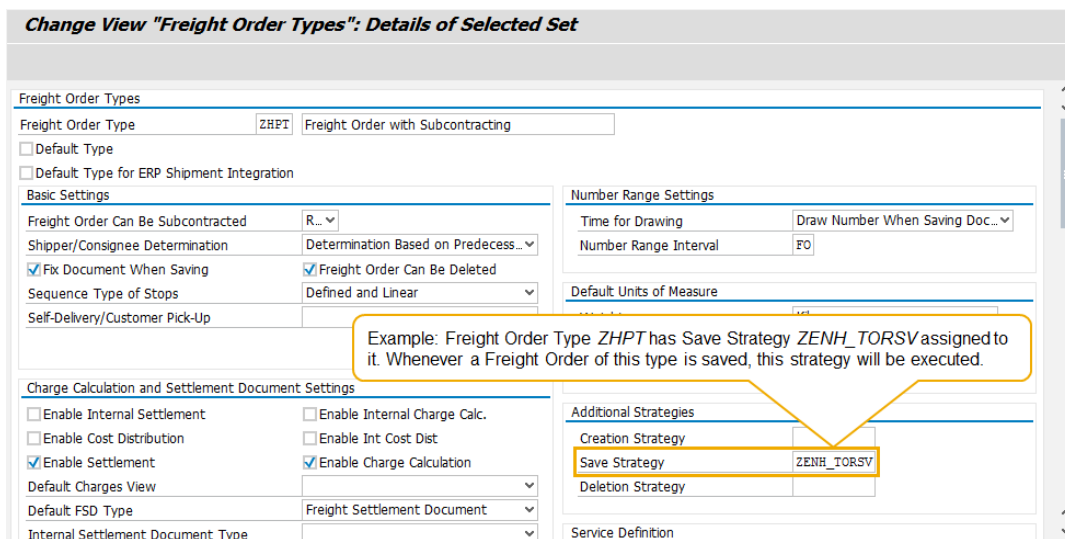
- 5) Finally, we assign the new method from the method pool to our new strategy which was defined in the first step. In the mentioned IMG path click on *Assign Methods to a Strategy* and on the following screen on *New Entries*.



Picture: Assigning the method from method pool to the strategy.

The example strategy is now ready to be used and allows triggering a Charge Calculation when saving e.g. a Freight Order that has this save strategy assigned in its corresponding Freight Order Type. Further methods can be added to the strategy by repeating the described steps 3 - 5. Further methods can be added and implemented in the defined class to realize additional functionality that shall be executed on save of a Freight Order. These additional methods are then assigned to the method pool of the underlying service and to the strategy. The execution of the methods is done in the sequence defined in the strategy.

Now the new save strategy can be assigned to a Freight Order Type and is executed whenever a Freight Order of this specific type is saved. This assignment is done in the IMG under the following path: *SAP Transportation Management* → *Transportation Management* → *Freight Order Management* → *Freight Order* → *Define Freight Order Types*.



Picture: Assigning the new strategy to a Freight Order Type.

With similar steps customers and partners can not only create their very own strategies but can also enhance SAP standard strategies to execute additional, customer specific functionality.

### 4.2.3 Using the Process Controller Framework for a new process

The following section describes an example how to use the Process Controller Framework for the implementation of a new process. Running your own Process Controller application requires corresponding customizing settings as described in the previous sections as well as a technical implementation part.

- 1) Create a new Process Controller Service in customizing under the path:

SAP Transportation Management → SCM Basis → Process Controller → Define Service.

Example:

Service	Description
ZENHDEMO	Enhancement Demo Service

- 2) Define an external and internal request and result structure: The request data must contain the information on which process specific strategy shall be executed. An example set of the required structures and table types:

Structure		Description
ZENH_S_DEMO_REQUEST_STR		Demo Request: Single Request (internal)
Component	Typing Method	Component Type
TOR_ID	Types	/SCMTMS/TOR_ID
TEXT	Types	/SCMTMS/STRING
APPROVAL	Types	BOOLEAN
APPROVALDATE	Types	/SCMTMS/DATETIME

Structure		Description
ZENH_S_DEMO_RESULT		Demo Request: Result (external)
Component	Typing Method	Component Type
ITEM_KEY	Types	/BOBF/CONF_KEY
ROOT_KEY	Types	/BOBF/CONF_KEY
GRO_WEI_VAL	Types	/SCMTMS/QUA_GRO_WEI_VAL
GRO_WEI_UNI	Types	/SCMTMS/QUA_GRO_WEI_UNI

Also define the corresponding table types:

Table Type	Description
ZENH_T_DEMO_REQUEST_STR	Demo Request: Request Table (internal)
Line Type	
ZENH_S_DEMO_REQUEST_STR	

Table Type	Description
ZENH_T_DEMO_RESULT	Demo Request: Result Table (external)
Line Type	
ZENH_S_DEMO_RESULT	

Define the internal request and result structure: These structures are used within the request objects and could be the same as the external structures in a simple process.

Structure		Description
ZENH_S_DEMO_REQUEST_INT		Demo Request: Internal View
Component	Typing Method	Component Type
.INCLUDE	Types	ZENH_S_DEMO_REQUEST_STR
.INCLUDE	Types	ZENH_S_DEMO_RESULT
RETURN_CODE	Types	/SCMTMS/STRING

Structure		Description
ZENH_S_DEMO_REQUEST		Demo Request: External View
Component	Typing Method	Component Type
STRATEGY	Types	/SCTM/DE_STRATEGY
REQUESTS	Types	ZENH_T_DEMO_REQUEST_INT

Also define the corresponding table types:

Table Type	Description
ZENH_T_DEMO_REQUEST_INT	Demo Request: Internal View Table



Line Type
ZENH_S_DEMO_REQUEST_INT

Table Type	Description
ZENH_T_DEMO_REQUEST	Demo Requests: External View Table
Line Type	
ZENH_S_DEMO_REQUEST	

- 3) Create a *Request Object Class*: The request object represents the container for request and result data as well as process specific options. It will be filled by the controller and passed through the Process Controller Framework from method to method (i.e. it is the generic interface between the methods of a strategy).

- Create a class [namespace]CL\_[process]\_REQUEST which inherits from the super class /SCTM/CL\_REQUEST. Example: ZENH\_CL\_DEMO\_REQUEST.
- Add public attributes for request and result data as well as (if required) process specific options. In example class ZENH\_CL\_DEMO\_REQUEST add the following attributes:

Attribute	Level	Visibility	Typing	Associated Type
MV_STRATEGY	Instance Attribute	Public	Type	/SCTM/DE_STRATEGY
MT_REQUESTS	Instance Attribute	Public	Type	ZENH_T_DEMO_REQUEST_INT
MT_RESULTS	Instance Attribute	Public	Type	ZENH_T_DEMO_RESULT
MO_MESSAGE_HANDLER	Instance Attribute	Public	Type Ref To	/BOBF/IF_FRW_MESSAGE

- Implement a constructor for the class with the following parameters (the method *CONSTRUCTOR* is a public instance method):

Parameter	Pass Value	Optional	Typing Method	Associated Type
IV_REQUEST_ID			Type	/SCTM/DE_REQUEST_ID
IV_STRATEGY			Type	/SCTM/DE_STRATEGY
IT_REQUESTS			Type	ZENH_T_DEMO_REQUEST_INT

The example coding for the constructor looks as follows:

```
METHOD constructor.
* call constructor of super class
  super->constructor( iv_request_id ).

* assign constructor parameters to member variables
  mv_strategy = iv_strategy.
  mt_requests = it_requests.

ENDMETHOD.
```

- 4) Create a *Controller Class*: The controller object is responsible for mapping input data to internal structures, creating requests, starting the Process Controller Framework and finally mapping the results to the external structures.

- Create a class [namespace]CL\_[process]\_CONTROLLER which inherits from the super class /SCTM/CL\_CONTROLLER. Example:

ZENH\_CL\_DEMO\_CONTROLLER

- Create a private instance method `CREATE_REQUEST_OBJECTS` with the following parameters:

Parameter	Type	Pass Value	Typing Method	Associated Type
IT_REQUEST_DATA	Importing		Type	ZENH_T_DEMO_REQUEST
ET_STRATEGY_REQUE STS	Exporting	Yes	Type	/SCTM/TT_CON_REQUEST_ STRATEGY
MR_MESSAGE	Changing		Type Ref To	/BOBF/IF_FRW_MESSAGE

The method is responsible for mapping the input data to internal structures, creating requests and sorting them into the Process Controller Framework.

The example coding for method `CREATE_REQUEST_OBJECTS` looks as follows and can be used as a template for your own implementations:

**METHOD** create\_request\_objects.

```

DATA: lv_message          TYPE string,                "#EC NEEDED
      lo_request          TYPE REF TO zenh_cl_demo_request,
      lt_strategy_ids     TYPE /sctm/tt_strategy_id,
      lv_exists           TYPE boole_d,
      lv_request_id       TYPE /sctm/de_request_id.

FIELD-SYMBOLS: <fs_request> TYPE zenh_s_demo_request.

CLEAR et_strategy_requests.

* Fill complete strategy, method sequence, parameter and detail
* buffer
CLEAR lt_strategy_ids.
LOOP AT it_request_data ASSIGNING <fs_request>.
    INSERT <fs_request>-strategy INTO TABLE lt_strategy_ids.
ENDLOOP.
fill_strategy_buffer( lt_strategy_ids ).

* Create the requests according to their strategy
lv_request_id = 0.
LOOP AT it_request_data ASSIGNING <fs_request>.
    lv_exists = check_strategy( <fs_request>-strategy ).
    IF lv_exists = abap_false.
* Given Strategy does not exist; all requests assigned can
* not be handled
* MESSAGE e033(/sctm/rg) WITH <fs_request>-
* request_id INTO lv_message.
        CONTINUE.
    ENDIF.

#####
* Insert your data mapping here if required!!!
#####

    lv_request_id = lv_request_id + 1.
    CREATE OBJECT lo_request
        EXPORTING
            iv_request_id = lv_request_id
            iv_strategy   = <fs_request>-strategy
            it_requests   = <fs_request>-requests.

    assign_request_to_strategy(

```

```

EXPORTING
    io_request      = io_request
    iv_strategy     = <fs_request>-strategy
    CHANGING ct_strategy_requests = et_strategy_requests ).

ENDLOOP.

ENDMETHOD.

```

- Create a public instance method *EXECUTE\_DETERMINATION* with the following parameters:

Parameter	Type	Opt.	Typing Method	Associated Type
IT_REQUEST_DATA	Importing		Type	ZENH_T_DEMO_REQUEST
IT_INPUT_METHPAR	Importing	Yes	Type	/SCTM/TT_CON_INPUT_METHPAR
ET_RESULT	Exporting		Type	ZENH_T_DEMO_RESULT
CT_REQUEST	Changing	Yes	Type	ZENH_T_DEMO_REQUEST
CO_MESSAGE_HANDLER	Changing	Yes	Type Ref To	/BOBF/IF_FRW_MESSAGE

The method is responsible Process Controller Framework execution including the data mapping. The example coding for method *EXECUTE\_DETERMINATION* looks as follows and again can be used as a template:

```

METHOD execute_determination.

    DATA: lt_strategy_requests TYPE /sctm/tt_con_request_strategy
    .

    DATA: lo_request          TYPE REF TO /sctm/cl_request,
           lo_demo_request    TYPE REF TO zenh_cl_demo_request,
           lt_bapiret2        TYPE bapirettab.

    FIELD-SYMBOLS:
        <fs_strategy_request> TYPE /sctm/s_con_request_strategy,
        <fs_result>           TYPE zenh_s_demo_result.

    CLEAR et_result.

    * Transform the supplied request data into request objects
    CLEAR lt_strategy_requests.

    create_request_objects(
        EXPORTING it_request_data      = it_request_data
        IMPORTING et_strategy_requests = lt_strategy_requests
        CHANGING  mr_message           = co_message_handler ).

    * Fill created request objects into controller
    clear_refill_attributes( it_input_methpar = it_input_methpar
                             it_strategy_requests = lt_strategy_requests ).

    CLEAR lt_bapiret2.
    start_perform_requests( IMPORTING et_bapiret2 = lt_bapiret2 )
    .

    * Get results / messages from every single request
    LOOP AT mt_strategy_requests ASSIGNING <fs_strategy_request>.
        LOOP AT <fs_strategy_request>-t_request INTO lo_request.
            lo_demo_request = zenh_cl_demo_methods=>

```

```

                                cast_request( lo_request ).
    CHECK lo_demo_request IS BOUND.

    * Take over results
    LOOP AT lo_demo_request-
>mt_results ASSIGNING <fs_result>.
        INSERT <fs_result> INTO TABLE et_result.
    ENDLOOP.

    * Take over request specific messages
    ENDLOOP.
    ENDLOOP.

ENDMETHOD.

```

- 5) Create a Method Pool Class: This class contains all the required functionality for the new process, i.e. it contains the implementation of the methods that can be combined to strategies.

- Create a class [namespace]CL\_[process]\_METHODS. Example:

ZENH\_CL\_DEMO\_METHODS

- Create a static public method *CAST\_REQUEST* with the following parameters:

Parameter	Type	Pass Value	Typing Method	Associated Type
IO_REQUEST	Importing		Type Ref To	/SCTM/CL_REQUEST
RO_REQUEST	Returning	Yes	Type Ref To	ZENH_CL_DEMO_REQUEST

The method is responsible for casting the generic request into the process specific request. The example coding for method *CAST\_REQUEST* looks as follows and can be used as a template:

```

METHOD cast_request.
    TRY.
        ro_request ?= io_request.
    CATCH cx_sy_move_cast_error.
        CLEAR ro_request.
    ENDTRY.
ENDMETHOD.

```

- For each of the required steps for the new process create a public instance method with the following parameters:

Parameter	Type	Typing Method	Associated Type
IO_METHPAR	Importing	Type Ref To	/SCTM/CL_METH_PARAMETER
IT_REQUEST	Importing	Type	/SCTM/TT_REQUEST

A general template for an implementation of the methods looks as follows:

```

METHOD your_functionality.
    DATA: lo_request TYPE REF TO /sctm/cl_request,
           lo_[PROCESS]_request TYPE REF TO [YOUR REQUEST OBJECT].

    LOOP AT it_request INTO lo_request.

```

```

        lo_[PROCESS]_request = cast_request( lo_request
    ).
    CHECK lo_[PROCESS]_request IS NOT INITIAL.

#####
*Execute your process here; take request data out
*of lo_[PROCESS]_request and put in result
#####
    ENDLOOP.
ENDMETHOD.

```

The process controller strategies can also include methods from other method pool classes. This possibility can be used to combine standard method with customer specific methods in a strategy where the customer specific methods are implemented in a separate, customer method pool class.

Add the following two example methods to class `ZENH_CL_DEMO_METHODS` with the following example code and the parameters as shown above.

**METHOD first\_enh\_method.**

```

DATA: lo_request      TYPE REF TO /sctm/cl_request,
      lo_demo_request TYPE REF TO zenh_cl_demo_request,
      ls_result       TYPE zenh_s_demo_result.

BREAK-POINT.

* get the demo request object and execute it
LOOP AT it_request INTO lo_request.
    TRY.
        lo_demo_request ?= lo_request.
        CATCH cx_sy_move_cast_error.
            lo_request->mv_interrupt = abap_true.
            RETURN.
        ENDTRY.

        ls_result-root_key    = '2'.
        ls_result-item_key    = '10'.
        ls_result-gro_wei_uni = 'PC'.
        ls_result-gro_wei_val = 100.
        APPEND ls_result TO lo_demo_request->mt_results.

    ENDLOOP.

BREAK-POINT.

ENDMETHOD.

```

**METHOD second\_enh\_method.**

```

DATA: lo_request      TYPE REF TO /sctm/cl_request,
      lo_demo_request TYPE REF TO zenh_cl_demo_request,
      ls_requests     TYPE zenh_s_demo_request_int.

FIELD-SYMBOLS: <fs_result> TYPE zenh_s_demo_result.

BREAK-POINT.

* get the demo request object and execute it

```

```

        LOOP AT it_request INTO lo_request.
            TRY.
                lo_demo_request ?= lo_request.
                CATCH cx_sy_move_cast_error.
                    lo_request->mv_interrupt = abap_true.
                    RETURN.
            ENDTRY.

            LOOP AT lo_demo_request->mt_results ASSIGNING <fs_result>.
*           Use the input data of the request to influence the result
                READ TABLE lo_demo_request->mt_requests
                    INTO ls_requests INDEX 1.
                IF ls_requests-approval = abap_true.
                    <fs_result>-gro_wei_val = <fs_result>-gro_wei_val * 10.
                ENDIF.
            ENDLOOP.

        ENDLOOP.

        BREAK-POINT.

    ENDMETHOD.

```

- 6) Configure a strategy in customizing as shown in the previous sections to test the new process. The following report allows direct execution of a strategy which is based on the new example process created with steps 1 - 5.

```

*&-----*
*& Report   ZREP_PCF_DEMO_DIRECT
*&-----*
*& This demo report allows direct execution of a demo strategy.
*&-----*
REPORT   zrep_pcf_demo_direct.
* declarations
DATA: lo_demo_controller TYPE REF TO zenh_cl_demo_controller,
      ls_req              TYPE zenh_s_demo_request,
      lt_req              TYPE zenh_t_demo_request,
      lt_demo_result      TYPE zenh_t_demo_result,
      ls_data             TYPE zenh_s_demo_request_int,
      lt_data             TYPE zenh_t_demo_request_int.

FIELD-SYMBOLS: <fs_result> TYPE zenh_s_demo_result.

BREAK-POINT.

* create an instance of the demo controller
CREATE OBJECT lo_demo_controller.

* define strategy to be executed (prepare in customizing!)
CLEAR ls_req.
* PLACE YOUR CONFIGURED TEST STRATEGY HERE!
ls_req-strategy = 'ZENHDEMO'.

* assemble some example input data for the strategy
CLEAR lt_data.
ls_data-tor_id      = '2'.
ls_data-approval    = abap_false.
ls_data-approvaldate = sy-datum.
ls_data-text        = 'Test'.
INSERT ls_data INTO TABLE lt_data.
ls_req-requests = lt_data.

```

```

INSERT ls_req INTO TABLE lt_req.

* execute the
lo_demo_controller->execute_determination(
    EXPORTING it_request_data = lt_req
    IMPORTING et_result       = lt_demo_result ).

BREAK-POINT.

WRITE: /,'A direct call of the Demo Controller'.
LOOP AT lt_demo_result ASSIGNING <fs_result>.
    WRITE: /,'Gross Weight Value',<fs_result>-gro_wei_val.
ENDLOOP.

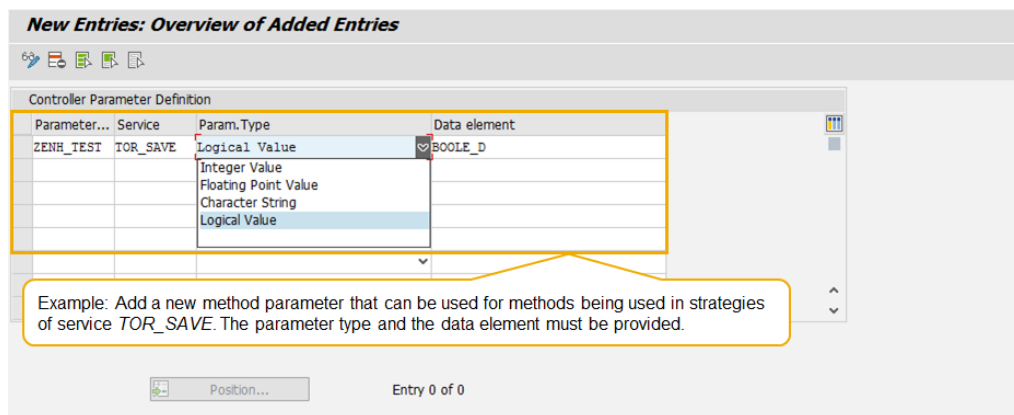
```

Execute the example report with a consistent example strategy and debug it to see how it finally works in general. The example code contains implemented break-points at all relevant places in the code to see the different aspects of the strategy being executed.

## 4.2.4 Using Method Parameters

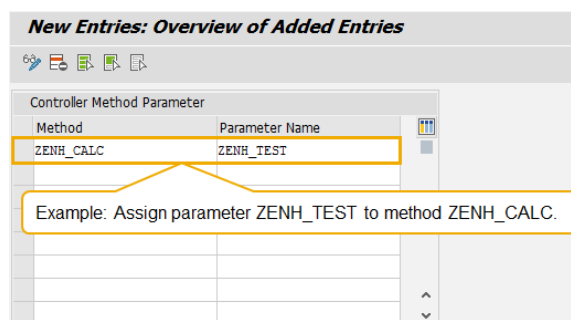
In case you define additional, similar strategies that only differ in minor parts of a method, you can use method parameters to reuse the same method for several strategies with a deviating logic. The logic of such a parameter must be implemented in the corresponding standard method.

- 1) In the IMG path for the Process Controller click on *Define Parameters* for defining method parameters. As an example, we define the new parameter *ZENH\_TEST* for service *TOR\_SAVE* which shall be a logical value defined by data element *BOOLE\_D*. Save this customizing entry.



Picture: Defining a new parameter.

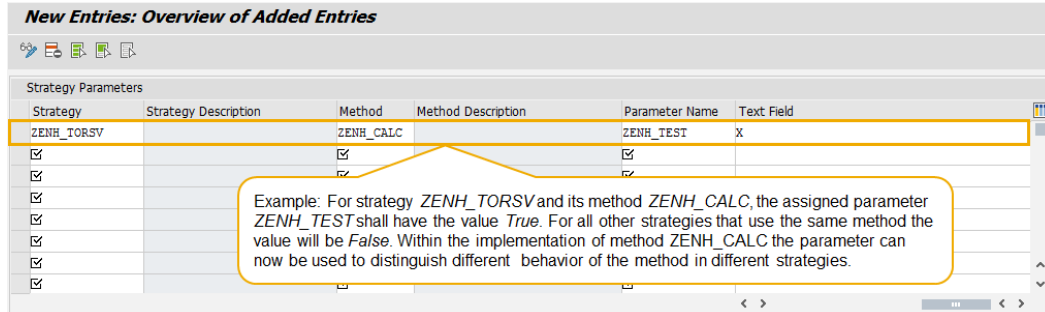
- 2) Now click on *Assign Parameters to a Method* in the IMG path for assigning the new parameter to our example method *ZENH\_CALC*.



Picture: Assigning the new parameter to the method.



- 3) Parameter `ZENH_TEST` can now be used by the method `ZENH_CALC` in a strategy. Assume we have multiple strategies defined for service `TOR_SAVE` and all of them make use of example method `ZENH_CALC`. In the next step, we define that the parameter `ZENH_TEST` of method `ZENH_CALC` shall have the value `True` in the context of our example strategy `ZENH_TORSV`. In the IMG path click on *Assign Method Parameters to a Strategy*.



Picture: Setting the value for a method parameter for a strategy.

In the implementation of the method the parameter value can be retrieved as follows (the example code for reading the parameter is highlighted):

**METHOD** calculate\_charges.

```

DATA: lo_request          TYPE REF TO /sctm/cl_request,
      lo_tor_save_request TYPE REF TO /scmtms/cl_chaco_request,
      lt_failed_key       TYPE /bobf/t_frw_key,
      lo_message          TYPE REF TO /bobf/if_frw_message.

DATA: lv_zenh_test       TYPE boole_d.

LOOP AT it_request INTO lo_request.
  lo_tor_save_request = /scmtms/cl_tor_helper_chaco=>
                        cast_request( lo_request ).
  CHECK lo_tor_save_request IS BOUND.

*****
* read the parameters assigned to the method and take over the value
* depending on the parameter type you must use method
* - io_methpar->get_bool_par
* - io_methpar->get_float_par
* - io_methpar->get_int_par
* - io_methpar->get_string_par
* to get the corresponding parameter values returned
*****
  lv_zenh_test = io_methpar->get_bool_par(
    iv_request_id = lo_tor_save_request->mv_id
    iv_method      = 'ZENH_CALC'
    iv_param       = 'ZENH_TEST' ).

*****
* use the parameter to define/change the methods logic
*****
  IF lv_zenh_test = abap_true.
    ...
  ENDIF.
...
ENDLOOP.
ENDMETHOD.

```

## 4.3 Conditions

In TM conditions can be defined and configured to influence the business logic. Such conditions are used as filters and for automated decision making in many areas of the TM application.

Conditions use a set of input values which can e.g. come from attributes of the TM Business Objects. These input values are then mapped onto corresponding output values. The output of a condition can be a simple Boolean which indicates to select a business object instance (yes/no), it can be a simple value like an ID of a business partner or it can be a set of output values which are determined by the condition based on the input values.

Example: In the customizing for a Forwarding Order Type, you can place a condition that will determine the Freight Unit Building Rule that shall be used when processing a Forwarding Order of this type. The condition can e.g. determine a Freight Unit Building Rule to be used based on source and destination location provided in the Forwarding Order. In the same customizing, you can define a similar condition that determines the sales organization to be used with the Forwarding Order.

### 4.3.1 Customizing: Condition Types and Data Access Definitions

A condition type defines possible input values as well as the output of conditions of that type. The input values are defined by so called data access definitions.

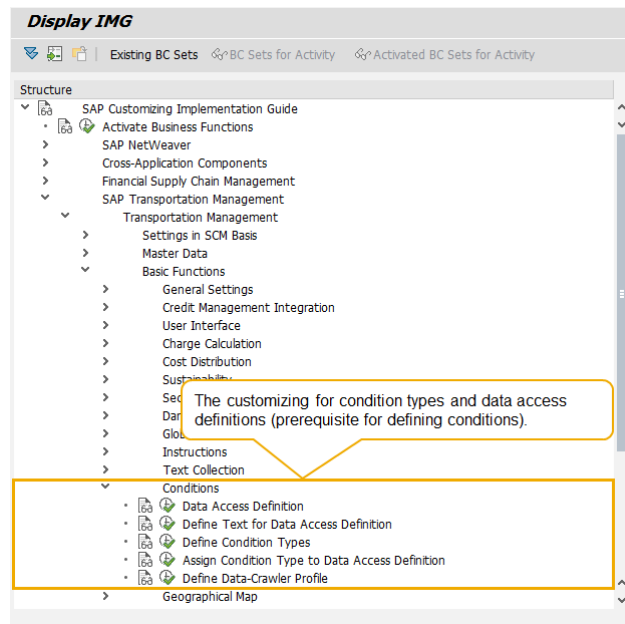
Data access definitions define how the content of the input values is read during runtime. This can e.g. be a generic access to a defined BO, BO node and BO node attribute (can be configured in the data access definition) or a specific class method that will provide the data (implementation required). A condition type gets assigned one or more such data access definitions. When creating a condition of this type, the input can be build up from these assigned data access definitions.

The output defined with a condition type can be single values like e.g. a simple Boolean or a Product ID. It can also be a complete structure with multiple attributes as the output result.

Example:

- Condition type */SCMTMS/FUBR* allows defining conditions to determine Freight Unit Building Rules for Forwarding Orders. The data access definitions assigned to it is */SCMTMS/TRQ* (Forwarding Order Type) and */SCMTMS/TRQ\_ITEM\_PRD* (Product ID in the Forwarding Order Item). The output of this condition type is defined to be the Freight Unit Building Rule ID.
- Conditions of this type can now be defined to use either the Forwarding Order Type or the Product IDs of the Forwarding Order Items or even both to determine a Freight Unit Building Rule.
- The condition can be assigned to a Forwarding Order Type in customizing. Whenever creating a Forwarding Order of this type, condition will then determine a required Freight Unit Building Rule.

SAP TM delivers a set of predefined condition types and data access definitions. Customers and partners can also define their own condition types and data access definitions in the customizing if required. You can find the customizing for condition types and data access definitions via transaction SPRO under the following path: *SAP Transportation Management* → *Transportation Management* → *Basic Functions* → *Conditions*.



Picture: Conditions Customizing.

Besides the Condition Types and Data Access Definitions delivered with SAP TM, customers and partners can define their own corresponding entities.

### 4.3.2 Creating Data Access Definitions

Data Access Definitions can be created using different approaches.

- 1) Defining a Data Access Definition by providing required Business Object data to access a single attribute of a given node. In the customizing follow the path *SAP Transportation Management* → *Transportation Management* → *Basic Functions* → *Conditions* → *Data Access Definition*.

- Click on button *New Entries*.
- Define a name and a description for the new Data Access Definition and provide the required details in section *BO Data*. Example: A Data Access Definition that returns the attribute *TOR\_CAT* from the Root node of Business Object */SCMTMS/TOR* (used for Freight Order, Freight Unit and others).

Field	Content	Comment
Data Access Def.	ZENH_TOR_CAT	The name of the new Data Access Definition.
Description	Enhancement DAD for TOR Category	A description of the new Data Access Definition.
Data Element for F4 help	/SCMTMS/TOR_CATEGORY	
Name of BO	/SCMTMS/TOR	Business Object Name
Name of BO Node	ROOT	The node of the Business Object where the attribute is read from.
BO Node Field Name	TOR_CAT	The name of the node attribute on the provided node.
Filter 1: BO Field Name	CREATED_BY	An additional filter attribute that has to be available on the same node.
Filter 1: Field Value	POLCH	The value for the filter.

The Data Access Definition in this example realizes a generic access to the attribute *TOR\_CAT* on the Root node of business object */SCMTMS/TOR* with a corresponding data element assigned that provides a suitable F4 Help during further usage of the Data Access Definition. As indicated in the example, you can also define up to two further attributes of the same specified node to serve as filters (example: only return the value for

attribute *TOR\_CAT* if attribute *CREATED\_BY* = "POLCH"). This type of Data Access Definition does not require further implementation and can be used right away.

Picture: Creating a new Data Access Definition.

- 2) Define a Data Access Definition by defining a *Data Crawler Profile*. The Data Crawler is a tool that allows to define a (cross BO) navigation path from a source node to a target node and to retrieve required data from the included nodes. Again, this type of Data Access Definition does not require further implementation and can be used right away.

The Data Crawler Profile information can be maintained in the IMG via path *SAP Transportation Management* → *Transportation Management* → *Basic Functions* → *Conditions* → *Define Data Crawler Profile* (in older SAP TM releases to be maintained via transaction *SM34* view cluster */SCMTMS/VC\_DCPRF*).

Picture: Maintaining View Cluster for Data Crawler Profiles.

- Click on button *New Entries* (see picture above) to create the following example Data Crawler Profile.

Picture: Example Data Crawler Profile.

- The example Data Crawler Profile shall simply allow navigation from the Forwarding Order Root node to the Item Node. Enter the following data:

Field	Content	Comment
Profile ID	ZENH_TRA_ITEM	The name of the new Data Crawler Profile.
BO Name	/SCMTMS/TRQ	Name of the BO where the profile starts reading data → Source BO.
Node Name	ROOT	Name of the BO node where the profile starts reading data → Source BO node.
Fill Data	(space)	Optional. If this flag is set, the data of the defined source node will be read during execution of the profile.

Picture: An example path step of a Data Crawler Profile.

- In the dialog structure (see picture above) double click on **Path Steps** to add further target nodes that shall be navigated to. For the example enter the following path step:

Field	Content	Comment
Step ID	010	The number of the step. A Data Crawler Profile can have multiple steps that are combined to a path through the involved BOs and corresponding nodes.
Prev. Step ID	(space)	Allows specifying the previous step in the path after which the new step shall be executed.
BO Name	/SCMTMS/TRQ	The name of the Business Object
Source Node	ROOT	Predefined by the source node defined in the profile header.
Association	ITEM_MAIN	The association to be used for navigating to

		the Item node.
Target Node	ITEM	Predefined by the association and its target node.
Fill Data	(flag is set)	In case the flag is set, the data of the target node instances will be read. Otherwise if the flag is not set, it just returns source and target node keys that are used for potential navigation to further nodes in the node hierarchy, i.e. further path steps.

Using the Data Crawler in your own coding can be done by creating an instance of class /SCMTMS/CL\_DATA\_CRAWLER. The class provides the implementation of all methods required for using Data Crawler Profiles. The following example report shows how to read data with the Data Crawler Profile defined above:

```

*&-----*
*& Report  ZREP_DC_TEST
*&-----*
*& This report demonstrates the usage of a Data Crawler Profile to
*& read and retrieve data from a business object.
*&-----*
REPORT  zrep_dc_test.

DATA: ls_dc_prof_id  TYPE /scmtms/dc_profile_id,
      lt_dc_prof_id  TYPE /scmtms/t_dc_profile_id,
      ls_bo_inst_key TYPE /bobf/s_frw_key,
      lt_bo_inst_key TYPE /bobf/t_frw_key,
      lo_crawler      TYPE REF TO /scmtms/cl_data_crawler,
      lt_dc_data      TYPE /scmtms/cl_data_crawler=>tt_data,
      lo_message      TYPE REF TO /bobf/if_frw_message.

BREAK-POINT.

CLEAR: ls_dc_prof_id,
      lt_dc_prof_id.

* Specify the Data Crawler Profile to be used
ls_dc_prof_id = 'ZENH_TRQ_ITEM'.
APPEND ls_dc_prof_id TO lt_dc_prof_id.

* Specify the key of an example BO instance (here: a TRQ instance)
ls_bo_inst_key-key = '4D08C2BEC9015E16E10000000A421A6A'.
APPEND ls_bo_inst_key TO lt_bo_inst_key.

* Create an instance of the Data Crawler class
CREATE OBJECT lo_crawler
EXPORTING
    it_profile_id = lt_dc_prof_id.

* Call the Data Crawler with the given profile and BO instance
CALL METHOD lo_crawler->get_data
EXPORTING
    it_profile_id = lt_dc_prof_id
    it_key        = lt_bo_inst_key
IMPORTING
    et_data       = lt_dc_data
    eo_message    = lo_message.

* The resulting data can be found in LT_DC_DATA

BREAK-POINT.

```

**New Entries: Details of Added Entries**

Data Access Def.   
 Description   
 Data Element for F4 helps   
 Data Crawler  
 ProfileID   
 Step ID   
 Field Name

A Data Access Definition with a Data Crawler Profile as the input data source

Picture: A Data Access Definition with a Data Crawler Profile as data source.

The Data Access Definition example in the following table makes use of the Data Crawler Profile *ZENH\_TRQ\_ITEM* defined above to read the *Product ID* from the items of a given Forwarding Order.

Field	Content	Comment
Data Access Def.	ZENH_TRQ_ITEM_PRD	The name of the Data Access Definition.
Description	Enhancement DAD	Description for the Data Access Definition.
Data Element for F4 helps	/SCMTMS/PRODUCT_ID	The data element that provides a suitable F4 help for the field to be returned.
Profile ID	ZENH_TRQ_ITEM	The Data Crawler Profile to be used.
Step ID	10	Defines that step in the path of the profile that provides the content for the field to be returned.
Field Name	PRODUCT_ID	The field/attribute to be returned by the Data Access definition.

- 3) Realizing a Data Access Definition by implementing a *Determination Class*. While the first two options allow defining Data Access Definitions via configuration, this option requires creating a new class that implements interface */SCMTMS/IF\_COND\_DETERM\_CLASS*. The only method defined in this interface is *EXTRACT\_DATA\_MASS*.

Class */SCMTMS/CL\_COND\_CAPA\_CHECK* represents a nice example for an implementation of such a determination class. Within method *EXTRACT\_DATA\_MASS*, the capacity utilization of provided Freight Orders is read and compared with the maximum capacity. The result returned by the method is a statement whether the capacity is sufficient (returned result = "S") or is the capacity is overloaded (returned result = "O"). Using Determination Classes as the data source for a Data Access Definition especially makes sense when data needs to be read and analyzed to return an aggregated result rather than the data itself.

The following example implementation can e.g. be used to define a Data Access Definition that realizes the same functionality as the Data Crawler Profile example before. Create your own Determination Class *ZCL\_ENH\_DET\_CLASS* that implements method *EXTRACT\_DATA\_MASS* as follows:

```
METHOD /scmtms/if_cond_determ_class~extract_data_mass.

    FIELD-SYMBOLS: <fs_data> TYPE any.

    DATA: lo_srvmgr      TYPE REF TO /bobf/if_tra_service_manager,
           lo_message     TYPE REF TO /bobf/if_frw_message,
```



```

ls_item_data    TYPE /scmtms/s_trq_item_k,
lt_item_data    TYPE /scmtms/t_trq_item_k,
ls_dobo_link    TYPE /scmtms/s_cond_do_data_k,
ls_key          TYPE /bobf/s_frw_key,
ls_key_data     TYPE /scmtms/s_bokey_dokey_data.

BREAK-POINT.

* Get a service manager to access the TRQ BO
lo_srvmgr = /bobf/cl_tra_serv_mgr_factory=>
            get_service_manager( /scmtms/if_trq_c=>sc_bo_key ).

* Retrieve the Item Data for a given TRQ key
* via association ITEM_MAIN
lo_srvmgr->retrieve_by_association(
    EXPORTING
        iv_node_key      = /scmtms/if_trq_c=>sc_node-root
        it_key            = it_boinst_key
        iv_association    = /scmtms/if_trq_c=>sc_association-root-
                           item_main
        iv_fill_data      = abap_true
    IMPORTING
        eo_message        = lo_message
        et_data            = lt_item_data ).

* Build up the result data from the read bo data
LOOP AT it_boinst_key INTO ls_key.
    UNASSIGN <fs_data>.
    CREATE DATA ls_key_data-data TYPE /scmtms/product_id.
    ASSIGN ls_key_data-data->* TO <fs_data>.

    LOOP AT it_dobo_link INTO ls_dobo_link.
        ls_key_data-bokey = ls_key-key.
        ls_key_data-dokey = ls_dobo_link-dobj_id.

        " Example: Take over the first Product ID found
        READ TABLE lt_item_data INTO ls_item_data INDEX 1.
        IF sy-subrc = 0.
            " Return the BO key, the link between Data Object
            " and the BO as well as the resulting Prodcut ID
            ls_key_data-bokey = ls_key-key.
            ls_key_data-dokey = ls_dobo_link-dobj_id.
            <fs_data>          = ls_item_data-product_id.
        ENDIF.

        " Insert result data to the return patameter
        INSERT ls_key_data INTO TABLE et_bokey_dokey_data.

    ENDLOOP.

ENDLOOP.

BREAK-POINT.

ENDMETHOD.

```

Picture: A Data Access Definition with a Determination Class as data source.

Field	Content	Comment
Data Access Def.	ZENH_TRQ_ITEM_PRD2	The name of the Data Access Definition.
Description	Enhancement DAD	Description for the Data Access Definition.
Data Element for F4 helps	/SCMTMS/PRODUCT_ID	The data element that provides a suitable F4 help for the field to be returned.
Determination Class	ZCL_ENH_DET_CLASS	The name of the Determination Class.
Class Attribute	(optional)	

### 4.3.3 Creating Condition Types

Just like Data Access definitions, customers and partners can define new Condition types. In the customizing follow the path *SAP Transportation Management → Transportation Management → Basic Functions → Conditions → Define Condition Types*.

- Click on button *New Entries*.
- Define a name and a description for the new Condition Type and provide the required details in section *Maintenance View for Condition Types*. Example: The standard Condition Type */SCMTMS/FUBR* that returns a Freight Unit Building Rule ID. Conditions of this type use the Forwarding Order Type and Forwarding Order Item Product (see assignment of Data Access Definitions to this type in customizing).

Picture: A (standard) Condition Type.

Field	Content	Comment
Condition Type	/SCMTMS/FUBR	The name of the new Data Access Definition.
Description	FUB Rule Determination Cond.	A description of the new Data Access Definition.
Only one condition allowed for this condition type	[space]	If this flag is set, only one single condition can be defined with this type.
Result is a structure	[space]	If this flag is set, the condition is intended to return a result structure instead of a single attribute. In this case enter the structure name in field Result DDIC Type.
Result DDIC Type	/SCMTMS/FUBR_ID	The DDIC type for the result of a condition of this type.
Business Object	/SCMTMS/TRQ	The Business Object node whose keys are used as input for the condition (check for consistency?).
BO Node Name	ITEM	

#### 4.3.4 Assign Data Access Definitions to Condition Types

In this step, Data Access Definitions are assigned to a Condition type. When defining a condition of a given type, the assigned Data Access Definitions represent the pool of possible input data for the condition.

In the customizing follow the path SAP Transportation Management → Transportation Management → Basic Functions → Conditions → Assign Condition Type to Data Access Definition.

- Click on button *New Entries*.
- Define the Condition Type and the Data Access Definition on the following screen.
- Repeat this for all Data Access Definitions to be assigned to the Condition type.
- Save your data.

Example: The standard Condition Type /SCMTMS/FUBR has assigned two Data Access Definitions which serve as input and can be used for the definition of conditions of this type:

Condition Type	Data Access Definition	Comment
/SCMTMS/FUBR	/SCMTMS/TRQ_TYPE	The type of the Forwarding Order.
/SCMTMS/FUBR	/SCMTMS/TRQ_ITEM_PRD	The Forwarding Order Item Product ID.

**Change View "Link between the Condition type and the data access ..."**

New Entries

Link between the Condition type and the data access definiti

Condition Type	Data Access Definition ...	Dflt DAD	Position
/SCMTMS/FRO_TYPE_SHP	/SCMTMS/TOR_TSP	Data Access Definition Is D...	
/SCMTMS/FUBR	/SCMTMS/TRQ_ITEM_DLO		
/SCMTMS/FUBR	/SCMTMS/TRQ_ITEM_PRD	Data Access Definition Is D...	
/SCMTMS/FUBR	/SCMTMS/TRQ_ITEM_SLO		
/SCMTMS/FUBR	/SCMTMS/TRQ_ITEM_TYP		
/SCMTMS/FUBR	/SCMTMS/TRQ_ITEM_VOL		
/SCMTMS/FUBR	/SCMTMS/TRQ_ITEM_WEI	Data Access Definition Is D...	
/SCMTMS/FUBR	/SCMTMS/TRQ_TYPE	Data Access Definition Is D...	
/SCMTMS/FU_LOAD_DURA	/SCMTMS/FU_TOT_WEI	Data Access Definition Is D...	

Example: The Data Access Definitions assigned to the Condition Type /SCMTMS/FUBR.

Position... Entry /8 or 685

Picture: Assigning Data Access Definitions to a Condition Type.

### 4.3.5 Creating Conditions

Prerequisite for defining conditions is the proper setup and availability of the before mentioned condition types and data access definitions. The following examples describe how to set up a condition based on the standard condition types and data access definitions delivered with TM 9.0. The transactions for creating and editing conditions can be found in the user menu under the following path: Application Administration → General Settings → Conditions.

- 1) Choose **Create Condition** to start the creation of a new condition.
- 2) On the initial screen, enter the Condition Type, a description, the condition type and an origin of expression that shall be the basis for the new condition.

Picture: Enter condition type on the initial screen.

Example values to enter in this step:

Field	Value
Condition	DEMO_FUBR_COND
Description	Demo Condition for FUBR determination
Condition Type	/SCMTMS/FUBR
Origin of Condition	Condition based on BRFplus Decision Table

For our example, we choose **Condition based on BRFplus Decision Table** as the origin of condition which represents the default. In general, you can choose from three options:

- **Direct Business Object Access:**  
The system directly takes over the input values of a condition as the output values.
- **Condition based on BRFplus Decision Table:**  
The system maps a set of input values onto corresponding output values. This mapping is defined in a decision table (default).
- **Condition based on BRFplus Expression:**  
The system decides based on a logical expression what the result of a condition shall be. The data read by the data access definition is combined in a logical expression that then returns either Yes (logical expression is true) or No (logical expression is false).

**New Condition Definition DEMO\_FUBR\_COND**

Save Cancel Edit Check Data Access Definition Transport

No Messages - Display Message Log

**Condition Definition**

Condition: DEMO\_FUBR\_COND  
Description: Demo Condition for FUBR determination  
Condition Type: /SCMTMS/FUBR  
Origin of Condition: Condition Based on BRFplus Decision Table

**Decision Table: DEMO\_FUBR\_COND**

Additional Actions Context Overview Start Simulation

**Table Contents**

TR: Type	TR Item: Product	FUB Rule
TR01 (TM100 Group 01 DOT)	DIT-PROD-10-1	A-FUBR-01
FWO (Forwarding Order)	DIT-PROD-20-1	...

Changed By: SCMSUPPORT Created By: SCMSUPPORT  
Changed On/At: 20.09.2013 16:03:43 CET Created On/At: 20.09.2013 15:58:37 CET

Picture: Entering the first condition details.

**New Condition Definition DEMO\_FUBR\_COND**

Back Edit

No Messages - Display Message Log

**Data Access Definition**

Create Delete

	Data Access Definition for Conditions	Data Object Description	Data Element Used for Input Help
20	/SCMTMS/TRQ_ITEM_PRD	TR Item: Product	/SCMTMS/PRODUCT_ID
10	/SCMTMS/TRQ_TYPE	TR: Type	/SCMTMS/TRQ_TYPE

**Business Object Based Data Access Definition**

Name of BO Used in Condition: /SCMTMS/TRQ  
Name of BO Node Used in Condition: ROOT  
Name of the Field of the BO Node: TRQ\_TYPE

Filter Definition for Data Access Definition: Key Field Value for BO Node Identif.:  
Second Filter: Key Field Value for BO Node Identif.:

**Data Crawler-Based Data Access Definition**

DC Profile ID:  
DC Step ID: 000  
Field Name:

**Class-Based Data Access Definition**

Determination Class:  
Attributes for Classes:

Picture: Specifying the relevant Data Access Definitions.

- 3) The next step is to define the input values that shall be used by the new condition. Click on button *Data Access Definition* (see picture above) to define a subset of data access definitions allowed for the chosen condition type that shall be used as input for the new condition. You can choose any data access definition that was assigned to the used condition type in customizing before. Example:

Data Access Definition for Condition	Comment
/SCMTMS/TRQ_TYPE	Forwarding Order Type
/SCMTMS/TRQ_ITEM_PRD	Forwarding Order Item - Product ID

On the same screen you can also adjust and add further details for the currently selected data access definition if required. Click on button *Back* to return to the main screen.

- 4) Back on the main screen enter the decision table entries for the condition. With this decision table, input values are mapped onto corresponding output values of the condition. Example:

Input values as per the used data access definitions.				Output values as defined in the used condition type.
Comparison Operation	Type	Comparison Operation	Product	FUB Rule
Is equal to	FWO	Contains String	DIT-PROD-10-1	A-FUBR-01
Is equal to	FWO	Contains String	DIT-PROD-10-2	A-FUBR-02
Is equal to	TRO1	Contains String	DIT-PROD-10-3	A-FUBR-04
...	...	...	...	...

The screenshot shows the 'Table Contents' screen in SAP. It features a table with columns for 'TR: Type', 'TR Item: Product', and 'FUB Rule'. The table contains three rows of data. Annotations with callouts provide instructions: 1) 'Add a new entry to the decision table of the new condition.' points to the 'Add' button. 2) 'Click to create a new column entry.' points to the '...' button in the table. 3) 'Enter the details of a column here. You can define single values, ranges, exclude certain values, etc.' points to the input field for 'TR: Type'. 4) 'Click on Button OK to take over the entered value.' points to the 'OK' button. Other callouts explain that the first two columns represent input and the third column represents output (FUB Rule).

Picture: Maintaining the content of the decision table.

- 5) Save the defined condition. The condition is now ready to be used. The described example condition can be used in the customizing for Forwarding Order Types to define its way how to determine a Freight Unit Building. The Freight Unit building rule is determined when creating a Forwarding Order of a corresponding type (note: in case you enter a Freight Unit Building Rule on the Forwarding Order UI, the system is implemented in a way that this will override the rule determined by the condition).



**Change View "Forwarding Order Types": Details**

BC Set: Change Field Values

Forwarding Order Type: FWO

Do NOT Change, Forwarding Order

Default Type for Category: ☒

Number Range Settings

Number Range Interval: 03 Template No. Range Interval: 04

Default Values

Default Weight UoM: KG Default Volume UoM: M3 Default Pieces UoM: PC Default Alt. Qty UoM:

FU Building Rule Condition: DEMO\_FUBR\_COND

Freight Unit Building Rule:

Planning Profile: BM-PF

Web Dynpro Configuration:

Default Charges View:

Picture: The condition in customizing for Forwarding Order Type.

### 4.3.6 Simulating Conditions

Before using a newly created condition the function that it is supposed to realize can be simulated. This allows verifying the expected functional correctness in advance of using the condition in a production environment. This can help to prevent inconsistencies and unwanted results. As per TM 9.0, the simulation of conditions can be triggered right away from the Conditions User Interface integrated in the TM User Interface.

- 1) The simulation can be started on the main screen of the Conditions User Interface. Click on button *Start Simulation*.

Decision Table: DEMO\_FUBR\_COND

Additional Actions Context Overview Start Simulation

Click here to start simulating the condition, e.g. already while creating it.

Picture: Starting the simulation on the Condition Main Screen.

- 2) On the next screen just click on button *Continue* (usually, the only option to define is whether the last active version of the condition or a version as of a specific Date/Time). Then you can enter example input values for the condition. Choose e.g. a combination of values that has an entry in the related decision table. Then click on button *Execute* to start the simulation.

Simulation

Business Rule Framework plus - Simulation

Continue Cancel

Simulation Mode: 1) Click to continue.

Action Settings: Execute

Object Type: Function Expression Action

Object: Name: DEMO\_FUBR\_COND Description:

Version: Last Active Latest Version As of Date

Simulation

Business Rule Framework plus - Simulation

Back to Selection Execute Execute and Display Processing Steps

Selected Object

Expression Name: DEMO\_FUBR\_COND Description:

Context Values

Import Test Data

TR Item: Product DIT-PROD-10-1

TR: Type TR01

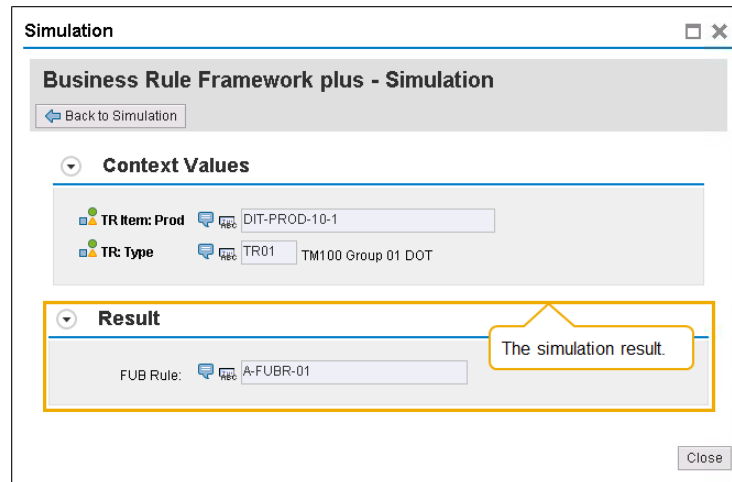
2) Enter example input values for the condition.

3) Click to start the simulation.

Close

Picture: Simulation of a condition.

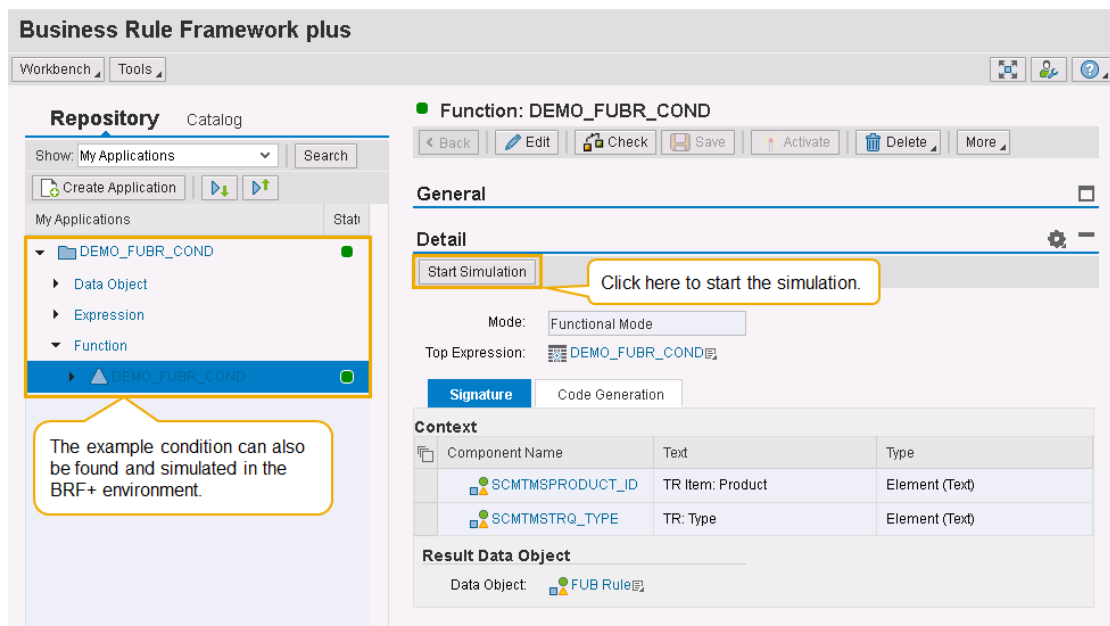
- 3) On the next screen you can see the result of the simulation (in the example a corresponding Freight Unit Building Rule was found for the entered Item Product and Forwarding Order Type).



Picture: Condition Simulation result.

Alternatively you can also use the BRF+ Framework directly to maintain and simulate your conditions.

- 1) Start transaction *BRF+*. The Business Rule Framework Plus (BRF+) is the technical framework that is used within Transportation Management to handle conditions.



Picture: The BRF+ User Interface.

- 2) Click on tab *Repository* and in selection box Show choose *My Applications*. You can find here a so called BRF+ application that contains the condition you created as a function. Navigate to your condition (Example: *DEMO\_FUBR\_COND*) in the Functions sub tree and double click on it. If the selected function is in status *Active* and already signed with a green traffic light, the condition is already active and can be used.
- 3) On the right side of the screen you can now see the details of the condition as represented in the BRF+ framework. If the selected function does not yet show status *Active*, you can click on button *Activate* to activate the condition. After activating, the

condition should be signed with a green traffic light. The active condition is now ready to be simulated or executed.

- 4) Click on button *Start Simulation* to get to the simulation screen. On the simulation screen you can enter a set of input values that will be used for the simulation run. The fields to be entered correspond to the input fields defined in the condition via its assigned data access definitions.

The following screens for the simulation are exactly those that have been shown already above, i.e. the SAP Transportation Management User Interface just simply integrated the BRF+ User Interface parts for simulation of conditions.

### 4.3.7 Implementing a condition call in your coding

To see the execution of a condition, you can set a breakpoint in method *PROC\_CONDITIONS* of class */SCMTMS/CL\_COND\_OL*. This class provides a variety of methods that help to implement condition calls. Especially, it helps to abstract from the more complex coding which is required to call BRF+ (Business Rules Framework +) which is used for the realization of conditions in SAP TM.

The following example report implements the invocation of a condition for a given business object instance. It calls the example condition created in section 4.3.5, i.e. the determination of a Freight Unit Building rule for a given Forwarding Order instance. The condition must be assigned to the Forwarding Order Type as also described in section 4.3.5.

```
*&-----*
*& Report  ZREP_COND_TEST
*&-----*
*& This report demonstrates the usage of class /SCMTMS/CL_COND_OL,
*& method PROC_CONDITIONS to execute conditions and receive back a
*& result.
*&-----*
REPORT  zrep_cond_test.

DATA: ls_bo_inst_key      TYPE /bobf/s_frw_key,
      lt_bo_inst_key      TYPE /bobf/t_frw_key,
      ls_condition_id     TYPE /scmtms/s_condition_id,
      lt_condition_id     TYPE /scmtms/t_condition_id,
      co_message          TYPE REF TO /bobf/if_frw_message,
      lt_cond_result      TYPE /scmtms/t_boid_cond_result,
      lt_cond_result_all  TYPE /scmtms/t_boid_cond_result.

CLEAR: ls_bo_inst_key,
       ls_bo_inst_key,
       ls_condition_id,
       lt_condition_id.

BREAK-POINT.

* Specify the key of an example BO instance (here: a TRQ instance)
ls_bo_inst_key-key = '4D08C62BC9015E16E10000000A421A6A'.
APPEND ls_bo_inst_key TO lt_bo_inst_key.

* Specify the condition to be executed
ls_condition_id-condition_id = 'DEMO_FUBR_COND'.
APPEND ls_condition_id TO lt_condition_id.

* Call method PROC_CONDITIONS
CALL METHOD /scmtms/cl_cond_ol=>proc_conditions
  EXPORTING
```

```

        it_boinst_key          = lt_bo_inst_key
        it_cond_id            = lt_condition_id
        iv_do_not_return_no_hits = abap_true
IMPORTING
    et_bokey_cond_result      = lt_cond_result
CHANGING
    co_message                = co_message.

* Collect all results from condition for later processing
INSERT LINES OF lt_cond_result INTO TABLE lt_cond_result_all.

BREAK-POINT.

```

The report also allows debugging the processing of a condition to learn more details on how it is actually executed during runtime. During execution, jump into method *PROC\_CONDITION* and set further breakpoints to see the different parts of the condition processing. For this, the method calls further methods defined in class */SCMTMS/CL\_COND\_OL*.

## 4.4 Change Controller

The Change Controller is a framework used in the context of SAP TM to dynamically react on changes that are done on Business Documents like Freight Orders, Freight Bookings, Freight Units or Service Orders. They are all based on the technical BO /SCMTMS/TOR. The Change Controller allows detecting changes on these Business Documents (Objects) and defining how the system should react on these changes. It triggers functionality that reacts on the changes, e.g. executing updates or changes on other related Business Documents (Objects) or checking tolerances for changed data.

You will see in the following sections that the change controller uses Process Controller Strategies (section 4.2) as well as Conditions (section 4.3) to realize its functionality. So it is recommended to get familiar with these two sections in advance before going on with this section.

The following sections describe how the Change Controller works from a technical perspective along with examples how to set it up. Moreover an enhancement concept is described that can be used to react on customer / partner specific changes in the mentioned BO /SCMTMS/TOR.

### 4.4.1 Basic Concept & technical aspects

The basic concept of the Change Controller is described in the following steps:

- 1) An instance of BO /SCMTMS/TOR (as mentioned, this can represent a Freight Order, Freight Booking, Freight Unit or Service Order) is changed, i.e. attributes of its so called triggering nodes are created, updated or deleted. The Change controller is able to react on changes in the data of these nodes. The triggering nodes are:
  - ROOT
  - ITEM\_TR
  - STOP
  - EXECUTIONINFORMATION
  - HANDLING\_CODE
  - CC\_CHG\_TR
- 2) The Determination *DET\_CALL\_CHACO* is called which is assigned to the Root Node of BO /SCMTMS/TOR. The following table shows the Triggering Conditions for the mentioned nodes (transactional point in time will be *BEFORE\_SAVE*).

Node	Create	Update	Delete
ROOT	No	Yes	No
ITEM_TR	Yes	Yes	No
STOP	Yes	Yes	Yes
EXECUTIONINFORMATION	Yes	Yes	No
HANDLING_CODE	Yes	Yes	Yes
CC_CHG_TR	Yes	Yes	No

- 3) Get the /SCMTMS/TOR nodes that triggered the changes and identify the discrete changes.
- 4) The Determination *DET\_TRIGGER\_STRATEGIES* is executed to determine the Change Strategy to be executed. Such a Change Strategy is finally a Process Controller Strategy (see section 4.2) that gets executed.

The found Change Strategy can be configured to be executed synchronously or asynchronously. It contains the execution of the functionality that represents the required reaction on changes in Business Documents based on the technical BO /SCMTMS/TOR.

- 5) In case of synchronous configuration it will be directly executed on triggering condition and transactional point in time *BEFORE\_SAVE*.
- 6) In case of an asynchronous configuration, the Change Strategy is first of all registered for asynchronous execution on triggering condition and transactional point in time *BEFORE\_SAVE*. The actual execution is then done via corresponding function modules that are called in update-task. Note: In earlier SAP TM releases before SAP TM 9.0 they were executed on triggering condition and transactional point in time *AFTER\_COMMIT*. This was replaced due to performance issues.

## 4.4.2 Customizing settings for the Change Controller

For each of the mentioned Business Documents you can maintain corresponding Document Type in transaction SPRO, path *SAP Transportation Management → Transportation Management → Freight Order Management → Freight Order (Freight Booking, Transportation Unit or Service Order)*. Freight Unit can be found under path *SAP Transportation Management → Transportation Management → Planning → Freight Unit*. In all mentioned document types you can define the following settings:

1. Default Change Strategy:  
This change strategy defines the default process controller strategy ("change controller strategy") that the change controller will use to react on changes to the business document. The default strategy is used in case no strategy determination condition is assigned to the business document type. It is also used in case an assigned strategy determination condition could not determine a suitable change controller strategy at runtime.
- Change Strategy Determination Condition:  
This condition specifies the condition that the system will use for determining a change controller strategy at runtime. If you need to use different change controller strategies depending on specific situations you can set up this condition for determining the correct change controller strategy at runtime. The condition type of this condition must be */SCMTMS/CC\_TOR\_STRAT*.
- Quantity Tolerance Condition:  
Here you can define a condition for determining quantity tolerances in the case of a quantity change. If you change a quantity the condition is used to dynamically check whether a quantity change can be tolerated. In case no condition is maintained, the standard logic will just classify any quantity change as a relevant quantity change. The condition type of this condition must be */SCMTMS/CC\_QUAN\_TOL*.

The result of this condition can be:

- "" = No Relevant Quantity Change Determined.
- "X" = Relevant Quantity Change Determined.

This result is then further provided to the change controller condition via data access definition */SCMTMS/TOR\_QUAN\_UPD*. The change controller condition can then decide which strategy to execute to react on a quantity change in case it is a relevant change outside the defined tolerances.

Example: Assume you change the quantity of a Forwarding Order for which a Freight Unit has been created before from 2.500 kg to 2.510 kg (Remember: Freight Units are instances of Business Object */SCMTMS/TOR*). This quantity change in the Forwarding Order will trigger the change controller strategy assigned to the Freight Unit Type of the related Freight Unit.

- In case no quantity tolerance condition is maintained in the Freight Unit Type, the quantity change will be considered as relevant (standard logic) and the Freight Unit weight is adjusted also to 2.510 kg.

- In case there is a quantity tolerance condition maintained in the Freight Unit Type, the tolerance will be determined at runtime according to the decision table defined for the condition.

The following standard Data Access Definitions are available for defining a quantity tolerance condition:

- /SCMTMS/TORQTYCHGWEI:  
Provides the quantity change of weight in kg.
- /SCMTMS/TORQTYCHGVOL:  
Provides the quantity change of volume in m3.
- /SCMTMS/TORQTYCHGPCS:  
Provides the quantity change of pieces.
- /SCMTMS/TOR\_UTIL\_CHG:  
Provides the maximum change of the utilization rate (not feasible for freight units).

These standard Data Access Definitions return negative values in case of quantity decreases and positive values in case of quantity increases. Using data access definition /SCMTMS/TORQTYCHGWEI, the condition can be e.g. set up to consider a quantity change of only 10 kg as not relevant. In the example mentioned above, the Freight unit would then keep its original weight of 2.500 kg.

- **Date Tolerance Condition:**  
This condition can be used for determining date/time tolerances in case of a date/time change. In case no condition is maintained, the standard logic will just classify any date/time change as critical changes. The condition type of this condition must be /SCMTMS/CC\_DATE\_TOL. The result of a maintained Date Tolerance Condition can be one of the following values:
  - " " = No Change.
  - "1" = Critical Change.
  - "2" = Non-critical Change.

Similar to the Quantity Tolerance Condition, the result is then further provided to the change controller condition via data access definition /SCMTMS/TOR\_DATE\_UPD. The change controller condition can then decide which strategy to execute to react on a date/time change in case it is a critical change outside the defined tolerances.

#### 4.4.3 Example Change Controller settings

The first example describes a simple demo setup for using the change controller to react on changes done for Freight Orders of a specific type.

- 1) Create an example Freight Order Type ZCCD in the TM customizing. Path: *SAP Transportation Management* → *Transportation Management* → *Freight Order Management* → *Define Freight Order Types* (you can copy an existing Freight Order Type and name it accordingly).
- 2) Create a new condition ZENH\_QUAN\_TOL\_DET with the following parameters:

Condition	: ZENH_QUAN_TOL_DET
Description	: Enh. Quantity Tolerance Det. Condition
Condition Type	: /SCMTMS/CC_QUAN_TOL
Origin of Condition	: Condition Based on BRFplus Decision Table



Define the following content for the decision table of the condition:

Quantity Change of Weight in default weight unit of measure	Qty Change
> 50	X (Relevant Quantity Change)
<=50	False

Save and simulate the new condition. It will consider a quantity change (in this case a quantity increase) of more than 50 kg as a relevant quantity change that the change controller shall later react on. Quantity changes below 50 kg will be considered as not relevant. The result of this Quantity Tolerance condition will be used in the following for the determination of a suitable change controller strategy.

- 3) Create a new condition *ZENH\_DATE\_TOL\_DET* with the following parameters:

Condition : *ZENH\_DATE\_TOL\_DET*  
Description : Enh. Date Tolerance Det. Condition  
Condition Type : */SCMTMS/CC\_DATE\_TOL*  
Origin of Condition : Condition Based on BRFplus Decision Table

Define the following content for the decision table of the condition:

Maximum Delta of Date Changes	Date Chg.
Is between 0 and 4000	“ “
Is between 4001 and 8000	2 (Non-Critical Change)
> 8000	1 (Critical Change)

Save and simulate the new condition. It will consider a date change (in this case a time increase) of more than 8.000 minutes as a critical date (time) change that the change controller shall later react on. Date changes between 4.001 and 8.000 minutes will be considered non-critical changes. Changes between 0 and 4.000 minutes are considered as no change. The result of this Date Tolerance condition will be used in the following for the determination of a suitable change controller strategy.

- 4) Follow the IMG path to the Process Controller *SAP Transportation Management* → *SCM Basis* → *Process Controller* and define the Change (Process) Controller Strategies *ZENH\_CHAC1* – *ZENH\_CHAC7* each of service type *TOR\_CHACO* (the type for asynchronous processing). For each strategy n (n = {1, ... , 7} assign the following standard methods (they just serve as a demonstration and you can of course play around with it and define your own sequences of methods for each strategy):

<i>ZENH_CHACn</i>	Change Strategy n	DEF_REACT	Def. reaction to date, location, quantity changes
<i>ZENH_CHACn</i>	Change Strategy n	CHECK_CAPA	Check Capacities
<i>ZENH_CHACn</i>	Change Strategy n	FIX_TOR	Fix Freight Document

- 5) Create a new condition *ZENH\_CC\_DET* with the following parameters:

Condition : *ZENH\_CC\_DET*  
Description : Enh. Change Cont. Strategy Det. Condition  
Condition Type : */SCMTMS/CC\_TOR\_STRAT*  
Origin of Condition : Condition Based on BRFplus Decision Table

Define the following content for the decision table of the condition:

Indicates whether a quantity has been changed for this TO	TO Type	Indicates whether a date change has happened and classifies this	CC Strat.
False	ZCCD	" "	DEF_CHACO
False	ZCCD	1 (Critical Change)	ZENH_CHAC1
False	ZCCD	2 (Non-Critical Change)	ZENH_CHAC2
X (Relevant Quantity Change Determined)	ZCCD	" "	ZENH_CHAC3
X (Relevant Quantity Change Determined)	ZCCD	1 (Critical Change)	ZENH_CHAC4
X (Relevant Quantity Change Determined)	ZCCD	2 (Non-Critical Change)	ZENH_CHAC5

Save and simulate the new condition. It will consider a date change (in this case a time increase) of more than 8.000 minutes as a critical date (time) change that the change controller shall later react on. Date changes between 4.001 and 8.000 minutes will be considered non-critical changes. Changes between 0 and 4.000 minutes are considered as no change. The result of this Date Tolerance condition will be used in the following for the determination of a suitable change controller strategy.

- 6) Use the created conditions in the customizing settings for the Freight Order Type ZCCD from step 1) to make the following entries in the required fields:

Default Change Strategy	DEF_CHACO
Change Strategy Det. Cond.	ZENH_CC_DET
Quantity Tolerance Cond.	ZENH_QUAN_TOL_DET
Date Tolerance Condition	ZENH_DATE_TOL_DET

With these settings, changes for Freight Orders of type *ZCCD* will use the standard Change Strategy *DEF\_CHACO* as the default strategy. Condition *ZENH\_CC\_DET* will be used to determine a Change Strategy depending on the types of changes and related entries in its decision table. The other two conditions that we have created will be used to determine the relevance of quantity and date/time changes by checking the configured tolerances.

**Display Enh. Change Controller Strategy Det. 6100002255**

Business Context Viewer

Save Cancel Edit Check Follow Up Scheduling Subcontracting

Create Service Cancel Document

Example: 2) After adjusting the Gross Weight the Freight Order was saved.

✓ Data saved successfully

⚠ Messages for Freight Order 6100002255

✓ Change strategy "ZENH\_CHAC3" registered for asynchronous processing

Display Message Log

General Data **Cargo**

Current Location and Date/Time

Location: Date: 00:00:00

Insert Product Insert FUs Based on Freight Unit ID Set to Loaded Set to Initial

Report Discrepancies Create Forwarding Order Change Hierarchy: Cargo Management for Current Location

Item Hierarchy	I...	C...	P...	P...	M...	Gross ...	U...	Actu...	U...	Gr...	U...	Act...	U...	Qu...	U...	Act...
Finishe...	10			F...		900.00	KG			2,100	M3			1	PC	
Produc...	20			F...		500.00	KG			500	M3			1	PC	

Example: 1) In a Freight Order the Gross Weight if this item was adjusted from 645 KG to 900 KG.

Example: 3) The Freight Order is of type ZCCD and the assigned the related setup of the change controller found Change Strategy ZENH\_CHAC3 that the system has registered for asynchronous processing.

Picture: A simple Quantity Change example.

In the example mentioned above a quantity change was done for a Freight Order of type ZCCD, i.e. the gross weight of the first item was changed from e.g. 645 KG to 900 KG. According to condition ZENH\_QUAN\_TOL\_DET created in step 2 this change is considered a relevant quantity change. The result of this condition is then passed to the Change Controller Strategy Condition ZENH\_CC\_DET. In the example there was just a quantity change but no date (time) change detected. This situation matches line 3 in the decision table of condition ZENH\_CC\_DET that leads to the usage of Change Strategy ZENH\_CHAC3 (just like shown in the example above).

The second example is based on the first one and describes a simple demo setup for using the change controller to react on quantity changes done for a Forwarding Order (instance of BO TRQ) that has unplanned unfixed Freight Units (instances of BO TOR) assigned. The Freight Units shall be adjusted to the new quantity entered in the Forwarding Order.

- 1) Create an example Freight Unit Type ZCCU in the TM customizing. IMG Path: *SAP Transportation Management → Transportation Management → Planning → Freight Unit → Define Freight Unit Types* (you can copy an existing Freight Unit Type and name it accordingly).
- 2) Enhance the decision table of Condition ZENH\_CC\_DET from step 5) of the first example with the following entry:

X (Relevant Quantity Change Determined)	ZCCU	“ “	ZENH_CHAC7
---	------	-----	------------

- 3) Follow the IMG path to the Process Controller *SAP Transportation Management → SCM Basis → Process Controller* and define the Change (Process) Controller Strategy ZENH\_CHAC7. Assign the following standard method to the strategy:

ZENH_CHAC7	Change Strategy 7	REBUILD_FU	Rebuilds unplanned, unfixed FUs
------------	-------------------	------------	---------------------------------

This strategy will execute standard method *REBUILD\_FU* for rebuilding Freight Units based on quantity changes done for the related Forwarding Order from where the Freight Units were created.

- 4) Use the created conditions in the customizing settings for the Freight Unit Type ZCCU from step 1) to make the following entries in the required fields:

Default Change Strategy	DEF_CHACO
Change Strategy Det. Cond.	ZENH_CC_DET
Quantity Tolerance Cond.	ZENH_QUAN_TOL_DET
Date Tolerance Condition	ZENH_DATE_TOL_DET

With these settings, changes for Freight Units of type ZCCU will use the standard Change Strategy *DEF\_CHACO* as the default strategy. Condition *ZENH\_CC\_DET* will be used to determine a Change Strategy depending on the types of changes and related entries in its decision table. The other two conditions that we have created will be used to determine the relevance of quantity and date/time changes by checking the configured tolerances. In this specific example strategy *ZENH\_CHAC7* will be found and executed when a quantity change is done to the Forwarding Order that a Freight Unit of type ZCCU relates to.

- 5) Follow the menu path *Application Administration → Planning → (General Settings →) Freight Unit Building Rule*. Create a Freight Unit Building Rule *ZCCU\_FUBR* with document type ZCCU that was created in step 1). Use the following settings:

Freight Unit Building Rule	ZCCU_FUBR
----------------------------	-----------

Description	Enhancement Freight Unit Building Rule
Document Type	ZCCU
Incompatibility Settings	[space]
Freight Unit Building Strategy	Consolidate per Request (Compatible Parts)
Critical Quantity	Gross Weight

On tab strip *Advanced Settings* choose *FUBR\_AUTO* as the Process Controller Strategy. On tab strip *Planning Quantities* make the following entry:

Planning Quantity for Freight Unit Building	Gross Weight
Unit of Measure for Split Quantity	KG
Split Quantity	1.000

- 6) Follow IMG path *SAP Transportation Management* → *Transportation Management* → *Forwarding Order Management* → *Forwarding Order* → *Define Forwarding Order Types* and define a new Forwarding Order Type ZFWO.

You can copy e.g. an existing Forwarding Order Type, name it ZFWO and assign Freight Unit Building Rule ZCCU\_FUBR to it. Moreover set the flag *Automatic Freight Unit Building* so that Freight Units are built when saving a newly created Forwarding Order of this type.

- 7) You can now create a simple Forwarding Order of type ZFWO with e.g. a single product item that contains a Gross Weight Quantity of e.g. 2.570 KG. Save the Forwarding Order and take a look at the created Freight Units via tab strip *Document Flow*. With the given example configuration you should receive exactly one Freight Unit that carries the complete quantity.
- 8) Edit the example Forwarding Order and adjust the Item Gross Weight Quantity to e.g. 3.240 KG. Save this change and display (or refresh the eventually still open UI for your example Freight Unit) the Freight Unit again. The Change Controller Strategy found for the Freight Unit has detected the quantity change in the underlying Forwarding Order and adjusted the Freight Unit quantity accordingly.

2) Save the change done for example Forwarding Order 2100001952. This will trigger the Change Controller Strategy defined for the related Freight Unit Type.

1) Change the Gross Weight to e.g. 3.240 KG in the example Forwarding Order 2100001952.

3) The example Freight Unit 4100021160 that was created from example Forwarding Order 2100001952 got updated with the new quantity.

Picture: A quantity change in a Forwarding Order Item updated the related Freight Unit.

#### 4.4.4 The Change Controller and how it works at runtime

When a *TOR* BO instance is changed, the change controller functionality is started via Determination *DET\_CALL\_CHACO* which is registered on transactional point in time *BEFORE\_SAVE (Finalize)*.

Within this determination Action *FILL\_TRANS\_CHANGEINFO* is called that executes the following tasks:

- Date and Location changes are determined based on the evaluation of the configured Date Tolerance Condition.
- Quantity Changes are determined using the Quantity Tolerance Condition.
- New execution events are determined.
- The transient node *CC\_CHG\_TR* is filled with the information about the identified changes. Here you can find the types as well as the related values of changes.
- Customer/Partner enhancements can be determined (see also next section).

In the following picture the structure of node *CC\_CHG\_TR* is shown with some remarks on the semantics of different groups of attributes.

Component	Typing Method	Component Type	Data Type	Length	Decl...	Short Description
CHG_QUANTITY	Types					less Object Transportation Order
CHG_DATE	Types					Object Transportation Order
CHG_LOCATION	Types	✓ /SCMTMS/TOR_LOC_	CHAR	1		0 Location Change in Business Object Transportation Order
EXEC_TOR_KEY	Types					0 NodeID
EXEC_TOR_EVENT	Types					0 Event Occurring for a Transportation Activity
FORCE_CHACO	Types					
FORCE_VALIDATE	Types					
TRQ_CTX_FU_ADD	Types	✓ /SCMTMS/TOR_TRQ_	CHAR	1		0 Freight Unit Added in Transportation Request Context
TRQ_CTX_FU_DEL	Types					0 Freight Unit Deleted in Transportation Request Context
TRQ_CTX_LOC_ADD	Types					0 Location Added in Transportation Request Context
TRQ_CTX_LOC_CHG	Types	✓ /SCMTMS/TOR_TRQ_	CHAR	1		0 Location Changed in Transportation Request Context
TRQ_CTX_LOC_DEL	Types	✓ /SCMTMS/TOR_TRQ_	CHAR	1		0 Location Deleted in Transportation Request Context
TRQ_CTX_QUAN_CHG	Types	✓ /SCMTMS/TOR_TRQ_	CHAR	1		0 Quantity Change in Freight Unit in Transp. Request Context
CHG_DATE_MAX_DE	Types	✓ /SCMTMS/DATE_CH	INT4	10		0 Delta of Date Change in Minutes
CHG_QUAN_MAX_DE	Types					0 Delta of Weight Change
CHG_QUAN_MAX_DE	Types	✓ /SCMTMS/PCS_CHA_	INT4	10		0 Delta of Volume Change
CHG_QUAN_MAX_DE	Types	✓ /SCMTMS/PCS_CHA_	INT4	10		0 Delta of Quantity Change
CHG_QUAN_MAX_DE	Types	✓ /SCMTMS/PCS_CHA_	UNIT	3		0 Unit of Measure for Delta of Quantity Change
CHG_UTIL_DELTA	Types	✓ /SCMTMS/UTIL_CH_	INT4	10		0 Delta of Utilization Change
CHG_SERVICE_LEV	Types	✓ /SCMTMS/TOR_SER_	CHAR	1		0 Service Level Change in Business Object Transportation Order
CHG_ITEM	Types	✓ /SCMTMS/TOR_ITE_	CHAR	1		0 Item Change in Business Object Transportation Order
CHG_SUBCONTRACT	Types	✓ /SCMTMS/TOR_SUB_	CHAR	1		0 Subcontracting-Relevant Change in Transportation Order
CHG_DG	Types	✓ /SCMTMS/TOR_ITE_	CHAR	1		0 Change of Dangerous Goods Checkbox
CHG_HC	Types	✓ /SCMTMS/TOR_HC_	CHAR	1		0 Change of Handling Codes
.INCLUDE	Types					0 Enhancement Structure for Change Info
EEW_TOR_CCCHG	Types					0 Enhancement Structure for Change Info

Picture: Data structure */SCMTMS/S\_TOR\_CCCHG* of node *CC\_CHG\_TR*.

In the next step the Change Controller Strategy is determined. For this, the *TOR Type Customizing* is read and the maintained Change Strategy Determination Condition is evaluated. If the condition was not maintained in customizing or it does not return a result the maintained default Change Strategy will be used for further execution. The default Change Strategy is only executed if there is actually a change determined and registered in transient node *CC\_CHG\_TR*.

Moreover, all keys of changed *TOR* instances are grouped by the found strategy as well as the changes identified in transient node *CC\_CHG\_TR* (→ mass execution enablement, i.e. for a group of changed *TOR* instances several different strategies as well as different kind of changes may be detected).

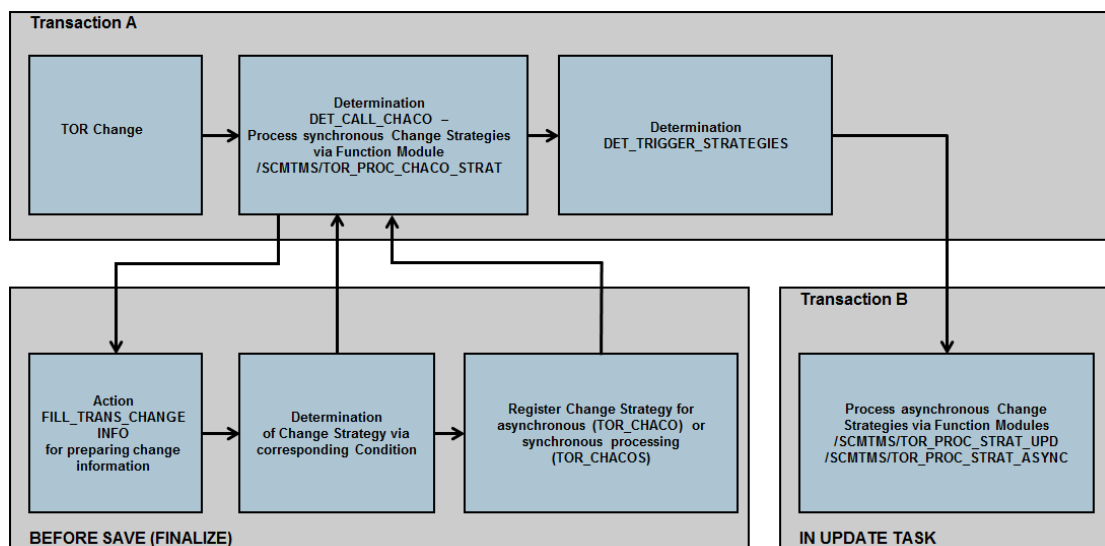
The attributes of transient node `CC_CHG_TR` are provided via Data Access Definitions to evaluate the decision table of the Change Strategy Determination condition. The condition then returns the corresponding strategy to be executed if the provided attributes uniquely match an entry in the decision table.

The found strategy is either configured to be executed synchronously (service type `TOR_CHACOS`) or asynchronously (service type `TOR_CHACO`).

Strategies of service type `TOR_CHACOS` are processed synchronously by calling Function Module `/SCMTMS/TOR_PROC_CHACO_STRAT`. You should use synchronous strategies with a bit care. Not only has the user (the current transaction) to wait for the strategy to be finally executed before the next step can be processed (→ performance) but you should also keep in mind that the BOPF Determination-Validation-Cycle is not executed for the changes made by the executed strategy.

Strategies of service type `TOR_CHACO` are registered for asynchronous processing. In Determination `DET_TRIGGER_STRATEGIES`, the registered strategies are then executed by calling Function Module `/SCMTMS/TOR_PROC_STRAT_UPD` in *Update Task*. This function module in turn calls Function Module `/SCMTMS/TOR_PROC_STRAT_ASYNC` in *Background Task*. Together they represent a separate transaction and in this case also the BOPF Determination-Validation-Cycle is executed for the changes made by the executed strategy.

The following picture provides a schematic overview and summary of the functional blocks described in this section.



Picture: A rough picture of how the change controller works at runtime.

#### 4.4.5 Enhancing the Change Controller

In the previous section we could see that there are already quite some attributes available in the data structure `/SCMTMS/S_TOR_CCCHG` of node `CC_CHG_TR` that represent different kinds of changes. But customers and partners may have to react on different or additional changes.

1. The BAdI `/SCMTMS/TOR_CHACO_CHANGES_DET` allows enhancing the standard logic for identifying changes in TOR BO instances. It provides the following methods that can be used:
  - **DET\_DATE\_AND\_LOC\_CHANGES:** Determine Date and Location Changes. The method allows enhancing or even replacing the standard logic for identifying and classifying date and location changes.



- **DET\_QUANTITY\_CHANGES:** Determine Quantity Changes. The method allows enhancing or even replacing the standard logic for identifying and classifying quantity changes.
  - **DET\_CUSTOM\_CHANGES:** Determine Customer-Defined Changes. Customer and partner-specific changes can be determined and stored in extension fields of the transient node *CC\_CHG\_TR*. These extension fields can be added to the node via its extension include *EEW\_TOR\_CCCHG*.
2. The reaction on changes can be enhanced with the following means that make use of functions and features provided with conditions and process controller strategies. These concepts have been introduced in the previous sections and are used here as well in the context of the Change Controller:
- Customer/partner specific Data Access Definitions can be added in customizing for providing access to extension fields added to transient node *CC\_CHG\_TR*. These additional Data Access Definitions can then be used to define the Change Controller related conditions for determination of date and quantity tolerances as well as finding an appropriate Change Strategy.
  - Additional customer/partner specific process controller strategy methods of service type *TOR\_CHACO* or *TOR\_CHACOS* can be created.
  - The standard as well as the customer/partner specific methods can be combined into new change (process) controller strategies of service type *TOR\_CHACO* or *TOR\_CHACOS*.
3. You can use the Change Controller to raise events, e.g. that the execution of a Freight Order will be delayed. The system can then e.g. send a mail with a related alert message to the responsible user that can then react accordingly.
- An example is the standard Change Controller Strategy method *TOR\_DELAY*. It is implemented in the standard class */SCMTMS/CL\_CHACO\_METHODS* method *HANDLE\_DELAY\_FROM\_EXECUTION*.
  - This method fills the internal table *MT\_ALERT\_CAT\_KEYS\_MESS* of the Change Controller Request Object with the TOR keys per alert category and related messages.
  - If you implement your own Change Controller Strategy methods that shall be able to check customer/partner specific changes and to raise events depending on the determined changes, you have to make sure that it fills the mentioned internal table as described, i.e. the events are “registered” in this internal table.
  - You can then add such methods to your Change controller Strategies. At runtime they will check changes and add related events where required.
  - To finally trigger the events that have been registered in internal table *MT\_ALERT\_CAT\_KEYS\_MESS* you have to add the standard Change Controller Strategy method *CHACOALERT* after the event raising methods in the sequence of strategy methods. This method is implemented in the standard class */SCMTMS/CL\_CHACO\_METHODS* method *CHACO\_CREATE\_ALERT*. It is used to raise events from a change strategy.
  - Raised alerts can be displayed in the Alert Inbox of the corresponding Users that are responsible to react on the different alerts.



### 4.4.6 The Trigger Concept

The Trigger Concept of the Change Controller allows actions to finish successfully even in cases where not all operations could be executed due to locking issues. An example use case looks as follows:

1. A quantity update on a Forwarding Order (*TRQ*) leads to a quantity change on a related Freight Unit.
2. Change Strategy *START\_TEND* is found to restart the tendering process.
3. But the assigned Freight Order is locked (assume the Freight Unit is already assigned to a Freight Unit). The current tendering process cannot be stopped and restarted immediately.
4. The trigger *TOR\_APPLY\_CHACO\_STRATEGY* is set to process the Change Strategy *START\_TEND* again at a later point in time.
5. Report */SCMTMS/PROCESS\_TRIGGER\_BGD* is used to reprocess the triggers periodically until the Change Strategy *START\_TEND* can be finished successfully. For this the mentioned report should be scheduled to run periodically.

There are different triggers provided with the SAP TM Standard and defined in a Trigger Registry Table (system table */SCMTMS/I\_TRIG*), i.e. in this table triggers with different semantics are defined. In general there are two types of triggers distinguished: Triggers set by Action Calls and triggers set by Function Module Calls.

TRIGGER_ID	TRIGGER_CLASS
<input type="checkbox"/> TOR_ADAPT_LOAD_TIME_TO_TRANS	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_APPLY_CHACO_STRATEGY	/SCMTMS/CL_TRIG_CONTROL_FM
<input type="checkbox"/> TOR_CANCEL	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_CHECK_AND_BLOCK	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_CHECK_DG	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_DELETE_DRAY_ITEM	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_DET_ORG_INTERACTION	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_DSO_STRATEGY	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_ITEM_SET_CARGO_RECEIPT	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_LC_DETERMINE	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_PLAN_STATUS	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_PROCESS_EXEC_INFO	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_ROOT_DET_SHIP_CONS	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_SET_EXECUTION_STATUS	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_STOP_PROPAGATE_TIMES	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_STOP_UPDATE_FOLLOW_UP	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_SUCC_TRQ_SETT_READY	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_UNASSIGN_STOP	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_UPDATE_FOLLOW_UP	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_UPDATE_FROM_CAPATOR	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_UPDATE_SCHED_REF_DATA_STA	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_UPD_CROSS_DOC_CHECK	/SCMTMS/CL_TRIG_CONTROL_ACTION
<input type="checkbox"/> TOR_UPD_FROM_SCHEDULE	/SCMTMS/CL_TRIG_CONTROL_ACTION

A Trigger set by a Function Module Call. The Trigger class provides methods to set the trigger and identify the triggering context, e.g. the BO, BO node, instance key, etc.

A Trigger set by an Action Call with the related Trigger Class.

Picture: Trigger Registry Table */SCMTMS/I\_TRIG*.

The helper class */SCMTMS/CL\_TRIG\_HELPER* provides methods that allow setting a corresponding trigger after having called an Action or a Function Module. You can call them e.g. after the call of your own Actions and Function Modules within you own coding and use it to set triggers defined in the above system table.

- **SET\_TRIGGER\_FOR\_ACTION:** Can be called after execution of Actions. It checks for failed keys returned by the Action due to locking issues and sets the trigger for the found keys if required. Example Call:

```
CALL METHOD /scmtms/cl_trig_helper=>set_trigger_for_action(
  EXPORTING
    io_message      = lo_message
    it_failed_key    = lt_failed_key
    is_action_context = ls_action_ctx
    iv_trigger_id    = lv_trigger_id
```

```

        it_forced_key      = lt_key_async
IMPORTING
        eo_message         = lo_message2
        et_failed_key      = lt_failed_key2 ).

```

- **SET\_TRIGGER\_FOR\_FUNCTION:** Can be called after execution of Function Modules. It checks for locking issues and sets the trigger for all those keys that required it. Example Call:

```

CALL METHOD /scmtms/cl_trig_helper=>set_trigger_for_function
EXPORTING
    it_key          = it_key
    it_failed_key   = lt_failed_key
    is_parameter    = is_parameter
    iv_trigger_id   = /scmtms/if_trig_c=>gs_c_trigger_id-
                    tor_apply_chaco_strategy
    iv_function_module = /scmtms/if_tor_strat_const=>sc_fm_pro
                    c_strategy-chaco
    io_message      = lo_message2.

```

The set triggers are registered in the Trigger Header Table */SCMTMS/D\_TRIGHD*. An entry of this table provides information about the Trigger Context, i.e. which BO, BO node, instance of this BO node (i.e. which instance key), etc. has led to setting which trigger.

When running report */SCMTMS/PROCESS\_TRIGGER\_BGD* it tries to execute the function related with the trigger again. The corresponding information of the set triggers comes from the above mentioned Trigger Header Table */SCMTMS/D\_TRIGHD*. The report will delete entries from the table when they could be finally executed successfully. The table so to say represents the workload for the report. When you schedule the report to run periodically it will continue to restart functions until they are finally executed successfully. The report will remove entries from the Trigger Header table on successful execution.

## 4.5 Implicit Enhancements

For ABAP programs, a number of so called implicit enhancement options exist, for example:

- At the end of an include,
- At the end of a structure definition (`types`, `data`, `constants`, `statics`),
- At the start and at the end of a method or function module,
- Replacing method implementations by `overwrite-methods`.

These options provide very powerful means to alter standard code. In some cases, there are no other ways to enhance, for example when adding a type or data definition. In other cases, SAP strongly recommends to use BADIs instead, since they provide a defined interface. Nevertheless, some of the mentioned options shall be described here to be used as a means to create enhancements.

### 4.5.1 Use Implicit Enhancements with care

Implicit Enhancements should be used with care. The following aspects should be kept in mind when making use of this enhancement technique:

- Detailed knowledge on the application code is required for identifying the objects to be enhanced for a specific purpose.
- In case of methods that are not part of a stable interface, the signature can potentially change.
- This can lead to problems in case a pre- or post-method implementation relies on parameters from the methods signature, especially when parameters might have been removed.
- Enhancement SPAU might become necessary after updates to analyze conflict situations related to your Enhancement Implementations.
- In case of overwriting methods by copying the code of a standard method and adjusting it within an overwrite method implementation, you will not get the changes / corrections for the standard portion of your implementation.

### 4.5.2 Pre-, Post- and Overwrite Methods for existing methods

ABAP objects classes or interfaces can be enhanced by Pre-, Post- and Overwrite methods that are executed before or after the original method implementation or in case of Overwrite Methods replace the complete original implementation at runtime. As an example let's assume that we have identified class `/SCMTMS/CL_TOR_A_CONFIRM`, method `Execute` as the right method where a customer-specific behavior needs to be added. The following steps are valid for any class and for Pre-, Post- as well as Overwrite Methods:

- 1) Navigate to class `/SCMTMS/CL_TOR_A_CONFIRM` that shall be enhanced by the implementation of a Pre-Method (Post-Methods are created the same following way) and display the class.
  - Transaction `SE24` can be used, in case the class is already known.
  - Transaction `SE80` can be used to navigate to the class.
  - Transactions `/BOBF/CONF_UI` and/or `/BOBF/CUST_UI` can be used to navigate to implementing classes of BO node elements like Actions, Determinations or Validations of a BO to be enhanced.
- 2) Now follow menu path `Class → Enhance (Shift+F4)` to switch the class into enhancement mode. On the following two popups enter an Enhancement Implementation name and a short text which describes the enhancement. Continue with pressing `Enter`.

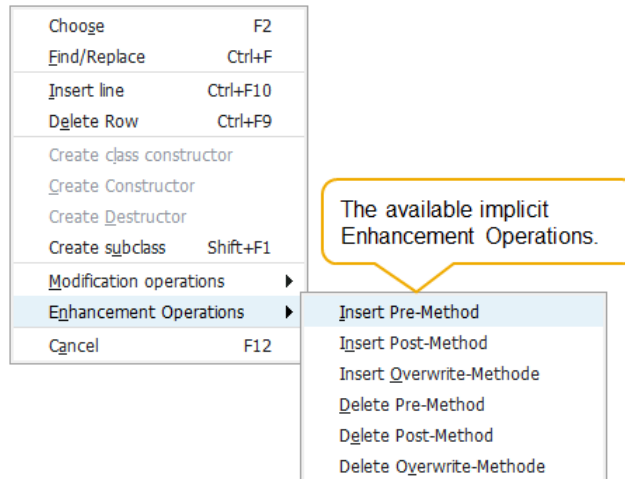
On the second popup specify the package where you want to store the Enhancement Implementation. In this example we store it as a local object in package `$TMP`.



- 4) As an example follow menu patch *Edit* → *Enhancement Operations* → *Insert Pre-Method* to add a Pre-Method to the marked method *Execute*.

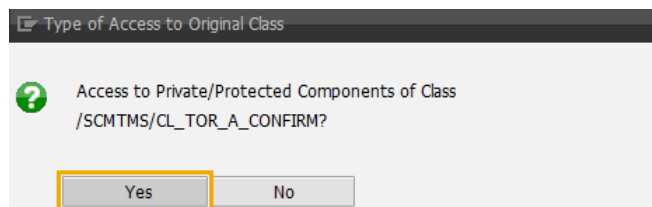
The menu (see picture below) also shows the other Enhancement Operations which can be executed, i.e. besides the Pre-Method you could also add a Post- and an Overwrite-Method via the same menu with the same described steps.

Moreover the last three operations in the menu allow also deleting existing Pre-, Post- and Overwrite-Methods, i.e. you can choose these operations to roll back corresponding implicit enhancements.



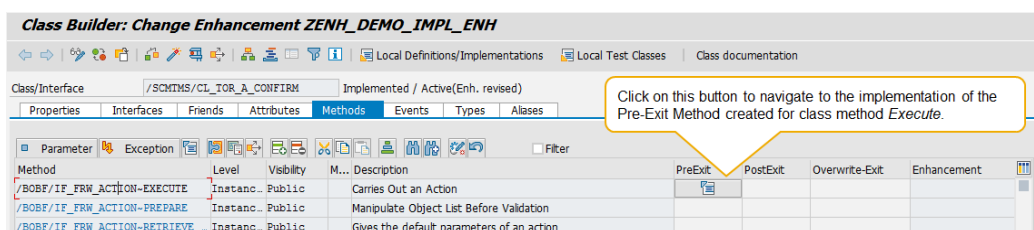
Picture: Available implicit Enhancement Operations.

After having chosen the required Enhancement Operation another popup will come up and ask you to specify whether you want to enable the enhancement implementation to have access to private and protected components of the original class. In this example we choose option Yes.



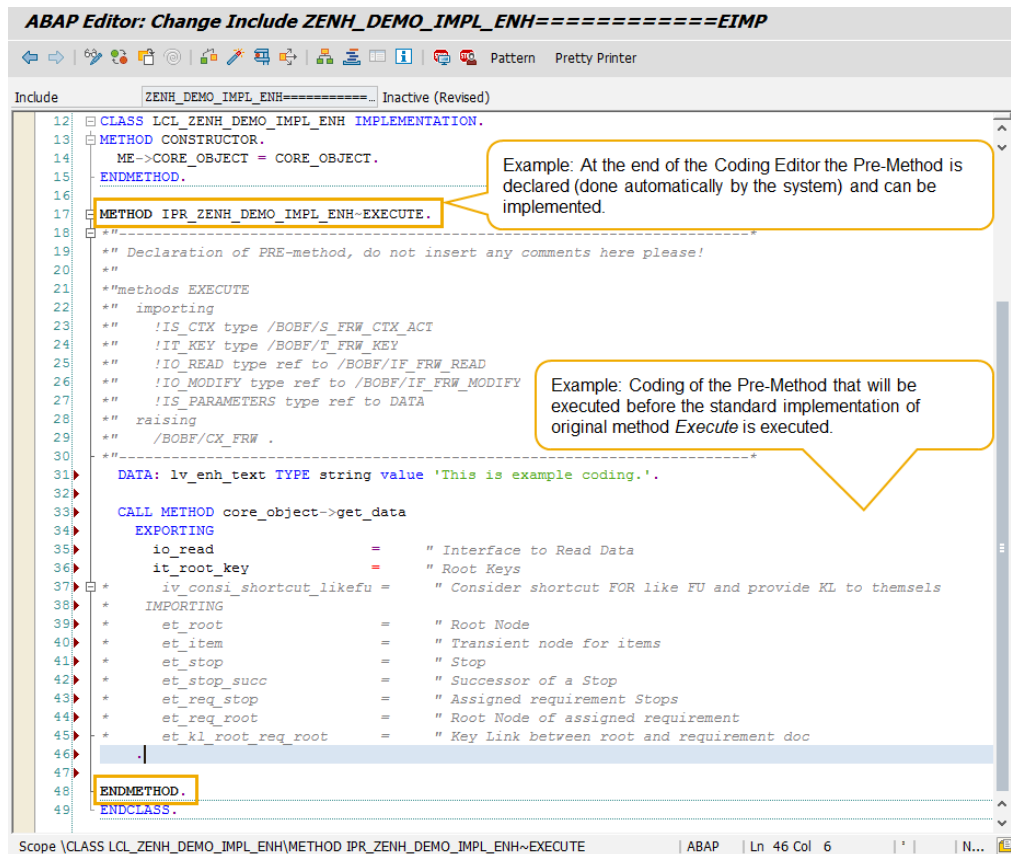
Picture: Specify the type of access to the original class.

- 5) For the selected method there will be a button displayed in one of the columns *PreExit*, *PostExit* or *OverwriteExit* of the method list, depending on the Enhancement Operation you have chosen. In the example we have chosen Insert Pre-Method. So the column *PreExit* for the enhanced method *Execute* now shows a button that allows navigating to the editor for the implementation of the Pre-Method coding. The same button would appear in the corresponding column in case of a Post- or Overwrite-Method.



Picture: Navigating to the implementation of an implicit enhancement.

- 6) Click on the button that is now visible in column *PreExit* for method *Execute* to Implement the Pre-Method with the coding that will be executed before the actual standard implementation at runtime. Implementation of Post- and Overwrite-Methods is triggered the same way where the Post-Method implementation is executed after the standard implementation and the coding of an Overwrite-Method would replace the complete standard implementation at runtime.



Picture: Implementing the example Pre-Method.

- 7) Finally save and activate the Enhancement Implementation. To stress again: You should make sure to handle the depicted implicit enhancements (Pre-, Post- and Overwrite-Methods) with care. Keep in mind the potential problems and consequences mentioned at the beginning of this section.

Moreover it is highly recommended to implement such enhancement code in your own local class methods and just place the call of these local class methods into the Enhancement Implementation. This will provide more transparency for customers and partners as well as for SAP in case of problem analysis, support, etc. This also provides a better overview for customers and partners over the coding that they have added with the described techniques.



## 4.6 Helper Classes provided by SAP TM

In the last sections techniques for enhancing the business logic were introduced. Some of them require customer or partner specific coding in available BAdIs, Implicit Enhancements, etc. During this work many tasks to be solved with coding occur repeatedly. Also the SAP TM standard implementation faces the situation that certain functions are needed repeatedly many times and in a variety of use cases. This kind of reuse functions are implemented in Helper Classes.

### 4.6.1 How to find SAP TM Helper Classes

Finding the helper classes provided by SAP TM is quite easy. Just start transaction *SE24* and enter */SCMTMS/\*HELPER\** in the field Object Type. Then press *F4* and check out the listed classes in the search result. You can find here more than 160 helper classes that serve a certain purpose (see short description for hints on what the class was implemented for). In general, the Helper Classes contain the term *HELPER* which makes it quite easy to find them.

Object Type Name	Short Description
/SCMTMS/CL_TOR_HELPER_OVERVIEW	BO TO Helper Class for Building Overview
/SCMTMS/CL_TOR_HELPER_QUANTITY	Quantity calculation methods
/SCMTMS/CL_TOR_HELPER_ROOT	BO TO Helper Class for Root-Related Tasks
/SCMTMS/CL_TOR_HELPER_ROUTING	Helper class for routing methods
/SCMTMS/CL_TOR_HELPER_SERVICE	Helper Class for TOR Service Handling
/SCMTMS/CL_TOR_HELPER_STAGE	Helper Methods for Stages
/SCMTMS/CL_TOR_HELPER_STATUS	BO TO Helper Class for Status-Related Tasks
/SCMTMS/CL_TOR_HELPER_STOP	BO TO Helper Class for Stop-Related Tasks
/SCMTMS/CL_TOR_HELPER_TEXT_COL	TOR Helper Class for Text Collection
/SCMTMS/CL_TOR_HELPER_TSP	Hel...
/SCMTMS/CL_TOR_HELPER_UI	Obs...
/SCMTMS/CL_TOR_HELPER_UNIT_TEST	Helper Class to define Unit test data
/SCMTMS/CL_TOR_HELPER_VALIDATE	BO TO Helper Class for Validations
/SCMTMS/CL_TOR_HELPER_WBN	Helper Class for WBN consumption
/SCMTMS/CL_TOR_OH_CUST_HELPER	Overview Hierarchy: Helper for maintenance view
/SCMTMS/CL_TOR_OUTB_HELPER	Helper Class for B2B Outbound
/SCMTMS/CL_TRANSPORT_HELPER	transport helper
/SCMTMS/CL_TRIG_HELPER	Helper Class for Setting Triggers
/SCMTMS/CL_TRQ_HELPER	Helper Class for Forwarding Order
/SCMTMS/CL_TRQ_HELPER_BLOCK	Helper Class for Block Handling
/SCMTMS/CL_TRQ_HELPER_CUST	TMS_TRQ: Helper Class for TR Customizing Access
/SCMTMS/CL_TRQ_HELPER_DEBUG	/SCMTMS/CL_TRQ_HELPER_DEBUG
/SCMTMS/CL_TRQ_HELPER_FWO_FWQ	/SCMTMS/CL_TRQ_HELPER_FWO_FWQ
/SCMTMS/CL_TRQ_HELPER_ITEM	TMS_TRQ: Helper Class for Item Handling
/SCMTMS/CL_TRQ_HELPER_SERVICE	TMS_TRQ: Helper Class for TRQ Service Handling

Picture: F4-Help with */SCMTMS/\*HELPER\** in *SE24*.

As you can see in the picture above there are multiple Helper Classes found that by naming convention relate to a certain Business Object. Class */SCMTMS/CL\_TOR\_HELPER\_STAGE* is an example for a class that contains functionality to extract Stage information of a set of given Freight Orders. Other reuse functionality in the context of the TOR Business Object is implemented in further Helper Classes */SCMTMS/CL\_TOR\_HELPER\_XXX*.

### 4.6.2 Why using SAP TM Helper Classes?

As indicated, there are certain functions that are required in many different use cases. It therefore made a lot of sense to implement such reuse functions in helper classes that can be reused.

Example: Instead of implementing the logic for extracting the stages of a Freight Order again and again with the risk of creating many different ways to do one and the same thing, this function is available in one of the Helper Classes that come with SAP TM. In this example instead of an own implementation, the developer could use method *GET\_SATGES* of the helper class */SCMTMS/CL\_TOR\_HELPER\_STAGE*. With its importing parameters this method allows e.g. specifying a list of Freight Orders (e.g. with their Root Keys) and a few



other parameters that determine what Stage Details shall be returned. In the exporting parameters the corresponding data is then returned.

It is recommended to check for available reuse functionality provided in the Helper Classes before making the decision for an own specific implementation. The following aspects should be considered:

- **Reduction of development time by reusing existing functions and methods:**  
SAP TM Standard development has already implemented numerous functions and methods that serve a specific purpose and are reused throughout the SAP TM application in all functional areas. The Helper Classes can help to reduce your development time by reusing already existing functions and methods.
- **Prevention of multiple approaches for one and the same function:**  
You have one single place in coding that realizes the required functionality which helps to keep the application consistent in terms of how a specific function or logic is realized and executed. The Helper Classes represent a single point of access.
- **Preventing inconsistent data retrieval:**  
Some SAP TM information like e.g. the Stop and Stages information of Freight Orders is stored in a way that multiple Business Object nodes are involved to represent the data in a very flexible way. Stop and Stages information of a Freight Order e.g. is represented by the STOP and STOP\_SUCCESSOR node of the related TOR Business Object.

Accessing this data should not be implemented by yourself if you do not exactly know the underlying data model. Using the methods of the Helper Classes ensures that you access this data exactly with the same consistent and performing logic like the SAP TM standard application.

Nevertheless when reusing helper classes and their methods you should always test and verify the provided functionalities. Make sure that in your use case the reused classes and methods really return the required data and prevent reading unnecessary data or executing unnecessary calls, i.e. follow the principle “as much as necessary and as few as possible”. As standard development is continuously optimizing the performance of the helper classes and their methods you should always make sure that they also perform well in the context in which you used them, i.e. they may not perform well if they are used in a non-ideal context.

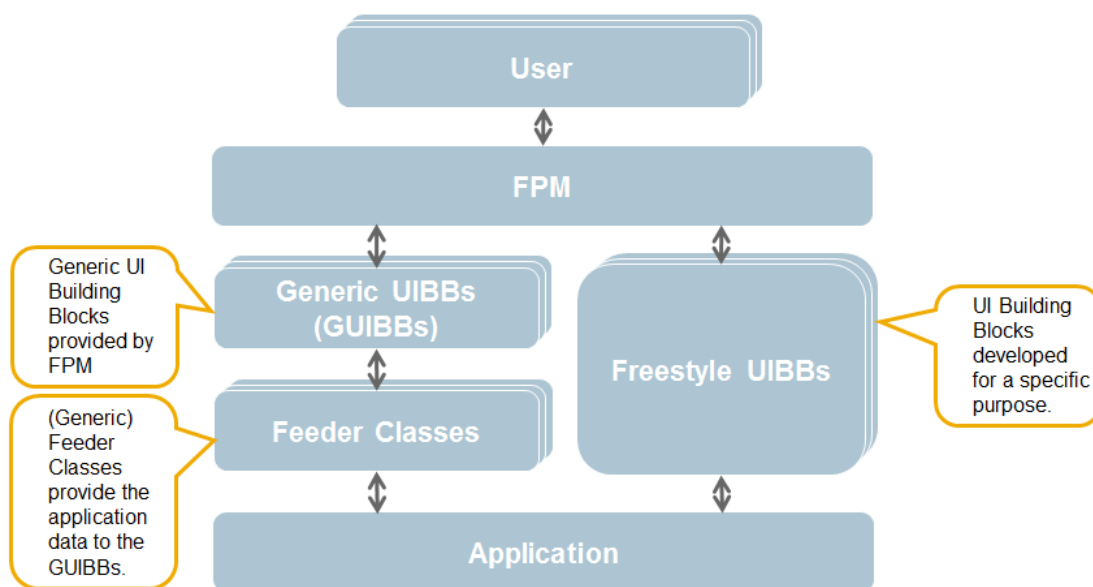
## 5 User Interface Enhancements

This chapter provides the basics on how to enhance the user interface of TM 9.0. As mentioned in sections 2.2 and 2.3 the user interface is built with the help of the Floor Plan Manager (FPM) and the Floor Plan Manager BOBF Integration (FBI). These two frameworks enable enhancing a user interface via configuration rather than having to implement additional code.

This document can for sure not cover a complete description of FPM and FBI. It therefore concentrates on the very basic things that customers and partners need for creating basic and common enhancements of the UI by adjusting the standard configurations of the TM user interface. The examples used here are based on the Freight Order UI but the principles and techniques are valid for any other TM user interface too. For more complex user interface enhancements, it is recommended to build up more detailed FPM and FBI knowledge.

### 5.1 FPM – Floor Plan Manager

Since release 8.0, SAP Transportation Management uses the Floor Plan Manager (FPM) to realize its User Interfaces. FPM is a Web Dynpro ABAP application that provides a framework for developing new Web Dynpro ABAP application interfaces consistent with the SAP UI guidelines. FPM allows a modification-free composition of discrete User Interface Building Blocks (UIBBs) which are compliant with the mentioned guidelines.



Picture: FPM Overview.

#### 5.1.1 User Interface Building Blocks

The Web Dynpro ABAP Floorplan Manager (FPM) is a framework which composes application specific views (UIBBs) to an application. This allows a homogeneous high-level application structuring and interaction behavior. Instead of building the User Interface as an individual Web Dynpro Application, FPM centrally provides predefined UIBBs, so called Generic UI Building Blocks (GUIBBs) that can be reused to create UIBBs. GUIBBs used in the TM 8.0 User Interface are:

- **Overview Pages** (*FPM\_OVP\_COMPONENT*): Defines the general layout of the screens. It displays a title bar, a tool bar as well as one or more UIBBs.
- **Form GUIBB** (*FPM\_FORM\_UIBB*): A flat collection of input elements which displays the content of a (flat) structure → must use a form-compliant Feeder Class.

- **List GUIBB** (*FPM\_LIST\_UIBB*): Displays the content of an (internal) table → must use a list-compliant Feeder Class.
- **Tree GUIBB** (*FPM\_TREE\_UIBB*): Displays the content of an (internal) table in a hierarchically way → must use a tree-compliant Feeder Class.
- **Tabbed GUIBB** (*FPM\_TABBED\_UIBB*): Used to display a tab strip including further UIBBs with an optional master UIBB on top of it → does not require a Feeder Class (sometimes misused for layout purposes which it was not designed for).

The application only provides the data and a layout configuration to these GUIBBs. The rendering is handled by the framework itself. Generic UI Building Blocks provide a comprehensive way of creating or changing User Interface Compositions, without the necessity to change the underlying application code base and thereby offering a concept for modification free customer UI enhancements. The composition (configuration) of those building blocks takes place in a design time application (in this case a Web Application) where all the necessary field attribute, positioning and layout properties are assigned or composed.

GUIBBs 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.

### 5.1.2 Feeder Classes

Necessary or mandatory application specific information will be supplied by the application itself via a so called Feeder Class implementation. Feeder Classes are based on a predefined interface definition providing all necessary methods and corresponding signatures for standardizing the communication between the application and the GUIBB. With these Feeder Classes the application

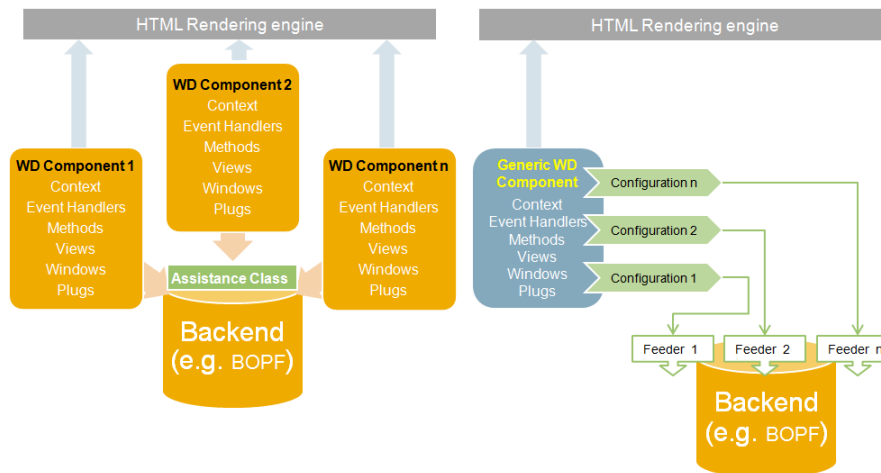
- Provides a field catalogue to the GUIBB design and runtime.
- Provides the data at runtime.
- Accepts UI changes at runtime by calling application middle ware.
- Handles user interactions (events) at runtime by calling application middle ware.
- Provides field control data to control visibility and changeability of UI elements.

The UI Administrator or Designer can

- Create UI layouts as a Web Dynpro Configuration for the standard GUIBBs.
- Put together such discrete GUIBB configurations in an application configuration.

In the traditional approach, the UI developer develops multiple Web Dynpro Components with fixed view layouts and delivers a fully assembled application. With this approach, realizing customer-installation specific UI variants requires modification of such applications.

With the FPM approach, it is possible to enhance application user interfaces and fit them to your business needs, based on configuration instead of modifications. Besides the GUIBBs, FPM still allows the implementation and usage of freestyle UIBBs that can be realized individually to serve specific purposes that cannot be handled via GUIBBs.



Picture: Traditional and FPM UI development approach.

At runtime, user interactions are handled by FPM events that pass an FPM phase model (Event Loop). Within the FPM event loop specific methods are called that are based on a predefined interface definition and corresponding signatures in order to standardize the communication between the application and the GUIBB. A Feeder Class implements such a predefined interface for a specific GUIBB, e.g. the interface *IF\_FPM\_GUIBB\_FORM* for Form components. Important methods are:

- **INITIALIZE:** Called at runtime when the form is created. It is the first feeder method which is called from FPM.
- **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).
- **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 data from the form to other components in the same application.
- **PROCESS\_EVENT:** Called within the FPM event loop. The FPM *PROCESS\_EVENT* is forwarded to the feeder class. Here the event processing can take place and this is where the event can be canceled or deferred.
- **GET\_DATA:** Called within the FPM event loop. The FPM *PROCESS\_BEFORE\_OUTPUT* event is forwarded to the feeder class. Here you specify the form data after the event has been processed.

There are two options when building an FPM-based application. First option: Individual Feeder Classes. Each GUIBB has its own individually implemented feeder class. Second option: Usage of Generic Feeder Classes that are provided with the contextual information via feeder parameters. In SAP Transportation Management, the second option was chosen. The advantage is that the feeders need to be implemented only once (high reuse) and enhancements in the feeder logic are implemented in less feeder classes.

### 5.1.3 Wire Model

The wire model is used to create a running FPM application by pure configuration (or at least with a 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 like BOPF.

A wire controls the runtime interdependencies between two UIBBs, i.e. they determine the data content of the target UIBB depending on user interaction changing the “output” of the source UIBB. Outputs can be of type lead selection, selection or collection. For example, changing the lead selection in a list of Forwarding Order Items may change the data content of another list displaying the associated Item Details.

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 output 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.

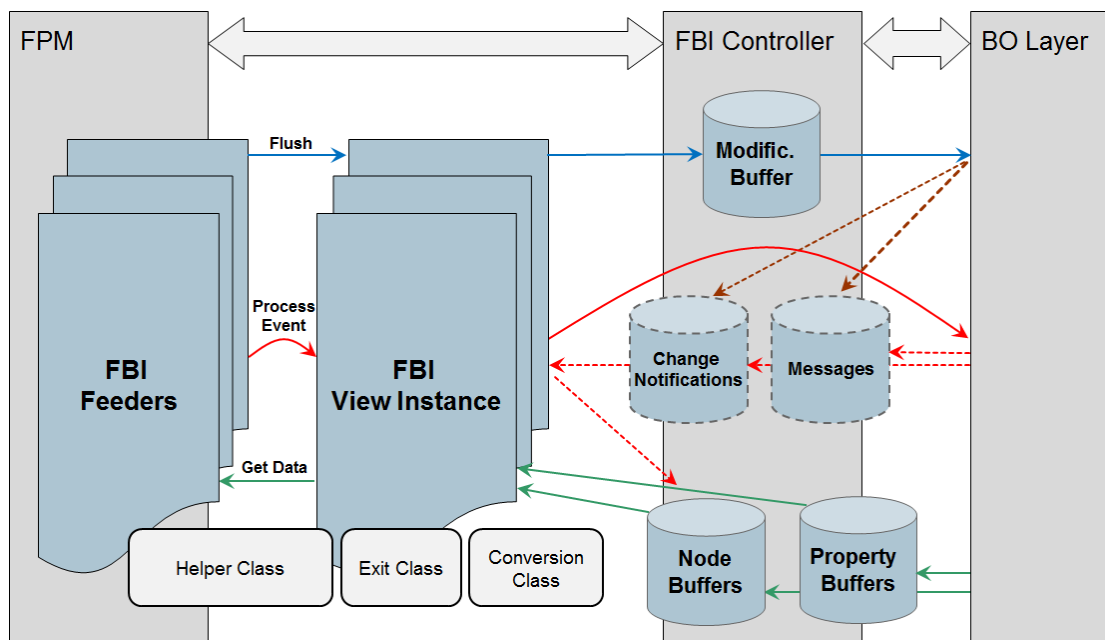
An example for using a wire is provided in section 5.4.3 where a new tab is added, based on an extension sub node.

## 5.2 FBI – Floor Plan Manager BOPF Integration

The Floor Plan Manager BOPF Integration (FBI) is used in SAP Transportation Management 8.0 to integrate FPM with the BOPF-based Business Objects. FBI provides generic FPM application feeder classes together with the relevant application configuration that allows consuming services of Business Objects modeled in BOPF. These BOPF services can be used seamlessly in a modification-free UI environment.

FBI provides the following functionalities that support the communication and corporation between FPM applications and BOPF-based Business Objects:

- Editing data of BO node instances in the standard GUIBBs FORM and LIST.
- Accepting action parameter values and invoking corresponding actions on BO node instances.
- Overview Search (OVS) based on BO node queries.
- Input of external IDs on initial screens and subsequent conversion of these external IDs into internal (technical) IDs (Alternative Key Conversion).
- UI-specific services are supported:
  - Navigation to multiple targets.
  - Calling dialog boxes and editing application data in these dialog boxes.
  - Support of UI-specific non-BOPF actions.



Picture: Technical relation between FPM, FBI and the BO layer.

Some concepts/entities of FBI that will be relevant for the UI enhancement topic described later in this document:

### 5.2.1 FBI View (design time)

FBI Views are the place where the design time UI structure of a building block is defined. Moreover, it contains the classes for conversion/mapping of BOBF BO data to this UI structure. An FBI View is closely related to a single BO node. But "Related Views" are also supported. They allow extracting data from multiple BO nodes into a single UI structure. Actions that are not related to the BO are also defined in the FBI View (FBI views are stored as a configuration of Web Dynpro Component `/BOFU/FBI_VIEW`).

Component Configuration /SCMTMS/FRE\_ORDER\_ITM

Save Cancel Edit Check New Window Enhance Properties

Component-Defined

Header Related Views Field Groups Field Descriptions Actions Field Mapping Usage in Component Config

Business Object: /SCMTMS/TOR  
 Node: ITEM\_TR  
 Node UI Structure: /SCMTMS/S\_UI\_TOR\_ITM\_TR  
 Mapper Class: /SCMTMS/CL\_UI\_CONVERSION\_TOR  
 Exit Interface Class: /SCMTMS/CL\_UI\_VIEWEXIT\_TOR

Read-only: ☐  
 No Retrieve: ☐  
 No Property: ☐  
 Expose Domain Values: ☐  
 DDIC CheckValue: ☐

Tech Fields Struct:   
 Output Struct:

Created By: FLORESCU  
 Created On: 07.08.2010 13:18:40  
 Changed By: MORMANN  
 Changed On: 07.03.2011 15:59:38

Example FBI View for the Freight Order Items screen.

Further details of the FBI View like information on related views, actions, etc.

The main header information of an FBI View with related BO, BO node and UI structure; Mapper (Conversion) Class and Exit Class can be specified here.

Picture: Example FBI View.

- **Header:**
  - Contains the mandatory part: Business Object and Node.
  - Optional UI Structure (if not specified, then node structure is used).
  - Optional Mapper Class (if not specified, MOVE-CORRESPONDING is used).
  - Optional Exit Interface Implementation Class (details later).
  - Additional settings, like Read-Only, etc.
- **Related Views (optional):**
  - Allows the definition of a chain of Views to read data from more nodes (e.g. when information coming from several nodes shall be combined into one flat UI structure to be displayed in a list).
  - Each related view is included with a mandatory suffix. This helps preventing collisions in case in two or more involved Views attributes with the same name appear.
- **Field Descriptions (optional):**
  - Can be used to specify additional properties for structure attributes.
  - These settings are passed to the FPM field catalogue.
  - E.g. Sorting Allowed, Allow Filter, Domain Fixed Values, Fixed Values, F4-Values from Code Value List etc.
- **Actions (optional):**
  - Allows definition of new Event IDs.
  - For the new Event IDs, as well as for the existing ones (Standard FBI and BO Actions, which are taken into account automatically), you can specify additional settings:
    - Set specific Name and Tooltip (via OTR aliases) – they will be passed to the FPM action catalogue.
    - Specify another Event ID, whose enable/disable properties are to be inherited.
    - Specify whether the action is allowed to be triggered in read only mode.
    - Specify whether the action is allowed to be executed only when a record is selected.



- Specify navigation target (in this case, FBI calls Navigation Class instead of Standard handling).

### 5.2.2 FBI View Instance (runtime)

The instances of an FBI View hold the keys of the displayed instances (coming from the wires that UIBBs are connected with). An instance e.g. prepares modifications, executes actions and posts change notification to the controller. It reacts to the FBI-specific SYNCUP Event, i.e. it evaluates change notifications and determines which of the keys must be refreshed.

Moreover it reads the data from node buffers or from the BO layer in case of modified keys. Where required it also calls conversion classes for the modified records (e.g. to convert a document ID into its corresponding technical key). A view instance calls available Exit methods at the appropriate places.

### 5.2.3 FBI Controller (runtime)

The FBI controller is responsible to do the orchestration between FBI View instances and the BO layer. It does not contain any application logic but only provides the technical framework for the orchestration. It provides a Modification Buffer for changes done on the UI that are then forwarded from there to the BO layer, i.e. it centralizes the BO Layer responses. It also collects the change notifications coming from the BO layer that then need to trigger updates on the UI.

Further buffers hold the information on the nodes read from the BO layer and the properties of nodes and their attributes. The properties determine e.g. whether an attribute is a mandatory field or is ready for input. Moreover, these buffers help to avoid redundant BOPF service calls.

### 5.2.4 Conversion Classes

When data is send from the BO layer to the UI, the conversion class is called to convert technical attributes into their clear text representation. The same conversion class is also called when data is send from the UI back to the BO layer, i.e. it converts clear text information in to its technical representation.

Conversions are done immediately after retrieval of data and shortly before sending modifications to the buffer. A conversion class is specified in the FBI View definition. Implementations of Conversion Classes do always inherit from TM super class `/SCMTMS/CL_UI_CONVERSION`. Each redefinition of the super class must define its own mapping table in method `BUILD_MAP_TABLE`. The super class already contains a few generic (bidirectional) mapping rules which are based on field naming conversions:

- **Mapping rule for Date-Time Conversion:** The conversion rule maps a field of type `TIMESTAMP` into its Date, Time and Time Zone part. A BO node attribute `FIELD` of type `TIMESTAMP` is automatically converted with this rule if the UI structure contains the attributes `FIELD_D` (Date), `FIELD_T` (Time) and `FIELD_TZ` (Time Zone).
- **Mapping rule for Date-Time Conversion into String:** The conversion rule maps a field of type `TIMESTAMP` into a String. A BO node attribute `FIELD` of type `TIMESTAMP` is automatically converted with this rule if the UI structure contains the attribute `FIELD_TTT` (Formatted Date).
- **Mapping Rule for Alternative Key Conversion:** The conversion rule maps a BO node foreign instance key into its corresponding foreign readable ID. For this, it uses e.g. the BO key, the BO node name and the alternative key for this node defined in the node Meta Data.

- **Mapping Rule for Code List Conversion:** The conversion rule maps a BO code into a readable UI code value. A BO attribute *FIELD* with its code value X is converted into its readable UI code value if the UI structure contains the attribute *FIELD\_TXT*.

### 5.2.5 Exit Classes

The generic feeder classes usually take care of all communication aspects between the corresponding GUIBB and the application. Nevertheless, there might be use cases that require a more specific implementation. For this, an Exit Class can be specified in the FBI View definition. Exit Classes provide many extension options and are the recommended means for adapting the standard FBI processing to customers and partner's needs.

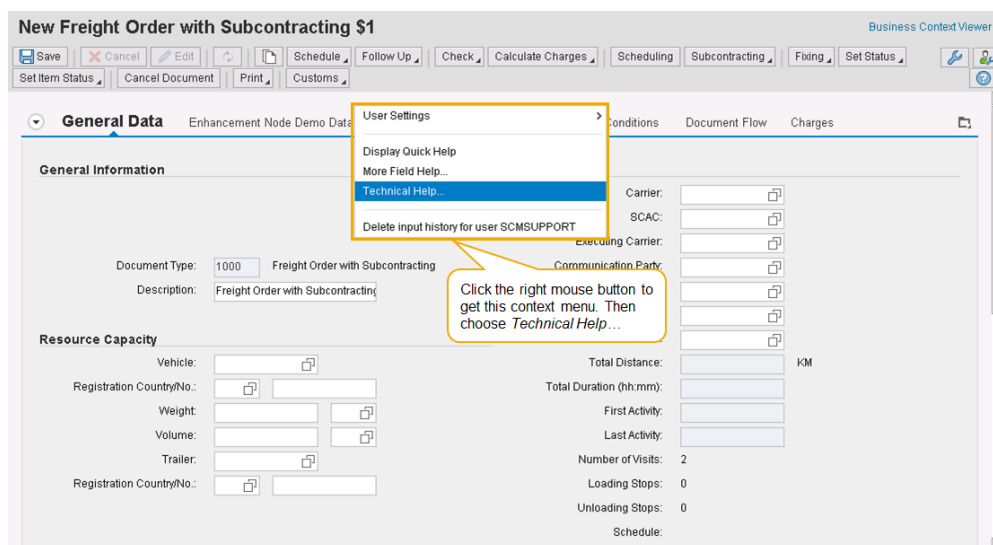
Exit Class implementations inherit from TM super class */SCMTMS/CL\_UI\_VIEWEXIT\_CMN*. An Exit Class implements the following FBI Exit Interfaces:

- **Core Interface */BOFU/IF\_FBI\_VIEW\_EXIT\_INTF*:** The interface does not have any interface methods but the Exit Class implements it for enabling FBI to instantiate an object from this class.
- **Definition Interface */BOFU/IF\_FBI\_VIEW\_EXITINTF\_DEF*:** The methods of this interface offer the possibility to influence the processing of FPM phases Initialization and Get Definition ("one time" phases). Its implementation is optional.
  - o Method *ADAPT\_FIELDS*: Modify the field catalogue.
  - o Method *ADAPT\_ACTIONS*: Modify the action catalogue.
  - o Method *ADAPT\_DND\_DEFINITON*: Modify drag & drop definitions.
- **Definition Interface */BOFU/IF\_FBI\_VIEW\_EXITINTF\_RUN*:** The methods of this interface offer the possibility to influence the processing of User Interactions (at each round trip). Its implementation is also optional.
  - o Method *ADAPT\_CHANGE\_LOG*: Modify the list of screen changes before converting them into BO modification records.
  - o Method *ADAPT\_EVENT*: Intercept and process any event that arrives in the underlying FBI view. If custom event IDs were added to the FBI View, this is the place to implement the action handling for them.
  - o Method *ADAPT\_MESSAGES*: Modify the returned messages from the Modify and *DO\_ACTION* service calls.
  - o Method *ADAPT\_DATA*: Modify the data before it is passed to FPM. The data is in the concatenated format (all related views in the chain plus the reference fields). Thus, the component of the UI structure must be accessed with *ASSIGN COMPONENT...*
  - o Method *ADAPT\_FIELD\_PROPERTIES*: Modify the field properties of this view (at column level, these values are merged with the properties from the reference fields of the data structure).
  - o Method *ADAPT\_ACTION\_PROPERTIES*: Modify the enabled/disabled properties of this view's actions.
  - o Method *ADAPT\_SELECTION*: Modify the selected lines (in list and tree).

### 5.3 General remarks on user interface enhancements

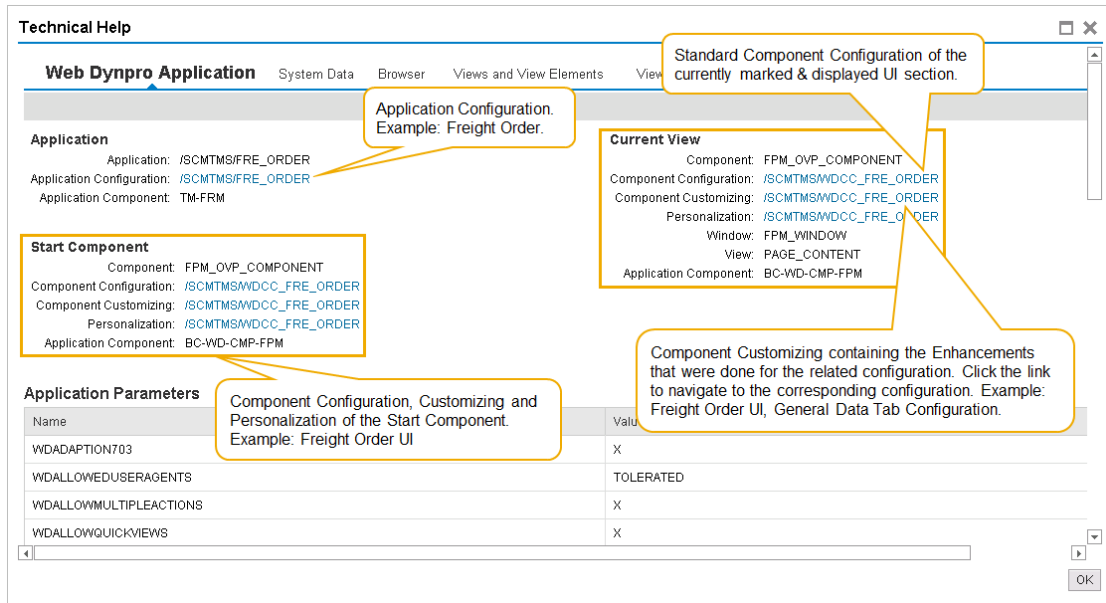
Some general remarks for creating user interface enhancements:

- Basic enhancements ideally can be done without any coding. For more complex user interfaces and enhancements, coding might be required, e.g. implementation of the Exit Class Methods mentioned in the last section. The examples in the following sections try to illustrate both.
- Each TM user interface is build up from so called User Interface Building Blocks (UIBBs) as already described in sections 5.1 and 5.2. Each of these building blocks has a configuration that can be adapted by partners and customers.
- The standard configurations will remain untouched. Configuration Enhancements can be created e.g. in a development system and get transported to a test or production system. The client of the system where you do the UI enhancements must be set up in a way that it allows development and transporting configurations.
- Enhancements can also be deleted again. After deleting e.g. an Enhancement of a configuration, the original standard configuration is in place again for processing the corresponding user interface.
- As per TM 9.0 the navigation to relevant configurations has been simplified. You just need to start the required User Interface and use the Technical Help link from where you then can navigate further to the different configurations that make up the UI.



Picture: Invoke *Technical Help...* to navigate to UI configurations.

Place the mouse pointer on the UI section of interest and click the right mouse button to display the context menu (see picture above). Then click on *Technical Help*. On the following screen, you get an overview of the current application configuration, the start configuration (Web Dynpro Component Configuration containing all sub-elements and their configurations) and the configuration of the current view, i.e. the part of the display that you marked before.



Picture: Technical Help for a selected UI section.

- On the Tab *Web Dynpro Application* of the technical help screen you can find the application configuration in section *Application*. Click on the link to navigate to the application configuration where you can display (and adjust) general application parameters.
- In section *Start Component* of the same screen you can find the leading Web Dynpro Component Configuration (WDCC). Three links are listed here.

#### Component Configuration:

It represents the starting configuration of the corresponding application and contains all sub-elements (UIBBs, Views, etc.) with their related configurations. When the start configuration is displayed you can navigate further to all sub components of this application.

#### Component Customizing:

In the component configuration you can only see the standard content. Only in the Component Customizing you can later on see the Enhancements that you did for a standard configuration. Moreover, only here you can add Enhancements to existing configurations.

Make sure that in transaction SICF the corresponding Service is activated. You can check this under the following path: sap → bc → webdynpro → sap → customize:component. In case the service is not active, let your system administrator activate it. Otherwise you cannot create any UI enhancements.

#### Personalization:

When following this link you get to the personalization settings for the specific application, i.e. here you can see all personalization settings of each authorized user as well as the general personalization settings for the application that are valid for all authorized users.

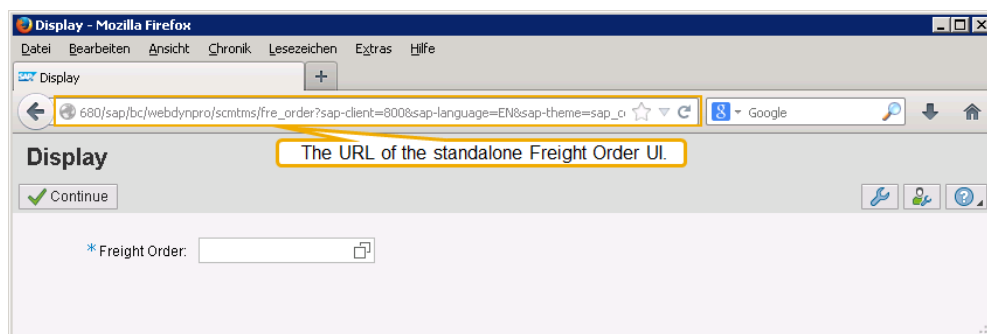
- In section *Current View* you can find a link to the Component Configuration and Component Customizing that you have marked with the mouse pointer to start the technical help, i.e. it allows navigating to the configuration and customizing of the UI component that currently has the focus.
- To create enhancements, you need to start the corresponding user interface from the SAP user menu or from within NWBC. Within the user interface you can then use the mentioned technical help (as per NW 7.31).

- A UIBB might in turn include other UIBBs. So when entering the configuration of a UIBB in the configuration editor you may have to navigate to further configurations in the Configuration Editor to get to the specific UIBB's configuration that shall get enhancements.
- There is another option to identify specific User Interface building blocks and their related configuration to be enhanced. When starting the User Interface from the SAP Menu (i.e. the “standalone” UI) you can add the following parameter to the URL of the User Interface: *sap-config-mode*.

In the browser, simply place *&sap-config-mode=X* at the end of the URL to switch the User Interface into the Customizing Mode. When setting it to *blank* (instead of *X*) you switch of the Customizing Mode again.

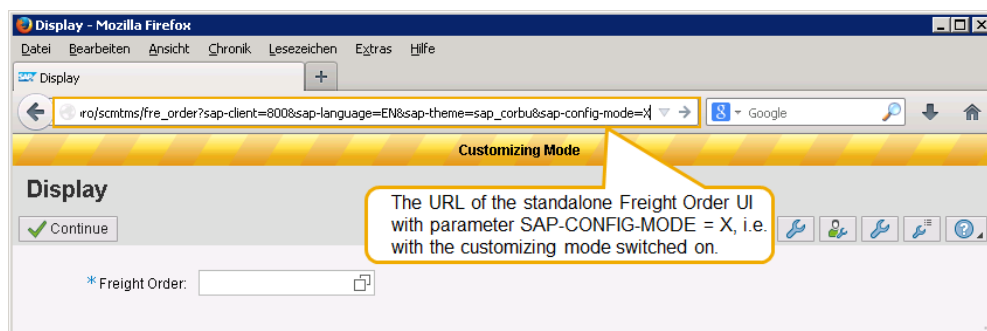
Note: This parameter can unfortunately still not be used for UIs started within NWBC.

- Example: Assume we want to enhance the user interface of business object Freight Order.
  - a) Start the user interface to be enhanced from your user menu to use it in your browser.



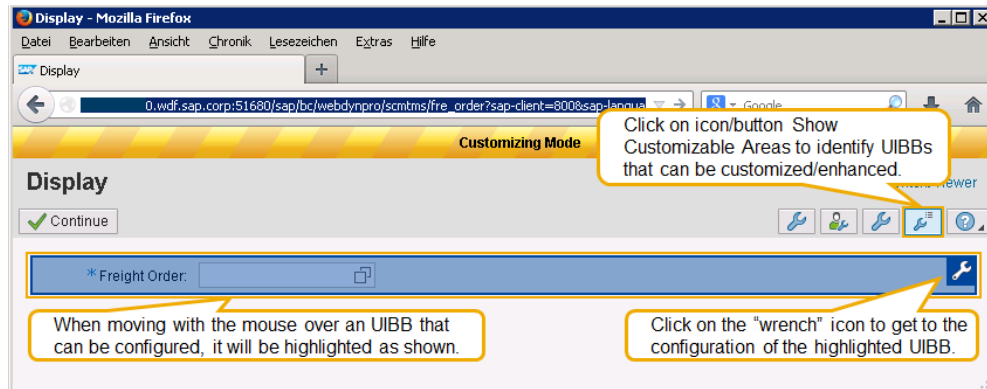
Picture: Example – Standalone Freight Order UI with its URL.

- b) At the end of the URL enter the following additional parameter: “&sap-config-mode=X” to enable enhancing/customizing this user interface. On the screens you will now see an orange bar with the text *Customizing Mode*, indicating that you are now exactly this mode.



Picture: Adjusting Parameter SAP-CONFIG-MODE in the URL.

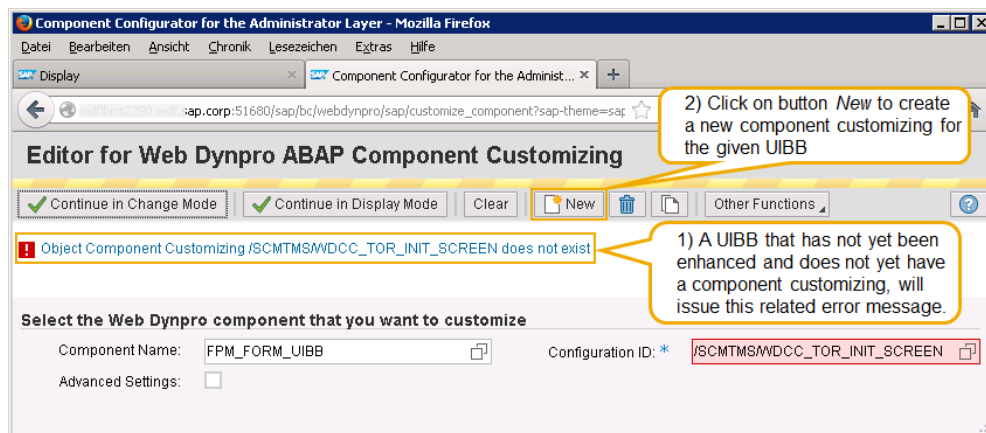
- c) Now click on the button (icon) *Show Customizable Areas*. This will now allow you to move the mouse over the different UI parts. Where ever a configurable component is detected, it will be marked with a little “wrench” icon that you can click on to start with enhancing this specific part (UIBB) of the UI. You will get directly to the Configuration Editor for the selected UIBB.



Picture: Identifying UIBBs that can be customized / enhanced.

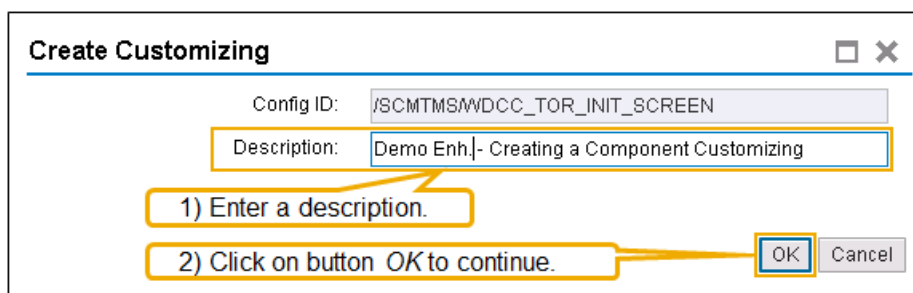
The simple example above shows the initial screen of the Display Freight Order UI. The described procedure of course works when you now continue and display a discrete Freight Order Document. On the next screen you can identify all UIBBs that make up the corresponding Freight Order UI or any other UI that you take a look at in Customizing Mode.

- A UIBB might in turn include one or more other UIBBs. When entering the configuration of a UIBB, within the configuration editor you may have to navigate to further configurations of such "sub" UIBBs to get to their specific configurations if they are supposed to get enhanced.
- When starting the configuration editor (in Customizing Mode) for a specific UIBB configuration that has not yet been enhanced, the system will always notify you with a corresponding error message: *Object Component Customizing xyz does not exist*.



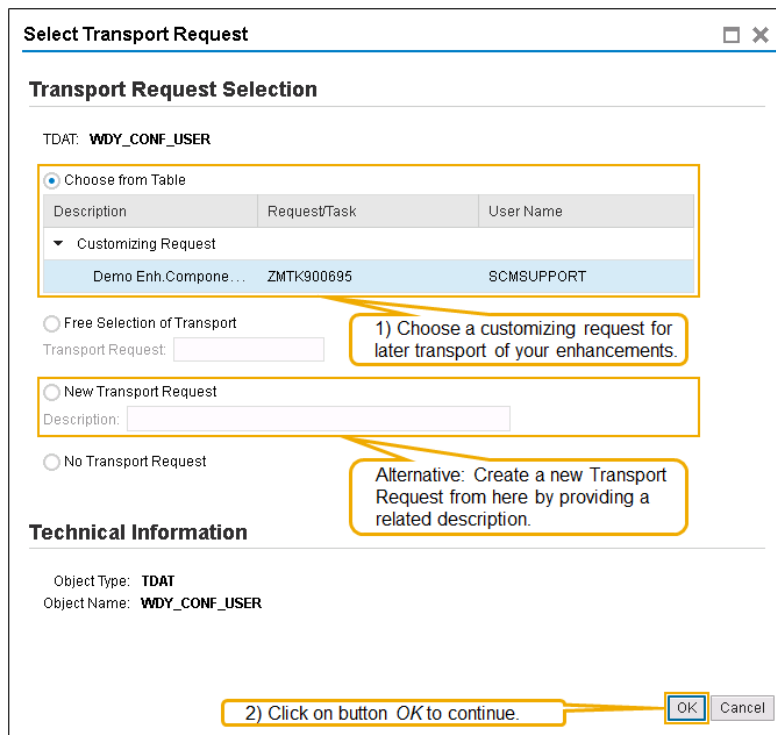
Picture: Creating the initial Component Customizing.

The required Component Customizing can now be created by clicking on button New which will first of all raise a popup where you can provide a description for the new Component Customizing.



Picture: Enter a description for the Component Customizing.

The System will now ask you to assign either an existing Transport Request or create a new one that will carry all the adjustments for later transport from your development system to a test or production system.



**Select Transport Request**

**Transport Request Selection**

TDAT: **WDY\_CONF\_USER**

☒ Choose from Table

Description	Request/Task	User Name
Customizing Request		
Demo Enh. Compone...	ZMTK900695	SCMSUPPORT

☐ Free Selection of Transport  
Transport Request:

☐ New Transport Request  
Description:

☐ No Transport Request

**Technical Information**

Object Type: **TDAT**  
Object Name: **WDY\_CONF\_USER**

Picture: Assigning or creating a Transport Request for Enhancements.

- So enhanced configurations or Component Customizing can be transported e.g. from the development system to the test system and further to the production environment after successful test.

Adjustments / enhancements of existing component configurations can be considered as customizing, i.e. in this case you enhance an existing standard object.

But: When you add completely new UIBBs (i.e. your very own configurations) to integrate them into an existing UI, the configuration, FBI View, etc. that you create are actually new objects that will be stored in your own package as your customer / partner specific objects. They can be attached to a workbench request

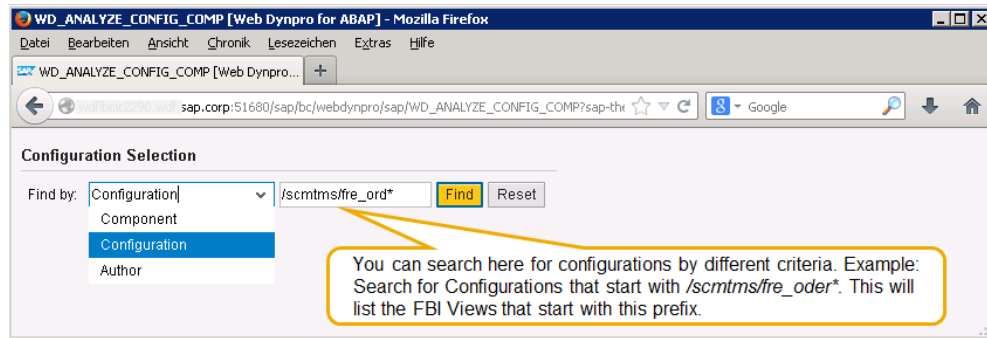
- Transporting as well as deleting created Component Customizing is also possible via a corresponding Web Dynpro application that can be started with the following general link which has to be enhanced with information on server and port, depending on where you want to start the tool.

[http://\[server\]:\[port\]/sap/bc/webdynpro/sap/wd\\_analyze\\_config\\_comp](http://[server]:[port]/sap/bc/webdynpro/sap/wd_analyze_config_comp)

Example:

[http://ukwtr9x.wdf.sap.corp:80089/sap/bc/webdynpro/sap/wd\\_analyze\\_config\\_comp](http://ukwtr9x.wdf.sap.corp:80089/sap/bc/webdynpro/sap/wd_analyze_config_comp)

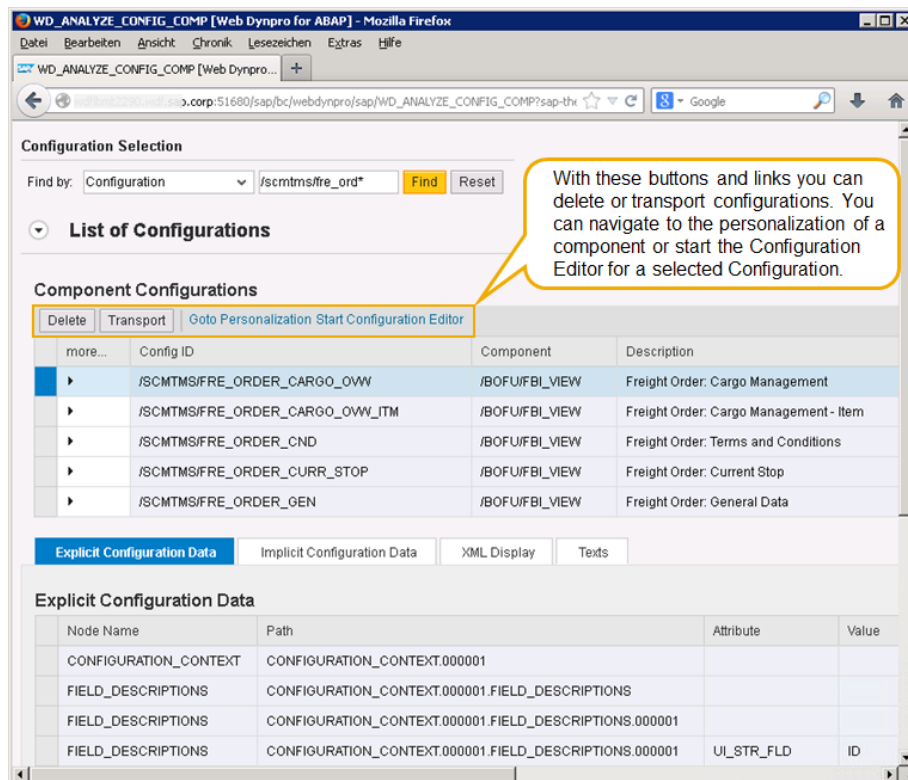




Picture: Initial screen of the `WD_ANALYZE_CONFIG_COMP` tool.

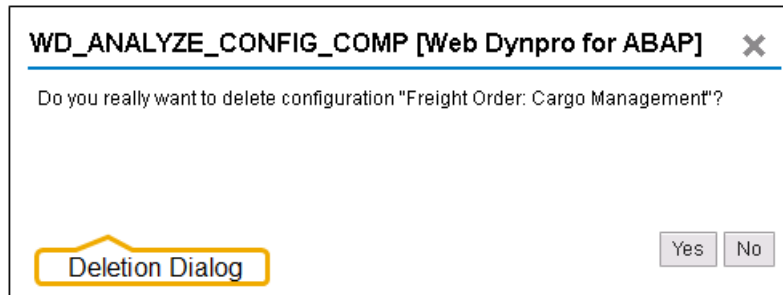
On the initial screen of the tool you can select configurations by different criteria (Component Name, Configuration Name or Author). Examples:

Enter e.g. `/SCMTMS/WDCC_FRE_ORD*` to search for UIBB configurations that are used for the Freight Order UI. Or enter `/SCMTMS/FRE_ORD_*` e.g. search for FBI Views used in the context of the Freight Order UI.

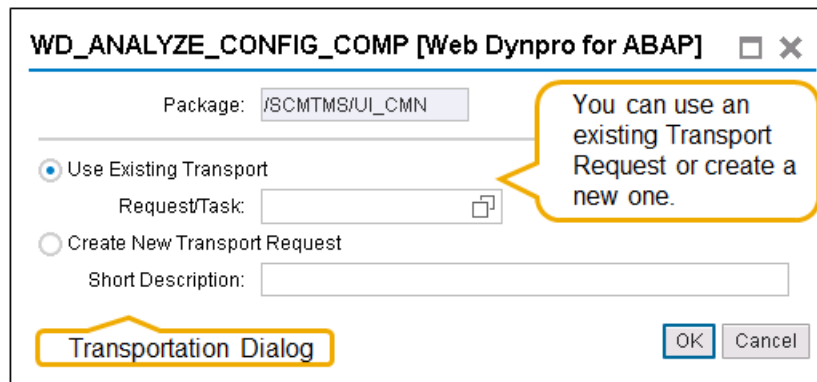


Picture: `WD_ANALYZE_CONFIG_COMP` tool with selected Configurations.

When clicking on button *Delete* the selected Configurations are deleted after confirmation. To transport selected Configurations (e.g. your own UIBB Configurations) click on button *Transport*. On the following popup you can enter either an existing transport request or create a new one.

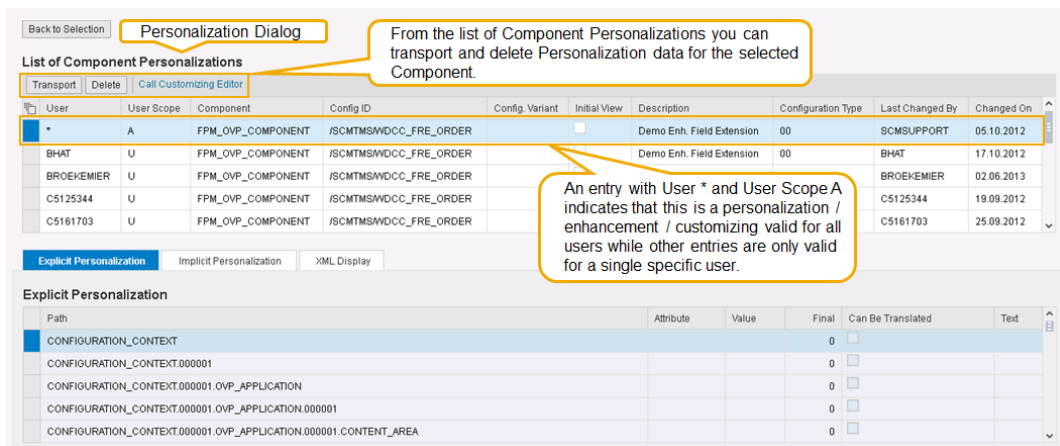


Picture: Confirm deletion.



Picture: Using an existing or creating a new Transport Request.

- When you click on the link *Goto Personalization* you can see all personalization records for the selected Component Configuration. In the list of Component Personalizations you can find entries / records that represent user specific settings, e.g. User = ABCDE and User Scope = U or entries / records that are valid for all users, i.e. User = \* and User Scope = A. Using the buttons *Transport* and *Delete* allows transporting and deleting selected entries. The link *Call Customizing Editor* allows starting the related editor.



Picture: List of Personalizations.

- When clicking the link *Start Configuration Editor* the Editor is started for the selected Component Configuration (i.e. in Customizing Mode) where you can adjust existing or create new enhancements.

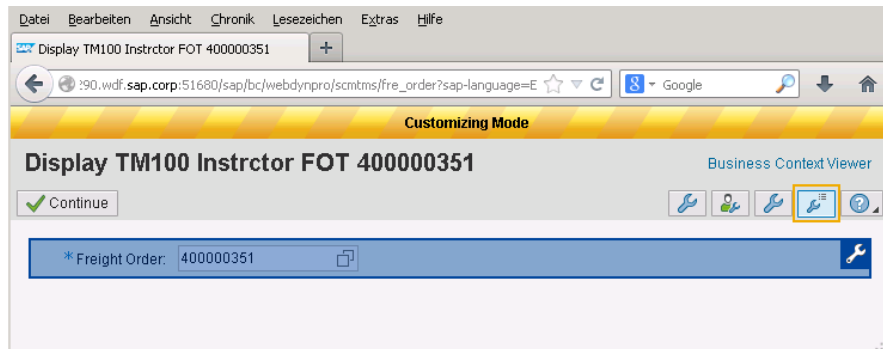
In the following sections different UIs will be used to demonstrate enhancement use cases. As mentioned, the principles and techniques used in these examples can be applied for any other TM UI too. The next section describes how to do a simple field extension on the UI. This section will also describe some more detailed aspects of the Configuration Editor which will be used in general, i.e. the descriptions provided in this next section also hold for other UI configuration enhancements and creating new customer / partner specific configurations.

## 5.4 Enhancing the User Interface

### 5.4.1 Field Extensions

A very basic and common example for extending the user interface is adding additional fields on an identified building block that shall carry this additional information. The following example shows how to get a group of extension fields onto the tab *General Data* of the Freight Order UI.

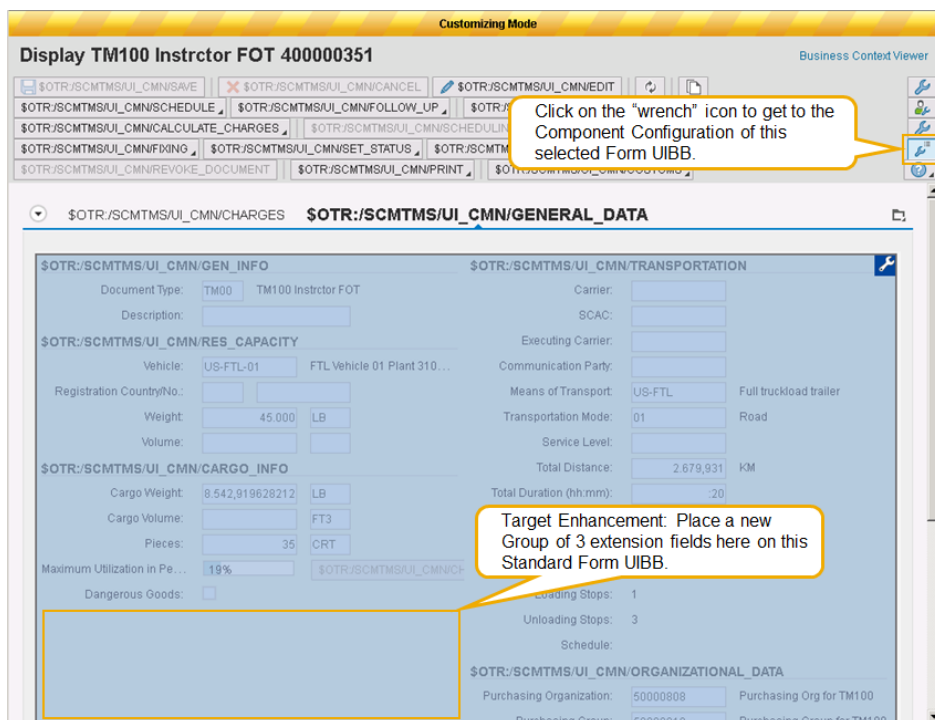
- 1) Start the Freight Order UI for displaying Freight Orders from the SAP user menu and set the configuration mode in the URL (Parameter `sap-config-mode = X`) for the UI as described in section 5.3. Click on button *Show Customizable Areas*.



Picture: Switching on Customizing Mode for the Freight Order Display UI.

Alternative: Start the same UI within NWBC and use the technical help as described in section 5.3 to navigate to the component customizing.

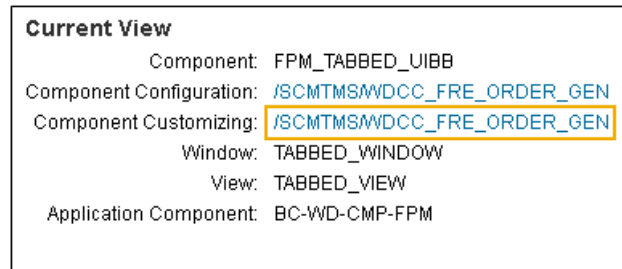
- 2) Enter an existing Freight Order number and display the document by clicking on button *Continue*. On the next screen display tab *General Data* (a FORM UIBB). Move the mouse over the top most area of the displayed tab to highlight the first Form UIBB within this tab.



Picture: Navigating to the Configuration to be enhanced.

Target of this example is to add a new group of 3 extension fields to the selected Form UIBB. To start the Component Configurator, click on the “wrench” icon as shown in the picture above. The name of the standard configuration of that we will enhance in this example is: `/SCMTMS/WDCC_FRE_ORDER_GEN_GNINF`.

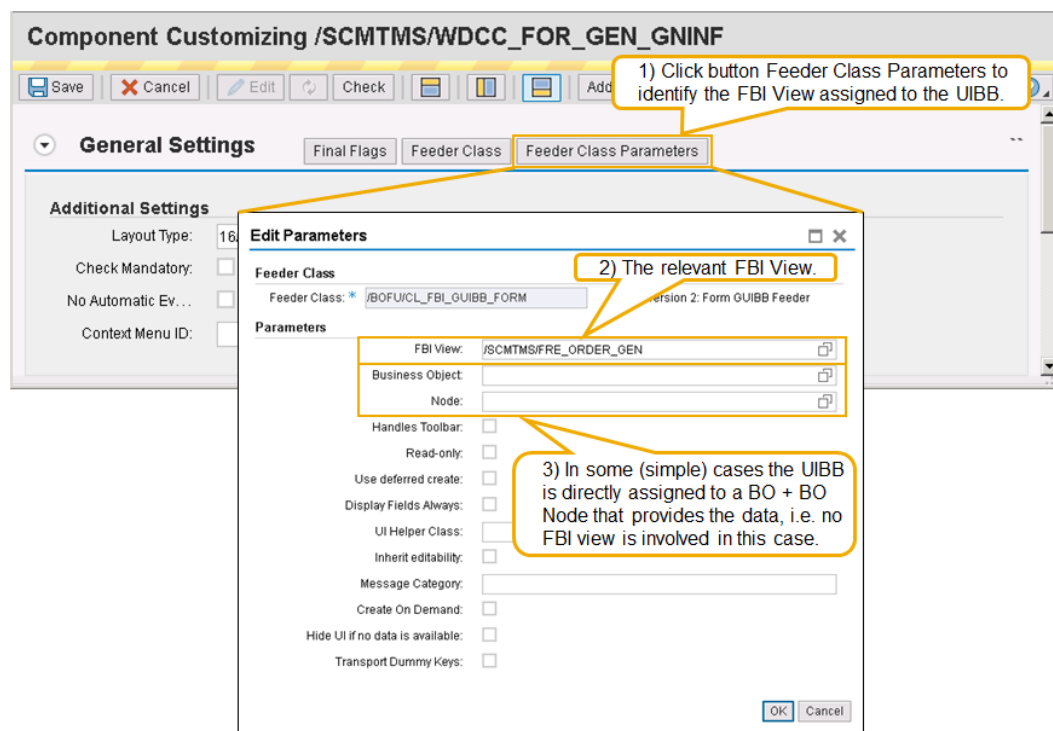
Alternative: Using the technical help e.g. within NWBC to identify the correct standard configuration to be enhanced. If you use this for the example, make sure that you position the mouse in the top most part of the *General Data* tab of the Freight Order UI and in the technical help, section Current View click on the link Component Customizing that will open the Configuration Editor for `/SCMTMS/WDCC_FRE_ORDER_GEN_GNINF`. Always make sure that the Editor is in the Edit Mode to allow creating enhancements.



Picture: Technical Help.

3) In the configuration `/SCMTMS/WDCC_FRE_ORDER_GEN_GNINF` open section *General Settings* and click on button *Feeder Class Parameters*. Here you can find information which is relevant for the next steps:

- **FBI View:** The FBI view which is assigned to the current UI building block. As described in section 2.3, the FBI view besides other information holds the information on the UI structure of a building block as well as the related business object node.
- **Business Object & Node:** In some cases the UI building block is directly assigned or related to a specific business object node. If so, extension fields can directly be added to this node as described in section 3.4.3 and can afterwards directly get included in the UI layout. In this case, just continue with step 5.



Picture: Feeder Class Parameters of the current UIBB configuration.

Example (see also picture above): FBI View /SCMTMS/FRE\_ORDER\_GEN is used, i.e. no direct relation to a business object node is used. In this case we need to display the FBI View to find out the business object node related to this UIBB (step 4). In other words, we take a look at the FBI View to find the BO / BO Node that provides the data to this UIBB and that needs to be extended in the backend to hold our extension fields.

#### 4) Display the identified FBI View /SCMTMS/FRE\_ORDER\_GEN.

Alternative 1:

- Start transaction *SE84* and follow the following path: Repository Information System → Web Dynpro → Component Configurations.
- On the selection screen enter /SCMTMS/FRE\_ORDER\_GEN in field Component Configuration. Press *F8*.
- In the following list mark the found entry. Press *F7*.
- Click on button *Display Configuration* (or click on button *Start Configurator* and then on button *Continue in Display Mode*).

Alternative 2:

- Use the Web Dynpro application mentioned in section 5.4 to display the identified FBI View (→ search by Configuration /SCMTMS/FRE\_ORDER\_GEN).

Picture: Header details of the FBI View.

**Example:** Our FBI View /SCMTMS/FRE\_ORDER\_GEN is related to the business object /SCMTMS/TOR (Freight Order), ROOT node. The name of the corresponding Node UI structure is /SCMTMS/S\_UI\_FRE\_ORDER\_GEN.

#### 5) Add your extension fields to the identified Business Object node as described in section 3.4.3, i.e. you create an append structure with the extension fields for the corresponding extension include.

Example: Create extension fields for the ROOT node of business object /SCMTMS/TOR.

Business Object Information		Comment
Business Object	/SCMTMS/TOR	Technical name of the business object.
Extension Include	/SCMTMS/INCL_EEW_TOR_ROOT	The persistent Extension Include.
Append Structure		
Name	ZENH_DEMO_TOR_ROOT	

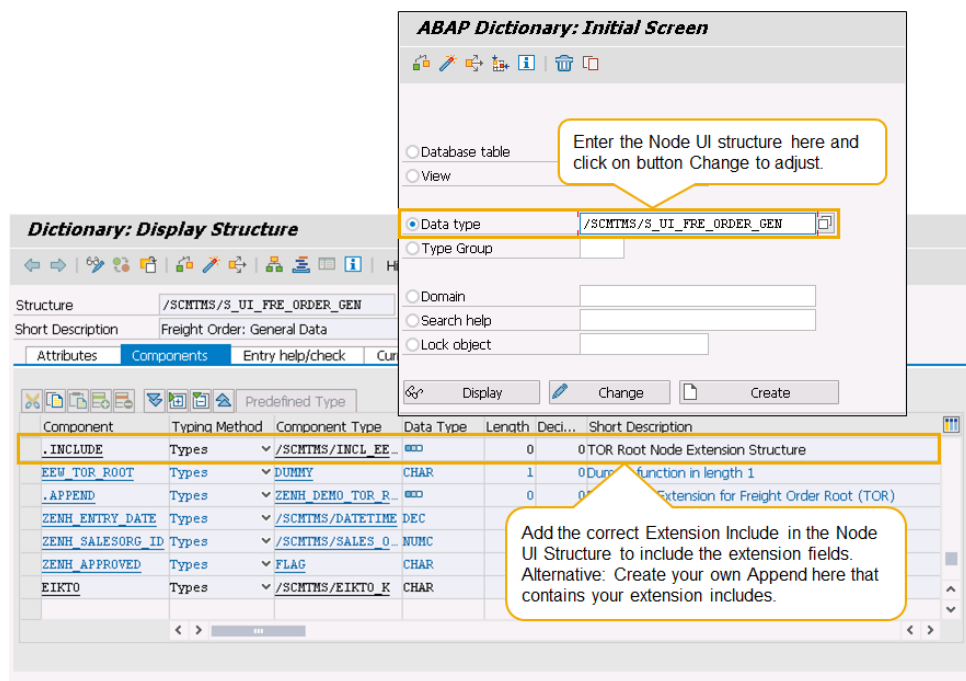
Description	Demo Field Extension for Freight Order (TOR)	
<b>Component</b>	<b>Typing Method</b>	<b>Component Type</b>
ZENH_ENTRY_DATE	Types	/SCMTMS/DATETIME
ZENH_PURORG_ID	Types	/SCMTMS/PURCH_ORG_ID
ZENH_APPROVED	Types	FLAG

In the easiest case the new fields are now already available in the field catalog of the required building block. But this is only the case if the UI structure mentioned in the related FBI View automatically includes the extension includes of the related business object node. If so, you can continue with step 7.

- 6) Check and enhance the identified UI node structure if required.

**Example:** For displaying Node UI structure `/SCMTMS/S_UI_FRE_ORDER_GEN`, use transaction `SE11`. Check if your extension fields are already available there. Assumption: The mentioned Node UI structure does not automatically include the required Extension Includes. In this situation you have two options to ensure that the extension fields are available in the field catalog of the UIBB to be enhanced.

- Add the Extension Include `/SCMTMS/INCL_EEW_TOR_ROOT` of Business Object `/SCMTMS/TOR, ROOT` Node to the UI structure. This is quite easy and works but can eventually have a (UI) performance impact: If you enhance the mentioned BO Node with further fields that you do not need in the context of this specific Node UI Structure / UIBB, they nevertheless cause some overhead when the Node UI structure is used at runtime.



Picture: Enhancing the Node UI Structure.

- Alternative: Create your own Append in the mentioned Node UI Structure and only add those fields there which are actually required for the UIBB to be enhanced. In other words, the Node UI Structure will only contain fields that are actually used, preventing the overhead caused by other extension fields that might have been added to the Extension Include for different purposes.



7) Use the extension fields to enhance the configuration of the UIBB.

The extension fields are now available in the field catalog of the UIBB to be enhanced. In this step, the configuration is enhanced by placing the new fields on the layout of the UIBB and defining the required field properties.

**Example step 1:** On the General Data tab, we add a new group to carry the new extension fields. In the configuration /SCMTMS/WDCC\_FRE\_ORDER\_GEN\_GNINF click on button *Add Group*.

Picture: Adding a new field group to the FORM UIBB.

For this group, provide the following information.

Field	Content	Comment
Text	Demo Enhancement Group	The header text to be displayed for the new group.
Start Row / Column of Element	13 / A	The starting row and column of the Layout position for the new Group.
End Row / Column of Element	13 / H	The ending row and column of the Layout position for the new Group.



Picture: Defining attributes & properties of a screen element.

Example: Now we add the extension fields to the new group. In the configuration /SCMTMS/WDCC\_FRE\_ORDER\_GEN\_GNINF navigate to section *Form UIBB Schema*. Then on the sections tool bar click on button *Element* and choose *Add Element at Next Level*.

Picture: Adding elements to the new Group.

On the following popup screen you can find the list of available fields, including the extension fields that were added backend in the previous steps. Mark the required extension fields to be added to the new group and then click on button *OK* to add them.

Picture: Assigning extension fields to the new Group.

**Component Customizing /SCMTMS/WDCC\_FOR\_GEN\_GNINF**

Form UIBB Schema

With the buttons Up and Down you can adjust the sequence of fields within the Group.

Select one of the new fields to specify its attributes and properties

Attributes of Element: ZENH\_ENTRY\_DATE

Standard Attributes

Element

Field Name: ZENH\_ENTRY\_DATE

Explanation:

Label Text: Date/Time

Tag is Active: ☐

Display Type: Input Field

Tooltip:

Label Visibility: Is Visible

Context Menu ID:

Position

Start Row of Label:

End Row of Label: 14

Start Row of Element: 14

End Row of Element: 14

End Col. of Label: C

Start Column of Element: D

End Col. of Element: H

Display-Type-Dependent Properties

Alignment: Automatic

Filter Method: prefixSearch

UI Element Width:

Password: ☐

Suggest Values: ☐

Search Help:

Action Assignment

FPM Event ID for onEnter: No Action Assigned

Picture: Specifying the attributes and properties of an element.

- You can adjust the sequence of the extension field in the new Group by using the buttons *Up* and *Down* in section Form UIBB Schema.
- The attributes and properties of a selected element (in this case e.g. a new extension field) can be maintained by marking it in the *Form UIBB Schema* (see picture above) and then specifying the different attributes and properties of this element in the screen section below.
- Example:** The first field in the new group is a flag. Change its *Display Type* to *Check Box* to display the new field as a check box on the screen. In field *Tooltip* add a tooltip for the field (e.g. "Approval for Enhancements").

#### 8) Adjusting the Label Texts.

In the example shown, it is not possible to directly enter a *Label Text* for the new field. Compared to NW 7.02 this feature has changed in NW 7.31 and higher releases. In the earlier NW releases you could just simply enter a Label Text directly in the configuration.

In the higher NW releases, the *Label Text* is first of all automatically taken over from the data element used for the definition of the extension field in the backend, i.e. it is pulled from *DDIC*.

But you can create your very own specific *Label Texts*, assign them to your extension fields and even have the chance to get language specific variants of these *Label Texts* which can be handled with the available translation tools available in the backend. Just like standard SAP TM development, you can create so called *OTR Texts* via transaction *SOTR\_EDIT* and assign these *OTR Texts* to extension fields in the FBI View related to the UIBB that you extend. The procedure how to do this is described in the following sections.

- 9) Finally, save the Component Configuration by clicking on button Save. The system might ask you for a transport request on a following popup. Select an existing or create a new transport request there and save your enhancements. The enhancements can now be tested by restarting the UI.

Test the enhancement: Display an example Freight Order. Make sure that you switch of the *Customizing Mode* before (sap-config-mode = [space]).

- Check if the new Group is shown on the correct UIBB with the assigned fields.
- Bring the displayed document into edit mode and enter some data in the new fields.
- Save the document, close the UI and reopen it again by displaying the example Freight Order again to check that the content of the new extension fields get persisted.

**Display TM100 Instructor FOT 400000351**

Business Context Viewer

Save Cancel Edit Schedule Follow Up Check Calculate Charges Scheduling Subcontracting Fixing Set Status Set Item Status Cancel Document Print Customs

**General Data** Notes Business Partner Terms and Conditions Document Flow Charges

**General Information** **Transportation**

Document Type: TM00 TM100 Instructor FOT Carrier: SCAC: Executing Carrier: Communication Party: Service Level: Total Distance: 2.679,931 KM Total Duration (hh:mm): :20 First Activity: 13.09.2012 01:39 Last Activity: 13.09.2012 00:00 Number of Visits: 4 Loading Stops: 1 Unloading Stops: 3 Schedule:

**Resource Capacity**

Vehicle: US-FTL-01 FTL Vehicle 01 Plant 3100... Full truckload trailer Road

Registration Country/No.: Weight: 45.000 LB Volume: 45.000

**Demo Enhancement Group**

Date/Time: 20.09.2013 15:00:00 Indicator: ☒ Sales Organization: 50500545

**Cargo Information**

Cargo Weight: 8.542,919628212 LB Cargo Volume: FT3 Pieces: 35 CRT

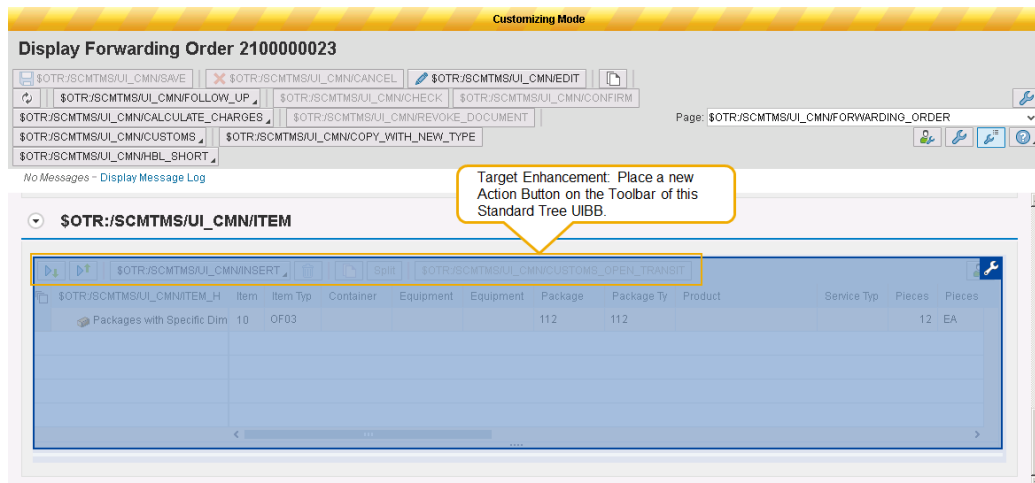
The new Group with the example extension fields displayed with an existing Freight Order document after editing and having saved the document again

Picture: The new group with the extension fields on the UI.

### 5.4.2 Adding a new action to a toolbar

A second example for extending the user interface is adding additional actions on a toolbar. The following example shows how to get a new action onto the toolbar of the tab *Items* of the Forwarding Order UI.

- 1) Start the User Interface for displaying Forwarding Orders from the SAP user menu and set the configuration mode in the URL (Parameter sap-config-mode = X) for the UI as described in section 5.3. Click on button *Show Customizable Areas*.
- 2) Display an existing Forwarding Order and click on tab *Items* (a TREE UIBB). Move the mouse over the top most area of the displayed tab to highlight the first Form UIBB within this tab.



Picture: The UIBB to be enhanced on the Forwarding Order UI Items Tab.

Target of this example is to add a new *Action Button* on the Toolbar of the *Items* Tree UIBB that shall execute an action for a selected *Forwarding Order Item*. To start the Component Configurator, click on the “wrench” icon as shown in the picture above. The name of the standard configuration of that we will start the enhancement from in this example is:

`/SCMTMS/WDCC_FWD_ORDER_ITM.`

Alternative: Using the technical help e.g. within *NWBC* to identify the correct standard configuration to be enhanced. If you use this for the example, make sure that you position the mouse on the *Tree UIBB* of the *Items* tab on the Forwarding Order UI. In the technical help, section *Current View* click on the link *Component Customizing* that will open the Configuration Editor for `/SCMTMS/WDCC_FWD_ORDER_ITM`. Always make sure that the Editor is in the Edit Mode to allow creating enhancements.

- 3) The system will start the Configuration Editor. If required and not yet done, create the Component Customizing for this configuration as described in section 5.3. In the Configuration Editor display section *Preview*. Here you can see the Toolbar to be enhanced, i.e. it is directly located in this configuration.
- 5) In the configuration `/SCMTMS/WDCC_FWD_ORDER_ITM` open section *General Settings* and display the *Feeder Class Parameters* just like described in 5.4.1. Here you can find information which is relevant for the next steps:

FBI view `/SCMTMS/TRQ_ITM` is assigned to the current UIBB. As described in section 3.3, the FBI view besides other information holds the information on the UI structure of a UIBB as well as the related business object node, etc.

Interesting in this example is the fact that the mentioned FBI View is an example for the usage of *Related Views* as mentioned in section 5.2.1. When displaying the FBI View above click on tab *Related Views* where you can find the following FBI View:

*/SCMTMS/TRQ\_ITM\_SEAL*

It provides additional data to be displayed along with the data provided in the main FBI View. The assigned BO node is *ITEMSEAL* which holds Seal Information for specific Forwarding Order Items.

- 6) Display the found FBI View */SCMTMS/TRQ\_ITM* (as described in section 5.2, step 6) to identify the BO and its node that it is related to.

Picture: Identifying the BO and BO node to be enhanced.

For our example, the BO is */SCMTMS/TRQ* (i.e. the Forwarding Order) and the node is *ITEM* (the item node of the Forwarding Order) → So this FBI View is provided with data from this specific BO Node which also propagates its Actions to be available in related UIBB configurations.

- 7) Now use the BOPF Enhancement Workbench to create a new enhancement action as described in section 3.3.6. Example: Action *ZENH\_TRQ\_ITEM\_DEMO\_ACTION* with the implementing class *ZCL\_ENH\_A\_TRQ\_ITEM\_DEMO\_ACTION*. Make sure that the Action Cardinality is set to *Multi Node Instances* and the flag *Action can be enhanced* is set.

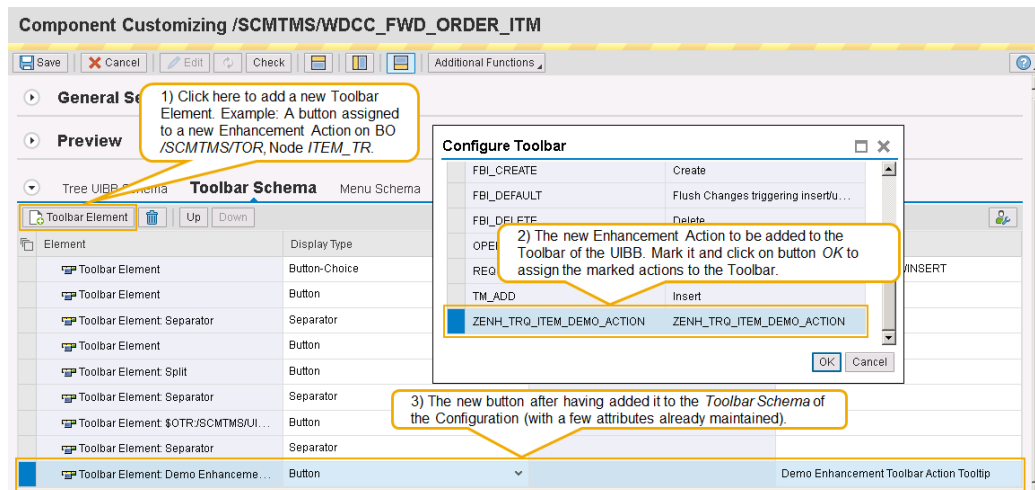
The example implementation for its method *EXECUTE* looks as follows:

```
METHOD /bobf/if_frw_action~execute.
  DATA: lv_temp TYPE c.
  MESSAGE s008(/scmtms/ui_messages) INTO lv_temp.
  CALL METHOD /scmtms/cl_common_helper=>msg_helper_add_symsg
    EXPORTING
      iv_key      = /scmtms/if_tor_c=>sc_bo_key    " Instance Key
      iv_node_key = /scmtms/if_tor_c=>sc_node-root " BO Key
  CHANGING
    co_message = eo_message. " Current message object
ENDMETHOD.
```

This very simple example action implementation takes a simple info message from message class `/SCMTMS/UI_MESSAGES` and puts it into the BOPF message object `EO_MESSAGE`. When you execute the new enhancement action the issued message will also be shown on the User Interface. This works of course for any message from any message class.

- 8) Use the new action to enhance the configuration of the toolbar. Remember that we do this enhancement on configuration `/SCMTMS/WDCC_FWD_ORDER_ITM`.

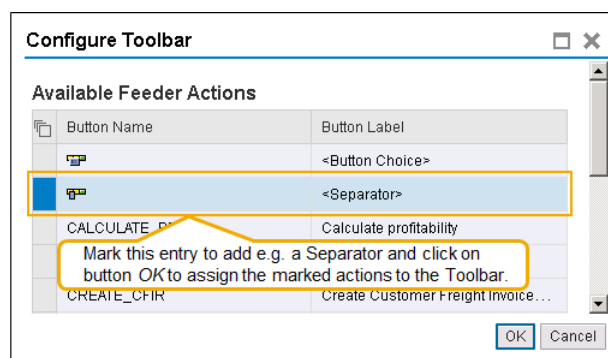
The action is now available in the action catalog for the toolbar to be enhanced. In this step, the configuration is enhanced by adding the new action to the toolbar and defining the required action properties.



Picture: Adding a new Action to the Toolbar of a UIBB.

**Example:** In configuration `/SCMTMS/WDCC_FWD_ORDER_ITM` navigate to section *Toolbar Schema* (see picture above) and click on button *Toolbar Element* to add the action `ZENH_TRQ_ITEM_DEMO_ACTION` we created in the previous step. On the following popup screen you can find the list of available actions, including the example action. Mark the action to be added and click on button **OK** to assign it to the Toolbar.

- 10) You can also add *Separators* between the standard actions and enhancement actions. To do this click on button *Toolbar Element* in section *Toolbar Schema* again, mark the corresponding entry for a *Separator* and click on button **OK**.



Picture: Adding a Separator.

Actually, with the same procedure you can not only add buttons and separators but even other Toolbar Elements like *Input Fields*, *Toggle Buttons*, *Button Choices*, etc. In column *Display Type* of section *Toolbar Schema*, you can use the drop-down list to define what kind of element an entry in the list shall represent. Depending on the type, in the screen section below corresponding properties and attributes will be displayed and can be defined there to further specify the chosen element.

- 11) If required, adjust the sequence of the actions in the list of Toolbar Elements in section *Toolbar Schema*. This is done with the buttons *Up* and *Down*. In the example place e.g. the added separator before the enhancement action button.
- 12) In this example, we have added a new button that is related to a BO Node Action. You can now maintain the corresponding attributes for the new action e.g. as shown in the following picture:

Picture: Define properties and attributes for the new Toolbar Element.

**Note:** The attribute *FPM Event ID* is filled with the name of the Action that we have implemented and added to this UIBB Toolbar. In this case *FPM/FBI* will automatically find and execute the right BO Node Action. Assume you overwrite the *FPM Event ID* with your very own ID (e.g. *MY\_ACTION\_ID*). In this case you would have to implement the behavior of the button in the Exit Class of the related FBI View, method *ADAPT\_EVENT* (see section 5.2.5). There you can then define via coding what shall be executed on clicking the new button.

- 13) Finally, save the Component Configuration by clicking on button *Save*. The enhancement action is now available as a button on Toolbar for the *Items Tab* of the Forwarding Order UI which displays the Forwarding Order items.

Test the enhancement: Make sure that you switch off the *Customizing Mode* before (sap-config-mode = [space]). Display an existing Forwarding Order (that contains items) and mark one or more of its items. Then click on the new button. After execution you can see the message in the message log that we implemented to be issued in the related BO Node Action.

Picture: The new action on the toolbar.



### 5.4.3 Adding a new tab with data from a new BO subnode

The third example will show a more complex configuration enhancement. A new subnode for the Freight Order will be created. The data of this subnode shall be displayed as a list on a tab, including a toolbar with actions that can be executed on the available data.

- 1) Create a new subnode for the Freight Order as a subnode of its Root Node with a cardinality of 1:N. We will use the example subnode `ZENH_ROOT_SUBNODE` as described in section 3.3.5 for the next steps.
- 2) Start the editor for the Web Dynpro ABAP Component Configuration to create a new configuration (in this example we create a completely new one). The editor can be started via the following general link which has to be enhanced with information on server and port, depending on where you want to start the tool.

[https://\[server\]:\[port\]/sap/bc/webdynpro/sap/configure\\_component](https://[server]:[port]/sap/bc/webdynpro/sap/configure_component)

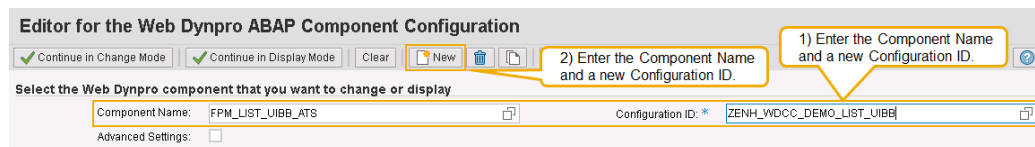
Example:

[https://ugaai9x.wdf.sap.corp:18562/sap/bc/webdynpro/sap/configure\\_component](https://ugaai9x.wdf.sap.corp:18562/sap/bc/webdynpro/sap/configure_component)

Here you can enter a component name and a new Configuration ID.

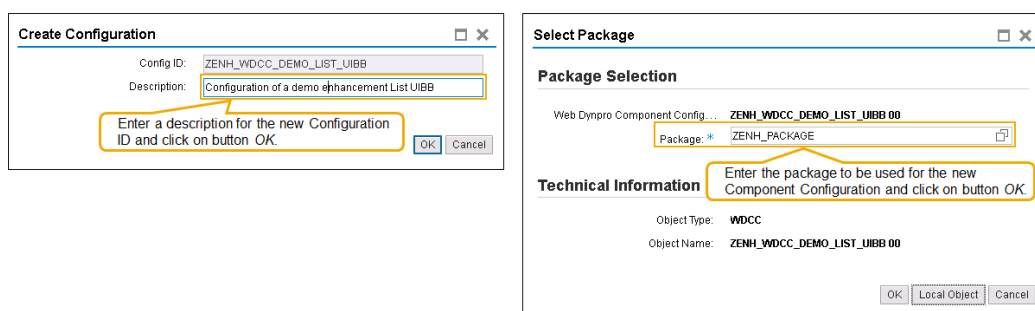
Component Name	<code>FPM_LIST_UIBB_ATS</code>	The new configuration shall represent a list which will contain data from a new BO subnode.
Configuration ID	<code>ZENH_WDCC_DEMO_LIST_UIBB</code>	This will be the new configuration to be integrated in the Freight Order UI.

Click on button *New* to create the new configuration. Note that this new configuration is a completely new object that is not part of the SAP TM Standard but is owned by the customer or partner that implements it.



Picture: The initial screen of the Component Configuration Editor

- 3) On the following popups, specify a description for the new configuration as well as a package where to store it, e.g. as a local object or assign it to your customer/partner specific transportable package. If you assign the new Component Configuration to a transportable package you will be asked to specify a corresponding transport request.



Picture: Specifying a description and a package for the new Component Configuration.

- 4) On the next popup specify the feeder class that shall be used for the new configuration and its related UIBB. For this example, we use a predefined feeder class for our new configuration: Class `/BOFU/CL_FBI_GUIBB_LIST_ATS`.

Picture: Create the configuration and define its feeder class.

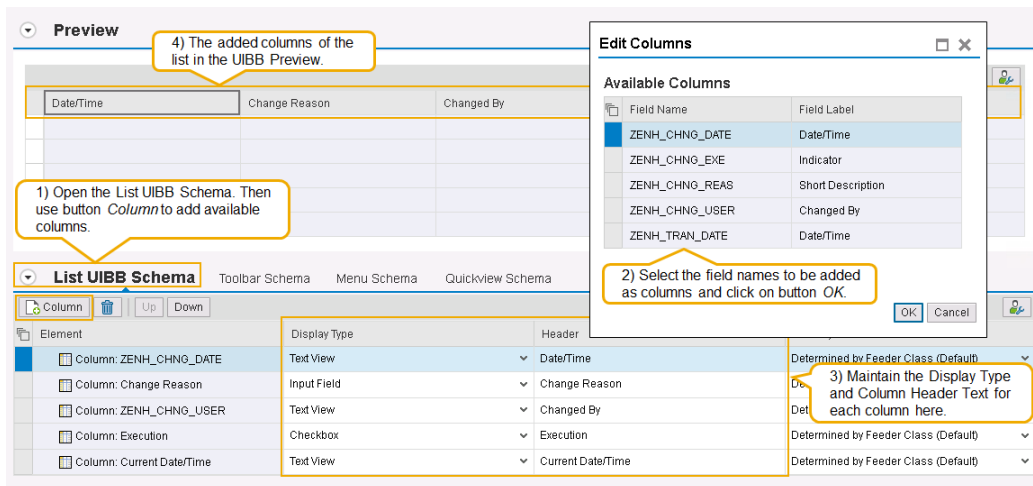
- 5) On the same popup click on button **Edit Parameters** to configure further parameters of the configuration. The following parameters define the data source from where our new list configuration will take the information to be displayed. They define the parameters that will be used by the feeder class in this configuration.

Business Object	<code>/SCMTMS/TOR</code>	The Freight Order BO.
Node	<code>ZENH_ROOT_SUBNODE</code>	Our new subnode that was assigned to the Freight Order Root Node in step 1.

Picture: Parameters to be used by the feeder class.

Click on button **OK** to get to the next configuration step.

- 6) In the next step further list elements and attributes are configured i.e. adding the columns to be displayed with the list and actions to be available on the list toolbar.



Picture: Adding columns to the List UIBB Schema.

For adding columns to the List UIBB navigate to the *List UIBB Schema* and click on button *Column*. On the following popup you will find all available attributes from the BO node that was assigned to this Component Configuration in step 5. Select the required fields to be added as columns and click on button *OK*. In the example we take over all fields available on the assigned BO node.

In the List UIBB Schema table you can now see all assigned columns. Here you can now further maintain the Display Type (e.g. Text View, Input Field, Checkbox, etc.) and Header (column header text that will appear on the UI) for each column. These and further attributes of a selected column can also be configured in the section below the list:

**Attributes of Toolbar Element** Final Flags X

**Standard Attributes**

Display Type:  Context Menu ID:

Index:  Explanation Text:

Tooltip:

**Display Type Dependent Properties**

Hide Text: ☐ Hotkey:

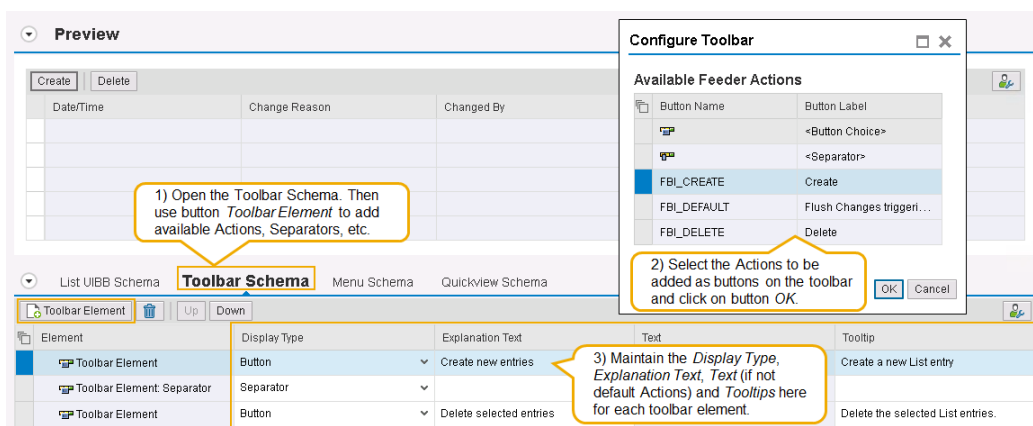
Image Source:  ToolbarButton Design:

**Action Assignment**

FPM Event ID:  Text:

Picture: Specifying the attributes of a column.

In this section of the Configuration Editor you can specify all available attributes that determine the behavior and appearance of the column selected in the list above. Besides the mentioned attributes e.g. Tooltips can be defined, F4-Search Helps assigned, etc.



Picture: Adding Actions to the Toolbar Schema.

For adding Actions to the Toolbar of the List UIBB navigate to the *Toolbar Schema* and click on button *Toolbar Element*. On the following popup you will find all available *FBI Standard Actions* (e.g. *Create*, *Delete*) and all Actions from the BO node that was assigned to this Component Configuration in step 5. Select the required Actions to be added as buttons in the Toolbar and click on button *OK*. In the example we just take over the two FBI Standard Actions *Create* and *Delete*. Other Actions assigned to the related BO node will be listed here as well and can be added the same way.

In the Toolbar Schema table you can now see all assigned Actions. Here you can now further maintain the *Display Type*, *Explanation Text*, *Text* and *Tooltip* for each toolbar element.

**Attributes of Toolbar Element**

**Standard Attributes**

Display Type:  Context Menu ID:

Index:  Explanation Text:

Tooltip:

**Display Type Dependent Properties**

Hide Text: ☐ Hotkey:

Image Source:  ToolbarButton Design:

**Action Assignment**

FPM Event ID:  Text:

Picture: Specifying the attributes of a column.

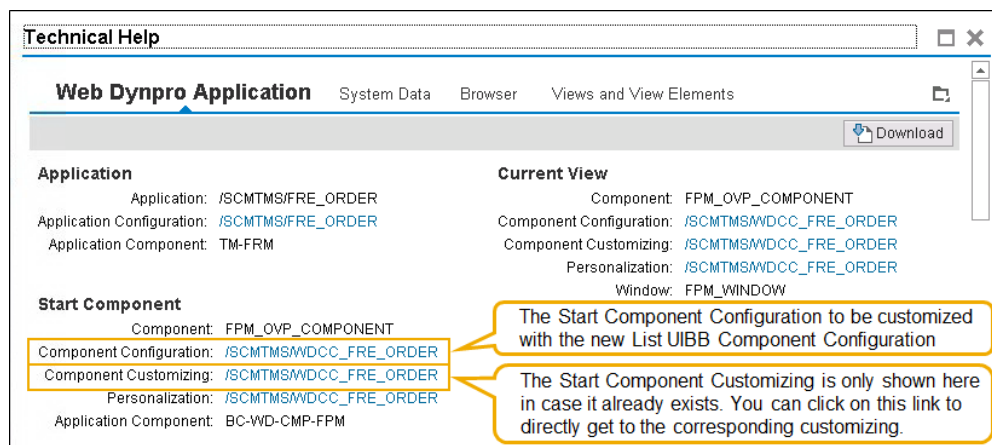
In this section of the Configuration Editor you can specify all available attributes that determine the behavior and appearance of the Toolbar Element selected in the list above. Besides the mentioned ones, you can e.g. a Hotkey, an Image Source for Icons to be assigned to a button, etc.

- 7) Save the new Component Configuration for the List UIBB.



With saving the new Component Configuration *ZENH\_WDCC\_DEMO\_LIST\_UIBB* it is now available to be integrated into the standard Freight Order UI. The next steps describe how to do this.

- 8) For integrating the new configured UIBB into the Freight Order UI you can start the UI in configuration mode (*sap-config-mode=X*). Here you can use the Technical Help (context menu via right mouse button) to identify the Start Component Configuration. In this example it is Component Configuration */SCMTMS/WDCC\_FRE\_ORDER*.



Picture: The Technical Help with links to Component Configurations.

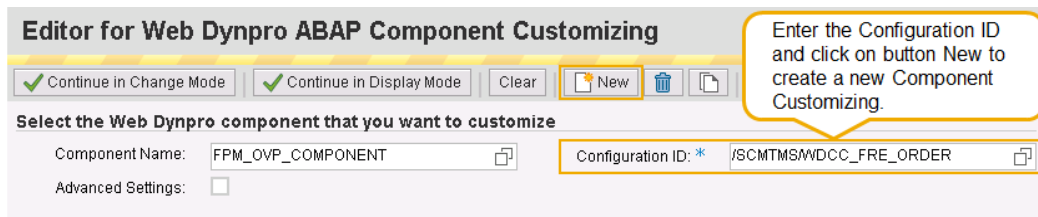
If there is already a Component Customizing for this Component Configuration, you can directly click on the link in the Technical Help popup (Start Component → Component Customizing). Otherwise you need to first of all create it as follows.

In this case you can start the Configuration Editor in customizing mode via the following general link which has to be enhanced with information on server and port, depending on where you want to start the tool.

[https://\[server\]:\[port\]/sap/bc/webdynpro/sap/customize\\_component](https://[server]:[port]/sap/bc/webdynpro/sap/customize_component)

Example:

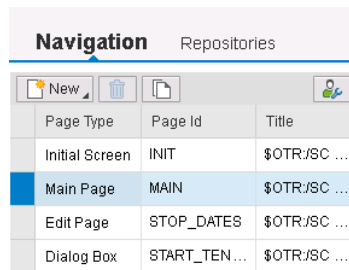
[https://ugaai9x.wdf.sap.corp:18562/sap/bc/webdynpro/sap/customize\\_component](https://ugaai9x.wdf.sap.corp:18562/sap/bc/webdynpro/sap/customize_component)



Picture: Creating a new Component Customizing.

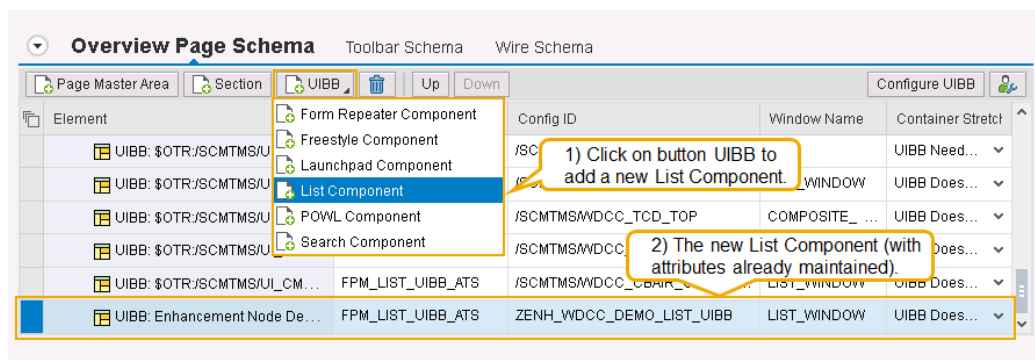
On the following popup specify a description for the new Component Customizing and click on button OK. Then assign the Component Customizing to a Transport Request if required.

- 9) On the right side of the Configuration Editor click on tab *Navigation* to display the available pages of the UI. Select the *Main Page* in the list and navigate to the Overview Page Schema (Configuration /SCMTMS/WDCC\_FRE\_ORDER configures an *Overview Page Component*) on the right side of the configuration editor.



Picture: Main Page selected.

- 10) On the Overview Page Schema add a new List Component. To do this, click on button *UIBB* and select entry *List Component*. In the table with the Overview Page Schema elements you can then find a corresponding entry. Mark this entry in the table to display and specify the attributes of the new UIBB.



Picture: Adding a new List Component to the Overview Page Schema.

- 11) Specify the attributes of the new List Component. For the example the following attributes are specified:

Attribute	Value
Component	<i>FPM_LIST_UIBB_ATS</i>
Window Name	<i>LIST_WINDOW</i>
Configuration ID	<i>ZENH_WDCC_DEMO_LIST_UIBB</i>
Instance ID	<i>1</i>
Title	<i>Enhancement Node Demo Data</i>

**Attributes of UIBB: Enhancement Node Demo Data**

**Standard Attributes**

Component: \* FPM\_LIST\_UIBB\_ATS

Window Name: \* LIST\_WINDOW

Config ID: ZENH\_WDCC\_DEMO\_LIST\_UIBB

Instance ID: 1

Column: 1

Row Number: 1

Sequence Index: 24

Container Stretching: UIBB Does Not Need Surrounding Contain

Hidden Element: Visible

Title: Enhancement Node Demo Data

Explanation Text:

Explanation Document:

Default: ☐

Default Edit Page:

Default Details Page:

Padding: Automatic (Default)

In the example you need to specify here the component, window name, Configuration ID (i.e. the name of the new UIBB's configuration) and Instance ID = 1.

This title will be displayed on the tab strip that the new UIBB will be displayed in.

Picture: Specifying the attributes of the new List Component.

- 12) Now switch to the Wire Schema. In this final step, we need to define where the new UIBB gets the data to be displayed from. This is done by declaring a so called wire which connects the new UIBB with the UI parts which already carry information of the current BO instance that is displayed. In our example, the new UIBB needs to know from which instance and which node of the Freight Order BO the data to be displayed is taken.

Click on button *Wire* to add a new wire. Then maintain the attributes for the new wire as listed below (note: the correct wiring is essential to make the new UIBB work).

**Wire Schema**

Transaction: 1) Click on button *Wire* to add a new Wire for connecting the new UIBB with the standard UI.

**Wire**

Graphical Wire Editor

Element	Component	Config ID	Port Type	Source Component	Source Config Name
Wire: Form /...	FPM_FORM_UIBB_...	/SCMTMS/WDCC_T ...	Collection	FPM_FORM_UIBB	/SCMTMS/WDCC_T ...
Wire: Tree /...	FPM_TREE_UIBB	/SCMTMS/WDCC_F ...	Collection	FPM_FORM_UIBB	/SCMTMS/WDCC_T ...
Wire: Form /...	FPM_FORM_UIBB_...	/SCMTMS/WDCC_F ...	Collection	FPM_FORM_UIBB	/SCMTMS/WDCC_T ...
Wire: List/S...	FPM_LIST_UIBB_ATS	/SCMTMS/WDCC_T ...	Lead Selection	FPM_FORM_UIBB	/SCMTMS/WDCC_T ...
Wire: List/S...	FPM_LIST_UIBB_ATS	/SCMTMS/WDCC_C ...	Collection	FPM_FORM_UIBB	/SCMTMS/WDCC_T ...
Wire: List ZE...	FPM_LIST_UIBB_ATS	ZENH_WDCC_DEM ...	Collection	FPM_FORM_UIBB	/SCMTMS/WDCC_T ...

2) The new Wire (with attributes already maintained).

Picture: Adding a new wire in the Wire Schema

The following table contains the Wire Attributes and the corresponding values for the given example. In general, a wire has a target as well as source component and configuration. The target is our new List UIBB with its configuration that is based on the new sub node *ZENH\_SUBNODE*. The source in this example is the initial screen of the Freight Order UI which carries the Freight Order number as the initial information. This source is based on the Root node of the Freight Order BO.

At runtime the Root node contains the instance of the Freight Order which is processed. Based on this instance, the wire allows navigation to the corresponding subnode defined in the target. For this the BOBF association between the Root node and subnode *ZENH\_SUBNODE* is used. With this, the new List UIBB can now get the corresponding data from the sub node of the Freight Order instance.

Attribute	Value	Comment
Component	<i>FPM_LIST_UIBB_ATS</i>	The generic <i>List UIBB</i> provided by FPM, i.e. the target component of the wire.
Configuration ID	<i>ZENH_WDCC_DEMO_LIST_UIBB</i>	The example configuration for the new <i>List UIBB</i> , i.e. the target configuration of the wire.
Instance ID	<i>1</i>	
Source Component	<i>FPM_FORM_UIBB</i>	The source component of the wire. In this example it is a <i>FORM UIBB</i> .
Source Configuration Name	<i>/SCMTMS/WDCC_TOR_INIT_SCREEN</i>	The source configuration of the wire. In this example it is the configuration for the initial screen form of the Freight Order UI.
Source Node Association	<i>ZENH_ROOT_SUBNODE</i>	This is the association that is defined between the node of the wire source and the node of the wire target. In this example it is the composition association between the Freight Order (TOR) Root node and our new subnode <i>ZENH_SUBNODE</i> .
Port Type	<i>Collection</i>	
Port Identifier	<i>CO</i>	
Connector Class	<i>/BOFU/CL_FBI_CONNECTOR</i>	Provides basic functions to connect FPM, FBI and BOBF.

**Attributes of Wire: List ZENH\_WDCC\_DEMO\_LIST\_UIBB**

Standard Attributes

Component: \* *FPM\_LIST\_UIBB\_ATS*

Config ID: *ZENH\_WDCC\_DEMO\_LIST\_UIBB*

Instance ID: *1*

Source Component: *FPM\_FORM\_UIBB*

Source Config Name: */SCMTMS/WDCC\_TOR\_INIT\_SCREEN*

Connector Parameters

SRC\_NODE\_ASSOC: *ZENH\_ROOT\_SUBNODE*

Port Type: *Collection*

Port Identifier: *CO*

Connector Class: \* */BOFU/CL\_FBI\_CONNECTOR*

Make sure to specify all attributes (as shown in the example) to make the wiring work.

Picture: Specifying the attributes of the new wire.

- 13) Save your configuration. The new List UIBB with its configuration is now ready to be used on the Freight Order UI. The list is shown as a tab and allows displaying as well as editing instances of sub node *ZENH\_SUBNODE* for a given Freight Order instance.

**Edit TM100 North America OB - GRP01 100000050**

Business Context Viewer

Save Cancel Edit Schedule Follow Up Check Calculate Charges Scheduling Subcontracting Fixing

Set Status Set Item Status Cancel Document Print Customs

General Data **Enhancement Demo Data** Notes Business Partner Terms and Conditions Document Flow

Create Delete

Date/Time	Change Reason	Changed By	Execution	Current Date/Time
07.10.2013 14:00:00	Was ok but not perfect	POLCH	<input checked="" type="checkbox"/>	
09.10.2013 18:30:00	A bit better now	POLCH	<input checked="" type="checkbox"/>	
10.10.2013 10:45:00	Perfect!	POLCH	<input type="checkbox"/>	

1) The new list embedded in the standard Freight Order UI.

2) The new list as configured in the example, here shown in Edit Mode and the assigned actions ready for use.

Picture: The final List UIBB embedded in the Freight Order UI.



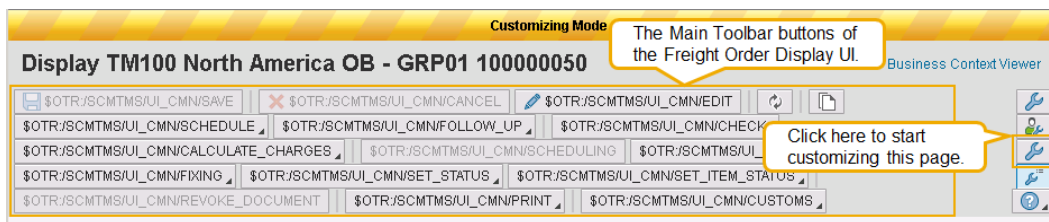
#### 5.4.4 Adding a new Action to the main tool bar

Adding a new action to the main tool bar of an FPM application works slightly different than adding an action to a component tool bar as shown in section 5.4.2.

For the following first example let's assume we have already added an enhancement action `ZENH_MAINTOOLBAR_ACTION` on the ROOT node of the Freight Order BO (TOR) that we would like to trigger from the main tool bar of the Freight Order UI. The coding for this action implementation could look as follows:

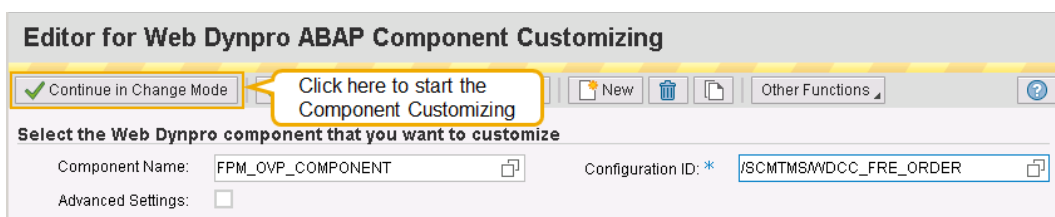
```
METHOD /bobf/if_frw_action~execute.
  DATA: lv_temp TYPE c.
  MESSAGE i008(/scmtms/ui_messages) INTO lv_temp.
  CALL METHOD /scmtms/cl_common_helper=>msg_helper_add_symsg
    EXPORTING
      iv_key      = /scmtms/if_tor_c=>sc_bo_key      " Instance Key
      iv_node_key = /scmtms/if_tor_c=>sc_node-root  " BO Key
  CHANGING
    co_message = eo_message. " Current message object
ENDMETHOD.
```

- 1) Start the UI for displaying Freight Orders from e.g. the SAP user menu, set the customizing mode in the URL for the UI as described in section 5.3 and display an example Freight Order.



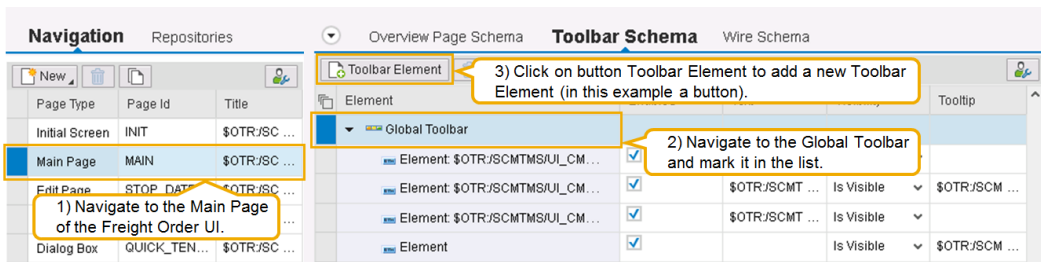
Picture: The Main Toolbar of the Freight Order Display UI with its buttons.

- 2) The component configuration `/SCMTMS/WDCC_FRE_ORDER` will be customized to add the new main toolbar action. If this component configuration has not yet been customized you first of all create a corresponding component customizing (see picture below) and then continue.



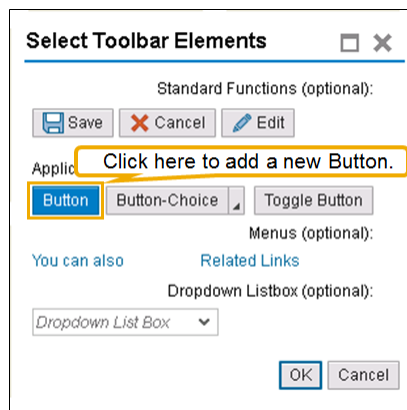
Picture: Component Customizing for Configuration `/SCMTMS/WDCC_FRE_ORDER`.

- 3) In the Configuration Editor first of all navigate to the Main Page and then on the right side of the Configuration Editor display the *Toolbar Schema*. In the table content of the Toolbar Schema you can find the *Global Toolbar* which contains the buttons and actions that you can see in the Main Toolbar of the Freight Order UI. Mark the *Global Toolbar* in the list.



Picture: Adding a button on the *Global Toolbar* of the *Main Page*.

- 4) Now click on button *Toolbar Element* to add a new toolbar element. The popup that is displayed allows you to add our new button by clicking on the application-specific Function *Button*. Then click on button *OK*.



Picture: Adding a new button.

Here you could also add standard functions like *Save*, *Cancel* and *Edit*, *Button Choices*, *Toggle Buttons*, *Dropdown List Boxes* and *Links*.

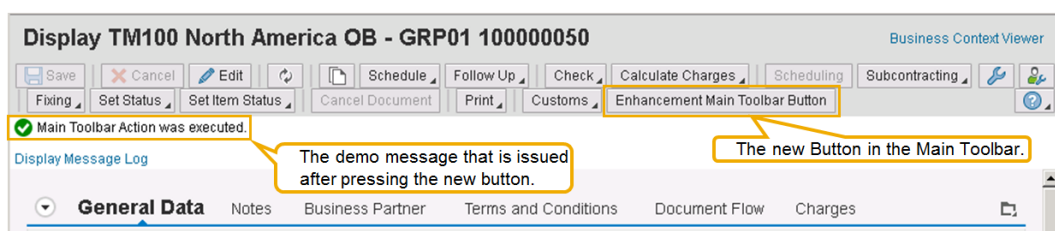
- 5) The new button will now be available in the *Toolbar Schema*. Mark the new entry in the list so that you can start specifying the attributes of the new button. Specify the following attribute values:

Attribute	Value
Text	Enhancement Main Toolbar Button
Tool Tip	An Enhancement Action on the main toolbar
FPM Event ID	ZENH_MAINTOOLBAR_ACTION
Action Type	Standard

Moreover maintain the following Event Parameter:

Parameter Name	Parameter Value
FBI_RAISED_BY_TOOLBAR	X

- 6) Save your configuration. The new button is now ready to be used on the Freight Order UI via its main tool bar.



Picture: The new button on the main tool bar.

With this first example the enhancement action will be triggered by the additional button on the Main Toolbar. As we have chosen the *FPM Event ID* to be identical with the action name, the execution of the action will be handled generically without any further coding required.

In case you don't want to relate the button to a BO action as shown in the first example, you can follow the second approach which allows you to implement arbitrary coding to be executed when clicking the related button on the UI. This works as follows:

For each application configuration (e.g. */SCMTMS/FRE\_ORDER*) there is a FBI View available that follows the naming convention *[application configuration name]\_HTLB*. For the example this is FBI View */SCMTMS/FRE\_ORDER-HTLB*. It is defined to handle the Toolbar of the application and the Exit Class defined there is called automatically.

Instead of providing an action name as the *FPM Event ID* (see step 4 above) you can provide an arbitrary FPM Event ID (e.g. *MyEventID*) that will be handled by the Exit Class of the HTLB FBI View. In the Exit Class you can add coding to method *ADAPT\_EVENT* to react on and handle the event. You can identify the corresponding Exit Class as follows:

- 1) Start the UI for displaying Freight Orders from e.g. the SAP user menu and display the technical help as described in section 5.3. On the technical help you can identify the application configuration, in our example */SCMTMS/FRE\_ORDER*.
- 2) Start transaction *SE84* and follow the path *Repository Information System → Web Dynpro → Component Configuration*. Enter Component Configuration name */SCMTMS/FRE\_ORDER-HTLB* in the input field *Component Configuration* on the right side.
- 3) Press *F8* to start the selection and then double click on the found entry. On the right side you can now see the general attributes of FBI View */SCMTMS/FRE\_ORDER-HTLB*. Click on button *Display Configuration* to display the details of the FBI View.
- 4) On tab strip *Header* you can see the name of the relevant exit class in field *Exit Interface Class*. For the example this is class */SCMTMS/CL\_UI\_VIEWEXIT-TOR*.

In the standard implementation of the identified Exit Class method *ADAPT\_EVENT* you can see how to react on your own *FPM Event IDs*. The *FPM Event ID* that was configured for the new button on the main tool bar will be available at runtime in method parameter *IV\_EVENTID*. The coding can then e.g. look as follows:

```
CASE iv_eventid.
  WHEN /scmtms/if_ui_cmn_c=>sc_action-cmn-show_plan_blkdet OR
        /scmtms/if_ui_cmn_c=>sc_action-cmn-show_exec_blkdet OR
        /scmtms/if_ui_cmn_c=>sc_action-cmn-show_inv_blkdet.
    handle_show_blkdet (
      EXPORTING
        iv_eventid          = iv_eventid
        ir_event_data       = ir_event_data
        it_selected_rows    = it_selected_rows
      CHANGING
        cv_failed          = cv_failed ).
*   React on your own Event ID here.
  WHEN MyEventId.
    " The coding that handles the event should be placed
    " in a separate method of e.g. a local class (in case
    " of customer extensions).
    CALL METHOD MyEventIdHandler().

  WHEN OTHERS.

ENDCASE.
```

### 5.4.5 Adding a new Parameter Action with a Popup

In the last section we added an action to the main tool bar of an FPM application. In the following example, a parameter action is added that invokes a popup to enter the parameters of an action before executing the action. Moreover the example indicates how a Confirmation Popup for actions can be realized by configuring an *OK* and a *Cancel* button.

Execute the following steps to create this example:

- 1) Start the BOBF Enhancement Workbench (transaction */BOBF/CUST\_UI*) and create the following enhancement action on the Root node of the Freight Order Business Object (technical name TOR. See also section 3.3.6 for creating actions.

Attribute	Value
Action Name	ZENH_POPUP_PARAM_ACT
Description	Enh. Parameter Action for UI Popup Demo
Implementing Class	ZCL_ENH_A_POPUP_PARAM_ACT
Parameter Structure	ZENH_S_A_POPUP_PARAM_ACT
Action Cardinality	Multiple Node Instances
Extensible	Yes

The example action parameter structure *ZENH\_S\_A\_POPUP\_PARAM\_ACT* shall look as follows:

Append Structure		
Name	ZENH_S_A_POPUP_PARAM_ACT	
Description	Enhancement Action Parameters	
Component	Typing Method	Component Type
ZENH_COMMENT	Types	/SCMTMS/DESCRIPTION
ZENH_WORKS_IND	Types	Boolean

Make sure that you have defined an enhancement category for the new structure. Then save and activate the action parameter structure.

The attributes of this structure will later be available to be placed on the popup. When executing the action via the UI, the popup will be displayed where you can enter corresponding values. In the example, we will configure an *OK* and a *Cancel* button that lets you execute or abort the execution of the action.

Use transaction *SE91* to create message class *ZENH\_MESS* with the following messages:

Message	Message Short Text
001	Yes, it works! &1
002	No, it doesn't work! &1

The coding for the action implementation shall look as follows (example code to be for method *EXECUTE* of implementing class *ZCL\_ENH\_A\_POPUP\_PARAM\_ACT*):

```
METHOD /bobf/if_frw_action~execute.
  FIELD-SYMBOLS: <fs_parameters> TYPE zenh_s_a_popup_param_act.

  DATA: ls_msg          TYPE symsg,
         lr_act_param    TYPE REF TO /scmtms/s_tor_a_conf,
         lv_temp         TYPE c.

  * take over action parameters
  ASSIGN is_parameters->* TO <fs_parameters>.
```

```

* prepare message text parameter
CLEAR ls_msg.
ls_msg-msgv1 = <fs_parameters>-zenh_comment.

* use parameter ZENH_WORKS_IND
IF <fs_parameters>-zenh_works_ind = abap_true.
  MESSAGE s001(zenh_mess) WITH ls_msg-msgv1 INTO lv_temp.
ELSE.
  MESSAGE e002(zenh_mess) WITH ls_msg-msgv1 INTO lv_temp.
ENDIF.

CALL METHOD /scmtms/cl_common_helper=>msg_helper_add_symmsg(
  EXPORTING
    iv_key          = /scmtms/if_tor_c=>sc_bo_key
    iv_node_key     = /scmtms/if_tor_c=>sc_node-root
  CHANGING
    co_message     = eo_message ).

ENDMETHOD.

```

- 2) Start the Web Dynpro ABAP Component Configuration Editor to create a new Component Configuration. As mentioned already in section 5.3, the editor can be started via the following general link which has to be enhanced with information on server and port, depending on where you want to start the tool.

[https://\[server\]:\[port\]/sap/bc/webdynpro/sap/configure\\_component](https://[server]:[port]/sap/bc/webdynpro/sap/configure_component)

Example:

[https://uscia9x.wdf.sap.corp:44352/sap/bc/webdynpro/sap/configure\\_component](https://uscia9x.wdf.sap.corp:44352/sap/bc/webdynpro/sap/configure_component)

Picture: Creating a new Component Configuration.

Enter a Component Name and a new Configuration ID. This new Component Configuration will define a From UIBB that will contain the action parameters as available input fields.

Field	Value	Comment
Component Name	FPM_FORM_UIBB	The new configuration shall represent a form which will contain data of the action parameter structure.
Configuration ID	ZENH_WDCC_POPUP_ACTION	This will be the new configuration to be integrated in the Freight Order UI.

Click on button *New* to create the new Component Configuration.

- 3) On the following popup provide a description and click on button *OK* to continue. On the next popup specify the package where you want to store the new Component Configuration and click on button *OK* again.

Field	Value
Description	Enhancement Action with Popup Demo
Package	\$TMP (or your own customer/partner specific package)

A third popup will come up requesting you to provide a transport request. Choose a valid existing transport request or create a new one (can be done on the same popup) and click on button **OK**.

- 4) Specify the FBI Feeder Class that shall be used for providing the data to the Form UIBB displayed on the new intended popup.

Picture: Entering the Feeder Class.

Choose Feeder Class `/BOFU/CL_FBI_GUIBB_ACTPRM_FDR` which represents a generic Action Parameter Feeder Class provided by the FBI framework. It makes use of the parameters defined for the action that will be assigned to trigger the new popup. The assignment of the action created in step 1 to the new Component Configuration is described in one of the next steps. Click on button *Edit Parameters* to get to the next step.

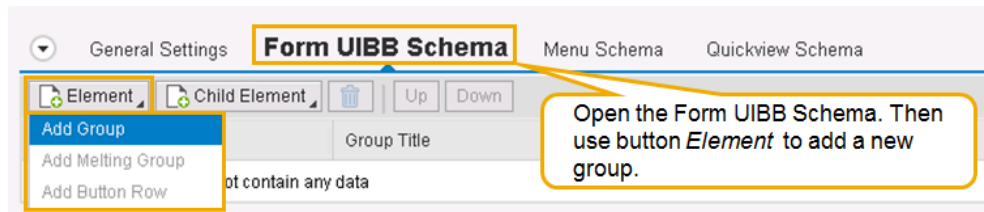
- 5) Now specify the parameters that the Feeder Class shall take into consideration at runtime. Enter the following values in the mentioned fields and then click on button **OK**.

Picture: Specifying the Feeder Class Parameters.

Field	Value	Comment
Business Object	/SCMTMS/TOR	The Business Object that the action is assigned to.
Node	ROOT	The node of the Business Object that the action is assigned to.
Action	ZENH_POPUP_PARAM_ACT	The name of the action that will be executed and delivers the fields to be provided on the popup based on its action parameter structure.

Action Parameter UI Structure	ZENH_S_A_POPUP_PARAM_ACT	The structure that serves as the UI structure for the action parameters (in this example it corresponds 1:1 to the Action Parameter Structure).
-------------------------------	--------------------------	---

- 6) In the Component Configuration Editor navigate to the Form UIBB Schema, click on button *Element* and choose option *Add Group* to add a new field group.

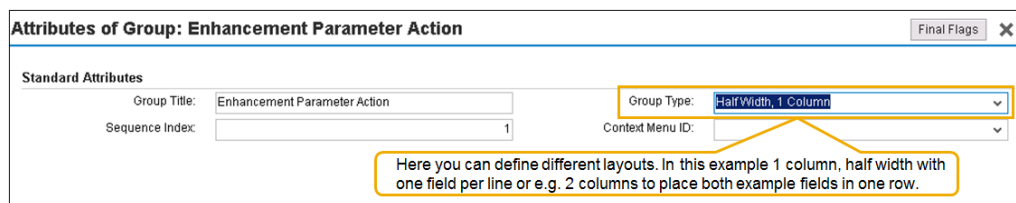


Picture: Adding a new Group in the Form UIBB Schema.

This group will contain the fields to be entered on the popup. In the attributes of the group enter the following value in field Group Title:

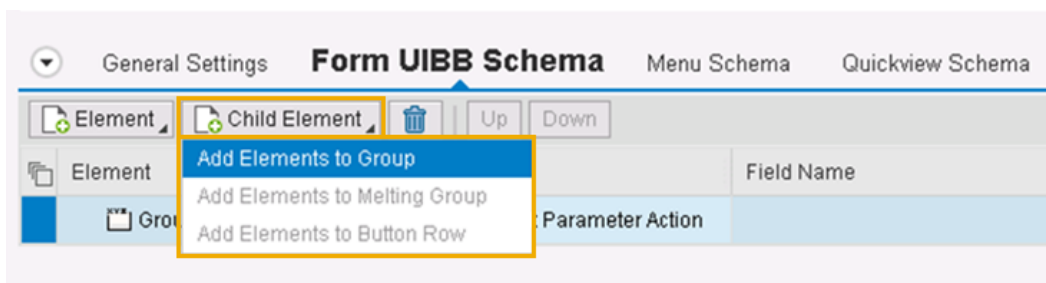
Field	Value
Group Title	Enhancement Parameter Action

Moreover you can specify further attributes for this group. You can give it a Group Title (e.g. Enhancement Parameter Action) and you can specify the Group Type which influences the layout of the two fields added to the group. In this example the option Half Width, 1 Column has been chosen which will arrange the two fields in a single column. Other options e.g. allow arranging the fields in two columns and in one row.



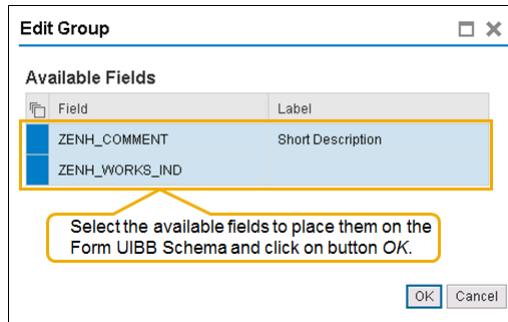
Picture: Specifying Attributes of the Group.

- 7) Select the new group and click on button *Child Element* in the Form UIBB Schema and add the available fields to the group. As we specified the Action Parameter UI Structure to be identical with the Action Parameter Structure, the available fields for the UI correspond exactly to the Action Parameters (if that was not the case you would have to specify an Action Parameter UI Mapper Class in the Feeder Class Parameters).



Picture: Adding new Group Elements.





Picture: Adding fields to the group.

Configure the following Attributes for the two added fields (this is the minimum set of attributes to be specified for this example but of course you can also specify further attributes like a Tooltip or specify a Search-Help for the corresponding field, etc.):

Field	Label	Display Type
ZENH_COMMENT	Comment	Input Field
ZENH_WORKS_IND	Flag whether it works or not	Check Box

- 8) Save the configuration for the new Form UIBB. With this step the definition of the popup to be displayed when executing the intended Action is finished. In the next steps, the popup as well as the Action are integrated into the Freight Order UI.
- 9) Start the editor for the Web Dynpro ABAP Component Configuration to add a new page to the Freight Order UI configuration `/SCMTMS/WDCC_FRE_ORDER`. As mentioned already in section 5.3, the editor can be started via the following general link which has to be enhanced with information on server and port, depending on where you want to start the tool. In this case we customize an already existing standard Component Configuration, i.e. to start the component customizing use the following kind of link (this time it's "customize\_component"):

[https://\[server\]:\[port\]/sap/bc/webdynpro/sap/customize\\_component](https://[server]:[port]/sap/bc/webdynpro/sap/customize_component)

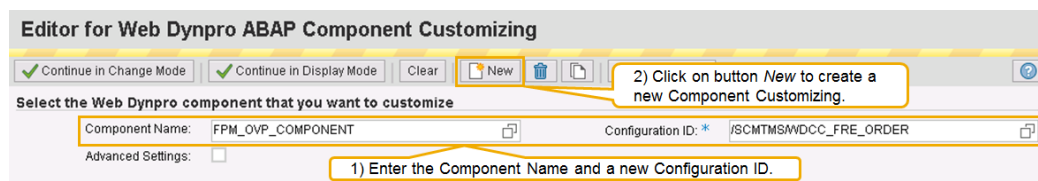
Example:

[https://uscia9x.wdf.sap.corp:44352/sap/bc/webdynpro/sap/customize\\_component](https://uscia9x.wdf.sap.corp:44352/sap/bc/webdynpro/sap/customize_component)

Here you can enter the Component Name and the Configuration ID of the Component Configuration to be customized. This new configuration will define a From UIBB that will contain the action parameters as available input fields.

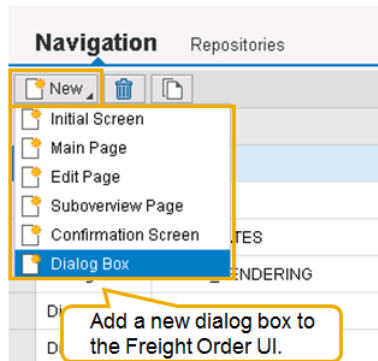
Field	Value	Comment
Component Name	FPM_OVP_COMPONENT	
Configuration ID	/SCMTMS/WDCC_FRE_ORDER	The standard Web Dynpro Component Configuration of the Freight Order UI.

Click on button *New* in case you have not yet created a corresponding component customizing for the given Component Configuration. If this already exists you can continue by clicking on button *Continue in Change Mode*.



Picture: Creating a new Component Customizing.

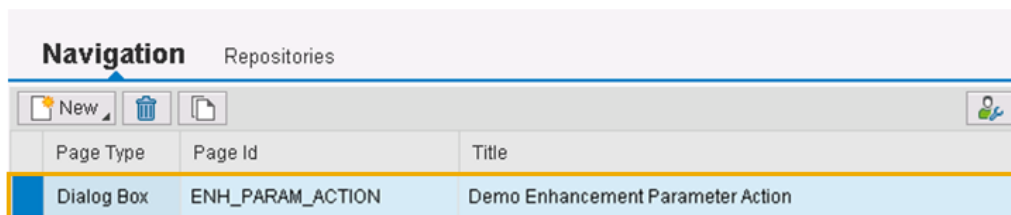
- 10) On the following screen go to section *Navigation* on the left side of the screen, click on button *New* and chose option *Dialog Box* to add a new Dialog Box to the Freight Order UI.



Picture: Adding a new Dialog Box.

Then double click on the new entry for the Dialog Box in the Navigation List and specify the required attributes directly in the related list entry. Note that the system will default a Page ID and a Title. For this example overwrite it with the following values:

Attribute	Value
Page ID	ENH_PARAM_ACTION
Title	Demo Enhancement Parameter Action



Picture: Specifying Dialog Box attributes in the Navigation List.

- 11) Now specify further required attributes for the Dialog Box on the right side of the Component Configuration Editor (if you do not directly see the Attributes Screen on the left side of the Editor double click again on the entry for the new Dialog Box in the Navigation List). This step is especially interesting as here the set of buttons to be available on the dialog popup for the Parameter Action is defined.

Picture: Specifying further attributes for the Dialog Box.

Attribute	Value
Dialog Name	Enhancement Parameter Action
Button Sets	OK and Cancel (OK is default button)
Tooltip for OK	Execute the BOPF Action assigned to this popup
Tooltip for Cancel	Cancel the Action and close the popup
Action Type Close	Validation-Independent

With the attribute *Button Sets* we have now specified what kind of buttons shall be available on the popup to trigger or cancel the related Action after having entered values in the input fields of the popup. In this example an *OK* and a *Cancel* button will be used where the *OK* button is the default button (i.e. it will have the focus when the popup is displayed).

- 12) Navigate to the Preview of the newly added Dialog Box on the right side of the Component Configuration Editor. Here you should see that the system has added a section for the Dialog Box with the content below this section still empty. Double click on the section content under the section. In the lower right part of the Editor you can now specify required attributes that specify the content to be displayed on the Dialog Box, i.e. the Form UIBB with the two input fields created in the previous steps will now be assigned to the Dialog Box.

Attribute	Value
Component	FPM_FORM_UIBB
Window Name	FORM_WINDOW
Configuration ID	ZENH_WDCC_POPUP_ACTION
Rendering Type	Without Panel
Title	Enhancement Parameter Action

The screenshot shows the SAP Component Configuration Editor. The top part is the 'Preview' section, which displays a dialog box titled 'SECTION\_1'. Below the title, there is a section for 'Form UIBB' with the window name 'FORM\_WINDOW' and configuration name 'ZENH\_WDCC\_POPUP\_ACTION'. The bottom part of the screenshot shows the 'Attributes of UIBB: Enhancement Parameter Action' section. This section contains two columns of attributes. The left column lists 'Standard Attributes' such as Component, Window Name, Config ID, Instance ID, Column, Sequence Index, Container Stretching, and Hidden Element. The right column lists 'Rendering Type', Title, Explanation Text, Explanation Document, Default Edit Page, and Default Details Page. The 'Component' attribute is set to 'FPM\_FORM\_UIBB', 'Window Name' to 'FORM\_WINDOW', 'Config ID' to 'ZENH\_WDCC\_POPUP\_ACTION', 'Rendering Type' to 'Without Panel', and 'Title' to 'Enhancement Parameter Action'.

Picture: Specifying the attributes for assigning the Form UIBB to the Dialog Box.

- 13) Save the Component Customizing. With this step the popup for the Action is assigned to the Freight Order UI. Before it can actually be used, a few more things need to be configured as shown in the next steps.
- 14) For the next step start the Component Configuration Editor to customize (remember “customize\_component” in the URL for starting the editor) the Application Controller Configuration /SCMTMS/WDCC\_APPCC. Here the BOPF Action implemented in the first steps is finally assigned to the popup that was configured in the previous steps.

**Editor for Web Dynpro ABAP Component Customizing**

Continue in Change Mode Continue in Display Mode Clear New

Select the Web Dynpro component that you want to customize

Component Name: /BOFU/WDC\_FBI\_CONTROLLER Configuration ID: \* /SCMTMS/WDCC\_APPCC

Advanced Settings: ☐

1) Enter the Component Name and a new Configuration ID.

2) Click on button New to create a new Component Customizing.

Picture: Creating a new Component Customizing for the Application Controller.

Field	Value	Comment
Component Name	/BOFU/WDC_FBI_CONTROLLER	
Configuration ID	/SCMTMS/WDCC_FRE_ORDER	The standard SAP TM Application Controller Configuration.

Click on button *New* to create a new Component Customizing for the Application Controller or continue with clicking on button *Continue in Change Mode* in case a corresponding Component Customizing already exists.

- 15) Open section *Component-Defined*. In its subsection *Configuration Context* mark the Tree List entry *context* and click the right mouse button to open the context menu for this entry. Choose *Add → actParamConfig* to add a new Action Parameter Configuration in the Application Controller.

**Component Customizing /SCMTMS/WDCC\_APPCC**

Save Cancel Edit Check New Window Enhance Properties

**Component-Defined**

**Configuration Context**

New

Elements

context Add actParamConfig

actPar Remove

actPar Display Quick Help

actPar More Field Help...

actPar Technical Help...

actParamConfig (5)

actParamConfig (7)

actParamConfig (8)

actParamConfig (9)

**Attributes of the Element actParamConfig (28)**

cfgIndex: \* 28 Final

Business Object: /SCMTMS/TOR Final

Node: ROOT Final

Action: ZENH\_POPUP\_PA Final

Dialog Box ID: ENH\_PARAM\_ACTION Final

Make sure to enter a valid number representing the last cfgindex in the list.

Specify the Action by providing the technical BO name, the node that it is assigned to and the Action Name as well as the Dialog Box ID.

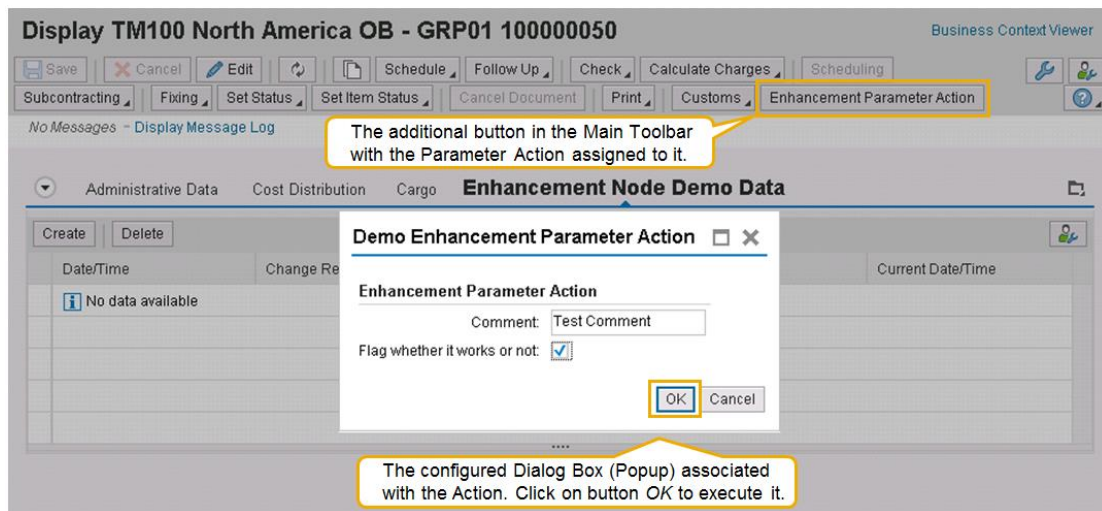
Picture: Adding a new Action Parameter Configuration in the Application Controller.

Specify the following attributes:


Attribute	Value
cfgIndex	A valid number representing the latest entry in the list (Final flag is set automatically by the system).
Business Object	/SCMTMS/TOR (Final flag not set)
Node	ROOT (Final flag not set)
Action	ZENH_POPUP_PARAM_ACT (Final flag not set)
Dialog Box ID	ENH_PARAM_ACTION (Final flag not set)

- 16) Click on button *Save* in the Component Customizing Tool bar to save the Application Controller customizing.
- 17) In the final step add a button to the main toolbar of the Freight Order UI as shown in the previous section 5.4.4. Assign the parameter action *ZENH\_POPUP\_PARAM\_ACT* to it that we created in the section.

The resulting popup that comes up when executing the new action can be seen on the following picture. The popup comes up after clicking on the button added with step 17. You can then enter values in the input fields and click on button *OK* to start the execution of the action with the entered values or click on button *Cancel* to abort the execution.



Picture: The final Parameter Action Popup in an example Freight Order.

With the input in the example shown in the picture above, the message that is issued looks as follows:  **Yes it works! Test Comment**

#### 5.4.6 Accessing and displaying data from external sources

The following use case is based on a real world scenario where a customer wanted to display external data on the Freight Order UI, i.e. the data to be displayed does comes from a data source outside the TM application / system.

The Use Case looks as follows: On the *Document Reference* tab of the Freight Order UI shipment numbers (originally coming from ERP) associated with the displayed Freight Order are shown. For these shipments some further details shall be read from ERP and displayed along with the Freight Order information. In the specific customer example the serial numbers for the products assigned to the shipments shall be displayed. The serial numbers are not known on TM side but need to be read from the shipments in ERP.

1) To simplify this example a bit and to rather “simulate” such an external data source we simply create the following database table that serves as our external data source and can be populated with example data: Start transaction *SE11* and create data base table *ZENH\_D\_SERNUM* as follows:

1. On the initial screen of transaction *SE11* enter the table name *ZENH\_D\_SERNUM* in field Database Table and click on button *Create*.
2. On the next screen enter the following short description for the new table: *Database Table for simulating access to external data*.
3. On tab strip *Delivery and Maintenance* specify *Delivery Class A (Application Table – master and transactional data)* and specify *Display Maintenance Allowed* in field *Data Browser / Table View Maintenance*.
4. On tab strip *Fields* enter the following fields that make up our “external data source”:

Field	Key	Initial Values	Data Type	Short Description
MANDT	Yes	Yes	MANDT	Client
BTD_ID	Yes	Yes	/SCMTMS/BTD_ID	Bus. Trans. Document ID
SERNUM_IDX	Yes	Yes	/SCMTMS/INTEGER_VALUE	Integer Value
SERNUM			/SCMTMS/STRING	String
INUSE			BOOLEAN	Boolean Variable (X = True, - = False, Space = Unknown)
WEIGHT			/SCMTMS/QUANTITY	Quantity
WEIGHT_UOM			/SCMTMS/DIM_WEIGHT_WT_UOM	Weight Unit of Measure for Dimensional Weight

5. On tab strip *Currency / Quantity Fields* specify for field *WEIGHT* the reference table *ZENH\_D\_SERNUM*, reference field *WEIGHT\_UOM*.
6. Maintain enhancement category *Can Be Enhanced (deep)* under *Extras* → *Enhancement Category*.
7. Click on button *Technical Settings* and maintain the technical settings of the new database table as follows:
  - Data class : APPL1
  - Size Category : 4
  - Buffering not allowed : Yes
8. Finally save and activate the new database table.

The following steps illustrate how to realize an additional tab strip on the standard Freight Order UI that contains a list with the shipments and related serial numbers.

2) Display FBI View */SCMTMS/TOR\_DOCREF* as follows:

- Start transaction *SE84* and choose Web Dynpro.
- Double click on Component Configuration.
- Enter */SCMTMS/TOR\_DOCREF* in field *Component Configuration* and press *F8*.
- Double click on the found entry on the next screen and here click on button *Start Configurator*.
- Click on button *Continue in Display Mode*.

3) On the FBI View you can find the defined UI Structure, Mapper Class and Exit Interface class used for the tab strip *Document Reference* tab of the Freight Order UI. Create a copy of all three objects as follows:

- Create a copy of Mapper Class */SCMTMS/CL\_UI\_CONVERSION\_TOR* with transaction *SE24* and name it *ZCL\_ENH\_UI\_CONVERSION\_SERNUM*. Save and activate the copy.
- Create a copy of Exit Interface Class */SCMTMS/CL\_UI\_VIEWEXIT\_TOR* with transaction *SE24* and name it *ZCL\_ENH\_UI\_VIEWEXIT\_SERNUM*. Save and activate the copy.



- Create a copy of Node UI structure `/SCMTMS/S_UI_CMN_DOCREF` with transaction `SE11` and name it `ZENH_S_UI_SERNUM`. Save and activate the copy
- 4) Add all external fields to the Node UI structure `ZENH_S_UI_SERNUM` that shall be read and displayed from the external data source, i.e. in the example this would be all fields that we have already used to define the database table in step 1, except the fields `MANDT` and `BTD_ID`.
- 5) Create a new FBI View `ZENH_SERNUM`: Display FBI View `/SCMTMS/TOR_DOCREF` as described in step 2.
- On the first screen make sure that you specify the Component Name as `/BOFU/FBI_VIEW` (already specified after displaying the mentioned standard FBI View). Clear field `Configuration ID` and enter `ZENH_SERNUM` as the name for the new FBI View.

Picture: Creating a new FBI View.

- Click on button `New` to start creating the new FBI View. On the Header tab of the following screen enter the following data:

Field	Value
Business Object	/SCMTMS/TOR
Node	DOCREference
Node UI Structure	ZENH_S_UI_SERNUM
Mapper Class	ZCL_ENH_UI_CONVERSION_SERNUM
Exit Interface Class	ZCL_ENH_UI_VIEWEXIT_SERNUM

Picture: Specifying the required header properties of the FBI View.

- Click on button `Save`.
- 6) Section 5.4.3 described already how to add a new List UIBB to the Freight Order UI. We now create another List UIBB that will carry the external data. Start the Component Configurator and create a corresponding Component Configuration for the List UIBB:

Picture: Creating the Component Configuration for the List UIBB.



Field	Value
Component Name	FPM_LIST_UIBB_ATS
Configuration ID	ZENH_WDCC_SERNUM

- Use Feeder Class `/BOFU/CL_FBI_GUIBB_LIST_ATS` for this Component Configuration. In the Feeder Class Parameters assign FBI View `ZENH_SERNUM` that we have created in step 5 before.

**Edit Parameters**

Enter valid feeder class parameters

**Feeder Class**

Feeder Class: `/BOFU/CL_FBI_GUIBB_LIST_ATS` FBI Feeder class for ATS List GUIBB

**Parameters**

FBI View: `ZENH_SERNUM`

Business Object:

Node:

Picture: Assigning Feeder Class and FBI View.

- Configure the columns of the List UIBB. The fields of the new Node UI Structure `ZENH_S_UI_SERNUM` are all available for representing columns in the new list. Configure them all correspondingly and save the List UIBB configuration.

**Preview**

Document T...	Document ID	Document T...	Document...	Item	Item Type	Item Type D...	Issuer	Is used	Serial Num...	Serial Num...	Weight	Wei...
											0	0,000000000...
											0	0,000000000...
											0	0,000000000...
											0	0,000000000...
											0	0,000000000...
											0	0,000000000...

Picture: The configured List UIBB in the preview.

- Save the Component Configuration for the new List UIBB.
- 7) Start the Configuration Editor in customizing mode for the standard Component Configuration `/SCMTMS/WDCC_FRE_ORDER` and add the new List UIBB. First of all add the new List UIBB on the Overview Page Schema as shown in the following picture.

**Overview Page Schema**

Page Master Area | Section | UIBB | Up | Down

1) Add a new List Component.

2) Specify the required attributes.

Config ID	Window Name	Container Stretching
/SCMTMS/WDCC_CMN_ADMINDATA	FORM_WINDOW	UIBB Does Not Need Surroundi...
/SCMTMS/WDCC_MDC_VB	MAIN	UIBB Needs Surrounding Conta...
/SCMTMS/WDCC_TOR_ROOTDOCRE	LIST_WINDOW	UIBB Does Not Need Surroundi...
/SCMTMS/WDCC_TCD_TOP	COMPOSITE_WINDOW	UIBB Does Not Need Surroundi...
/SCMTMS/WDCC_FRE_ORDER_CAR	COMPOSITE_WINDOW	UIBB Does Not Need Surroundi...
/SCMTMS/WDCC_CBAIR_GEN_H	LIST_WINDOW	UIBB Does Not Need Surroundi...
ZENH_WDCC_DEMO_LIST_UIBB	LIST_WINDOW	UIBB Does Not Need Surroundi...
ZENH_WDCC_SERNUM	LIST_WINDOW	UIBB Does Not Need Surroundi...

Picture: Adding the new List UIBB (as customizing) to the standard UI.

Specify the following attributes for the added List UIBB:

Field	Value
Component	FPM_LIST_UIBB_ATS
Window Name	LIST_WINDOW
Configuration ID	ZENH_WDCC_SERNUM
Title	Demo Enh. List with access to external data

**Attributes of UIBB: Demo Enh. List with access to external data** Final Flags ✕

**Standard Attributes**

Component: *	FPM_LIST_UIBB_ATS	Rendering Type:	With Panel
Window Name: *	LIST_WINDOW	Collapsed:	<input type="checkbox"/>
Config ID:	ZENH_WDCC_SERNUM	Title:	Demo Enh. List with access to external data
Instance ID:		Explanation Text:	
Column:	1	Explanation Document:	
Row Number:	1	Default:	<input type="checkbox"/>
Sequence Index:	25	Default Edit Page:	
Container Stretching:	UIBB Does Not Need Surrounding Containers to be Stretc	Default Details Page:	
Hidden Element:	Visible	Padding:	Automatic (Default)

Picture: The attributes for the new List UIBB.

- 8) On Wire Schema add a new Wire to connect the new List UIBB with the standard UI. Specify the following attributes for the Wire:

Field	Value
Component	FPM_LIST_UIBB_ATS
Configuration Name	ZENH_WDCC_SERNUM
Source Component	FPM_FORM_UIBB
Source Configuration Name	/SCMTMS/WDCC_TOR_INIT_SCREEN
Port Type	Collection
Port Identifier	CO
Connector Class	/BOFU/CL_FBI_CONNECTOR
Source Node Association	DOCREference

**Attributes of Wire: List ZENH\_WDCC\_SERNUM** Final Flags ✕

**Standard Attributes**

Component: *	FPM_LIST_UIBB_ATS	Src Inst. ID:	
Config ID:	ZENH_WDCC_SERNUM	Port Type:	Collection
Instance ID:		Port Identifier:	CO
Source Component:	FPM_FORM_UIBB	Connector Class: *	/BOFU/CL_FBI_CONNECTOR
Source Config Name:	/SCMTMSWDCC_TOR_INIT_SCREEN		

**Connector Parameters**

SRC_NODE_ASSOC:	DOCREference
-----------------	--------------

Picture: The attributes for the required Wire.

- 9) Save the customizing for Component Configuration /SCMTMS/WDCC\_FRE\_ORDER. With this step we have now added everything in the Freight Order UI to display the external data. Now a bit coding is required to read, prepare and get the external data displayed correctly on the new List UIBB.
- 10) In method *BUILD\_MAP\_TABLE* of class *ZCL\_ENH\_UI\_CONVERSION\_SERNUM* add the following lines of code at the beginning of the method implementation. The member variable *MV\_CALL\_EXIT* is set to true. At runtime this setting will force the application to call method *CALL\_EXIT\_METHOD*.

```
METHOD build_map_table.
```

```
DATA:
```

```
ls_map_data_ext TYPE ts_map_data_ext.
```

```
SET EXTENDED CHECK OFF.
```

```
* call exit method where the extraction of the external data
```

```

* will take place.
mv_call_exit = abap_true.

CASE iv_ui_struct.
...
ENDCASE.

ENDMETHOD.

```

- 11) In method `CALL_EXIT_METHOD` of class `ZCL_ENH_UI_CONVERSION_SERNUM` add the following lines of code at the beginning of the `CASE` statement:

```

METHOD call_exit_meth.

CASE mv_ui_struct.

    WHEN 'ZENH_S_UI_SERNUM'.
        CALL METHOD zcl_enh_sernum_ext_access=>read_serial_numbers
            CHANGING
                ct_ui_data = ct_ui_data.

    WHEN '/SCMTMS/S_UI_TOR_ITEM' OR
    ...
ENDCASE.

ENDMETHOD.

```

At runtime when the new UI structure `ZENH_S_UI_SERNUM` is used, a method of an external class is called that contains the coding to read data from the external data source.

When the method is called, the changing parameter `CT_UI_DATA` contains first of all the data of the Document Reference Tab of the Freight Order UI (remember that we have assigned `/SCMTMS/TOR` as the source BO and `DOCREERENCE` as the source node in the involved FBI View `ZENH_SERNUM`).

- 12) Create a simple static class `ZCL_ENH_SERNUM_EXT_ACCESS` via transaction `SE24` with a method `READ_SERIAL_NUMBERS`. The method shall have a changing parameter `CT_UI_DATA` of type `ANY_TABLE`. Within this method, the data from the external data source is read and used to rebuild and redefine the content of the initial data in `CT_UI_DATA` that will then be displayed in the new List UIBB that was added. The example coding for method `READ_SERIAL_NUMBERS` looks as follows:

```

METHOD read_serial_numbers.
* declarations
TYPES: BEGIN OF ls_btd_id_range,
        sign TYPE ddsign,
        option TYPE ddoption,
        low TYPE /scmtms/btd_id,
        high TYPE /scmtms/btd_id.
TYPES: END OF ls_btd_id_range.
TYPES: lt_btd_id_range TYPE TABLE OF ls_btd_id_range.

DATA: lv_component_bo TYPE string,
      lv_component_ui TYPE string,
      ls_sernum TYPE zenh_s_sernum,
      lt_sernum TYPE TABLE OF zenh_s_sernum,
      ls_btd_id TYPE ls_btd_id_range,
      lt_btd_id TYPE lt_btd_id_range,
      ls_sernum_ui TYPE zenh_s_ui_sernum,
      lt_sernum_ui TYPE TABLE OF zenh_s_ui_sernum,

```

```

ls_sernum_data                                TYPE REF TO data.

FIELD-SYMBOLS:
  <fs_ui_data>                                TYPE any,
  <fs_btd_tco>                                TYPE /scmtms/btd_type_code,
  <fs_btd_id>                                  TYPE /scmtms/btd_id,
  <fs_btd_date>                                TYPE /scmtms/btd_date,
  <fs_btd_issuer>                              TYPE /scmtms/btd_issuingparty_name,
  <fs_btditem_tco>                              TYPE /scmtms/btd_item_typecode,
  <fs_btditem_id>                              TYPE /scmtms/btd_item_id,
  <fs_btd_tco_txt>                              TYPE /scmtms/description_s,
  <fs_btditem_tco_txt>                          TYPE /scmtms/description_s,
  <fs_sernum>                                  TYPE zenh_s_sernum,
  <fs_sernum_ui>                              TYPE zenh_s_ui_sernum,
  <fs_key>                                      TYPE /bobf/conf_key.

CLEAR: ls_btd_id,
      lt_btd_id.

* collect all relevant BTD IDs to be used for
* accessing external data
LOOP AT ct_ui_data ASSIGNING <fs_ui_data>.
  ASSIGN COMPONENT 'BTD_TCO' OF STRUCTURE <fs_ui_data>
    TO <fs_btd_tco>.
  IF <fs_btd_tco> = 'I001'.
    ASSIGN COMPONENT 'BTD_ID' OF STRUCTURE <fs_ui_data>
      TO <fs_btd_id>.
    ls_btd_id-option = 'EQ'.
    ls_btd_id-sign    = 'I'.
    ls_btd_id-low     = <fs_btd_id>.
    APPEND ls_btd_id TO lt_btd_id.
  ENDIF.
ENDLOOP.

* read the data from the external source
* Example: A Select statement to access our "Simulation Table"
SELECT btd_id
      sernum_idx
      sernum
      inuse
      weight
      weight_uom
FROM zenh_d_sernum
INTO CORRESPONDING FIELDS OF TABLE lt_sernum
WHERE btd_id IN lt_btd_id.

* take over the data into the TM UI structure
IF sy-subrc = 0.
  CLEAR: ls_sernum_ui,
        lt_sernum_ui.

* create a table with the complete UI information, including the
* keys of the original entries from DOCREF.
LOOP AT ct_ui_data ASSIGNING <fs_ui_data>.
  ASSIGN COMPONENT 'KEY' OF STRUCTURE <fs_ui_data>
    TO <fs_key>.
  ASSIGN COMPONENT 'BTD_ID' OF STRUCTURE <fs_ui_data>
    TO <fs_btd_id>.
  ASSIGN COMPONENT 'BTD_TCO' OF STRUCTURE <fs_ui_data>
    TO <fs_btd_tco>.
  ASSIGN COMPONENT 'BTD_DATE' OF STRUCTURE <fs_ui_data>

```

```

        TO <fs_btd_date>.
    ASSIGN COMPONENT 'BTD_ISSUER' OF STRUCTURE <fs_ui_data>
        TO <fs_btd_issuer>.
    ASSIGN COMPONENT 'BTDITEM_TCO' OF STRUCTURE <fs_ui_data>
        TO <fs_btditem_tco>.
    ASSIGN COMPONENT 'BTDITEM_ID' OF STRUCTURE <fs_ui_data>
        TO <fs_btditem_id>.
    ASSIGN COMPONENT 'BTD_TCO_TXT' OF STRUCTURE <fs_ui_data>
        TO <fs_btd_tco_txt>.
    ASSIGN COMPONENT 'BTDITEM_TCO_TXT' OF STRUCTURE <fs_ui_data>
        TO <fs_btditem_tco_txt>.
    LOOP AT lt_sernum ASSIGNING <fs_sernum>
        WHERE btd_id = <fs_btd_id>.
        ls_sernum_ui-key = <fs_key>.
        ls_sernum_ui-btd_id = <fs_btd_id>.
        ls_sernum_ui-btd_tco = <fs_btd_tco>.
        ls_sernum_ui-btd_date = <fs_btd_date>.
        ls_sernum_ui-btd_issuer = <fs_btd_issuer>.
        ls_sernum_ui-btditem_tco = <fs_btditem_tco>.
        ls_sernum_ui-btditem_id = <fs_btditem_id>.
        ls_sernum_ui-btd_tco_txt = <fs_btd_tco_txt>.
        ls_sernum_ui-btd_tco_txt = <fs_btd_tco_txt>.
        ls_sernum_ui-sernum_idx = <fs_sernum>-sernum_idx.
        ls_sernum_ui-sernum = <fs_sernum>-sernum.
        ls_sernum_ui-inuse = <fs_sernum>-inuse.
        ls_sernum_ui-weight = <fs_sernum>-weight.
        ls_sernum_ui-weight_uom = <fs_sernum>-weight_uom.
        APPEND ls_sernum_ui TO lt_sernum_ui.
    ENLOOP.
ENDLOOP.

*   transfer the UI data to be passed to the List
*   UIBB into CT_UI_DATA
IF NOT lt_sernum IS INITIAL.
    CLEAR ct_ui_data.
    CREATE DATA ls_sernum_data TYPE zenh_s_ui_sernum.
    ASSIGN ls_sernum_data->* TO <fs_ui_data>.
    LOOP AT lt_sernum_ui ASSIGNING <fs_sernum_ui>.
        MOVE-CORRESPONDING <fs_sernum_ui> TO <fs_ui_data>.
        INSERT <fs_ui_data> INTO TABLE ct_ui_data.
    ENLOOP.
ENDIF.

ENDIF.

ENDMETHOD.

```

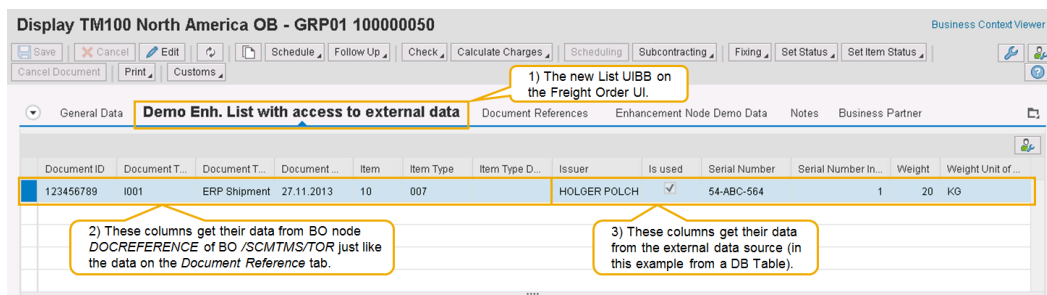
The coding does the following at runtime. From the UI data in *CT\_UI\_DATA* (which first of all just contains the same data as available on the Document Reference tab) those entries are filtered out for which serial numbers shall be read from the external data source. In the example, this shall e.g. only happen for those document references that represent a Shipment, i.e. *BTD\_TCO* = *1001* (this depends of course on the customizing for Business Transaction Document Type codes).

For the relevant documents a *SELECT* statement reads the serial number from the external data source which is in the example case the database table that we created in step 1. The result of this *SELECT* statement will then be used to rebuild the content of changing parameter *CT\_UI\_DATA* to finally contain the list of relevant documents with their related serial numbers. This modified data will then be provided to the new List UIBB that was added to contain the serial number information. So the new List UIBB combines

now a set of attributes that are filled from standard BO node DOCKERREFERENCE and another set of attributes that get their content from the external data source.

Note: If data from external data sources shall be displayed within the TM UI please always make sure that the access to this external data source is implemented with the highest performance possible. In real implementations of such a use case you should also consider a buffer mechanism for the external data that only reads data again from the external data source if really necessary.

- 13) For testing the example create a Freight Order (or use an existing one) and first of all enter a list entry on tab *Document Reference* (you may have to make this tab visible first of all via the personalization of the Freight Order UI). Then create an entry in Database Table *ZENH\_D\_SERNUM* that was created in step 1 with attribute *BTD\_ID* filled with the same Document ID as for the entry on tab *Document Reference* (this is the link between the BO node entry and the external data source entry). An example of the final new List UIBB on the Freight Order UI then looks as follows:



Picture: The new List UIBB with example data from the external data source.

### 5.4.7 Building a simple new User Interface

As SAP Transportation Management uses the Floor Plan Manager for Web Dynpro ABAP building a user interface is based on a standard technology which is available in SAP NetWeaver. For details on FPM and how to build user interfaces with it, please also refer to the corresponding SAP NetWeaver documentation under the following link:

[http://help.sap.com/saphelp\\_nw73/helpdata/en/ic/182711c34a4684a7c0214b42554514/frameset.htm](http://help.sap.com/saphelp_nw73/helpdata/en/ic/182711c34a4684a7c0214b42554514/frameset.htm)

Further information can also be found in the SAP Community Network under the following link (search there for FPM): <http://scn.sap.com/welcome>

To connect *FPM* build user interfaces with an application, so called Feeder Classes are used. Their implementation is based on a predefined interface definition providing all necessary methods and corresponding signatures in order to standardize the communication between the application and the GUIBBs. Some examples:

Interface	Comment
IF_FPM_GUIBB_FORM	Interface for FORM components.
IF_FPM_GUIBB_LIST	Interface for LIST components
IF_FPM_GUIBB_SEARCH	Interface for SEARCH components
IF_FPM_GUIBB_TREE	Interface for TREE components

Method *GET\_DEFINITION* of a Feeder Class defines the field catalog of the component (*GUIBB*). At runtime, method *GET\_DATA* supplies the component with data from the application.

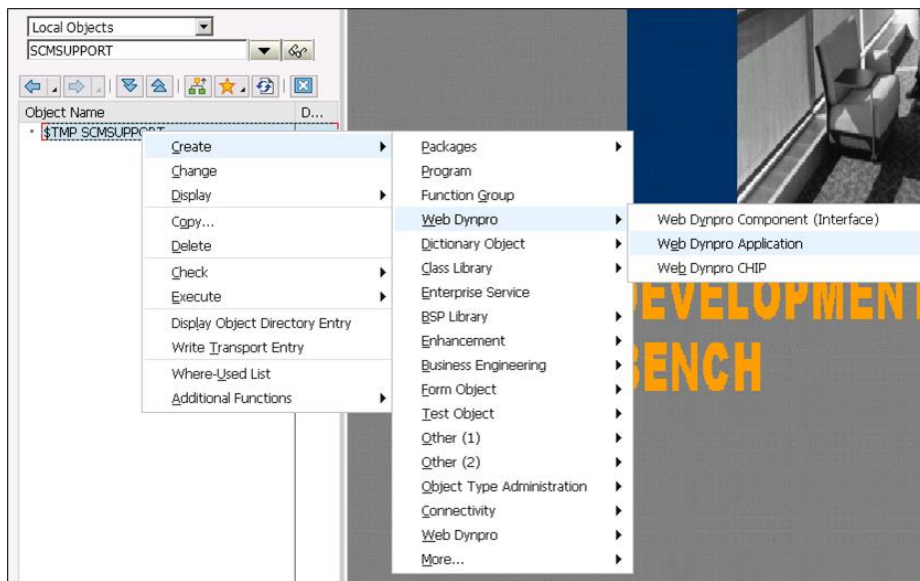
SAP Transportation Management uses FBI (Floor Plan Manager BOPF Integration) to connect the FPM user interface with the application which is based on BOPF business objects and related business logic. Instead of having to implement individual Feeder Classes,

FBI provides a list of already implemented (generic) Feeder Classes that can be reused to connect a FPM user interface with the BOPF based application.

These available generic FBI Feeder Classes allow creating a new UI on an existing BOPF business object without having to implement any own coding (of course there might be UIs with a higher complexity which require coding e.g. in an Exit Class of an involved FBI View).

The following example shows how to build a small and simple UI on top of the Freight Order BO (/SCMTMS/TOR) from scratch. Don't be scared about the number of steps. The intention is to show a variety of different configuration aspects for realizing even specific and useful details. This will help to understand the configuration possibilities from scratch.

- 1) Start transaction *SE80*, navigate e.g. to your local objects (we will store the example in the local package *\$TMP*) and create a new Web Dynpro Application with the following parameters:

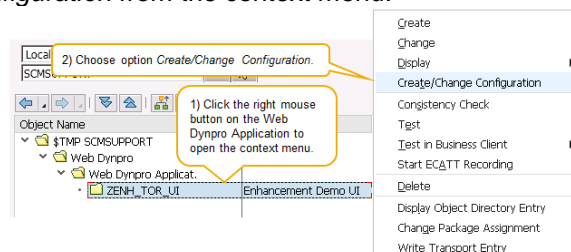


Picture: Creating a new Web Dynpro Application in *SE80*.

Field	Value
Application	ZENH_TOR_UI
Description	Enhancement Demo UI
Component	FPM_OVP_COMPONENT
Interface View	FPM_WINDOW
Plug Name	DEFAULT

Save the new Web Dynpro Application and assign it to your preferred package (in this example we simply use package *\$TMP*).

- 2) Create an Application Configuration for the new Web Dynpro Application as follows. Click the right mouse button on the Web Dynpro Application in *SE80* and choose option *Create/Change Configuration* from the context menu.



Picture: Creating an Application Configuration.



The Configuration Editor will come up in a browser window. Specify the following Application Name and Configuration ID and then click on button **New**:

Picture: Creating the Application Configuration.

Field	Value
Application Name	ZENH_TOR_UI
Configuration ID	ZENH_TOR_UI

On the following popups specify a description and a package where to store the Application Configuration, e.g. package \$TMP. In case you want to transport this demo UI, assign it to a package that will request you with each relevant step to specify a transport request.

Field	Value
Description	Enhancement Demo UI for TOR BO
Package	\$TMP (or your own package)

- On the following screen open section *Assign Web Dynpro Component*. Here you can find an entry for your Application Configuration. Mark it and click on button **Assign Configuration Name** to specify a corresponding Configuration ID on the following popup. With this step we assign a Web Dynpro Component Configuration (WDCC) to the Application Configuration that will carry all the content of the new UI. Enter the following Configuration ID:

Picture: Assigning and creating a Configuration for the Application.

Field	Value
Component Usage	ZENH_TOR_UI (defaulted)
Component	FPM_OVP_COMPONENT (defaulted)
Implementation	FPM_OVP_COMPONENT (defaulted)
Configuration	ZENH_WDCC_TOR_UI

The Configuration Name *ZENH\_WDCC\_TOR\_UI* is now assigned to the Application Configuration but does not yet exist. Click on the Configuration Name in the related column to create it (see also picture above). The Configuration Editor will come up in a browser window. Enter the following data and click on button **New** to create the Component Configuration:

Field	Value
Component Name	FPM_OVP_COMPONENT
Configuration ID	ZENH_WDCC_TOR_UI

Picture: Creating the Component Configuration.

As always, on the following popups specify a description and a package where to store the Component Configuration (which represents the Application Configuration).

Field	Value
Description	Enhancement Demo UI Application Configuration
Package	\$TMP (or your own package)

- 4) In the Component Configuration Editor open section *General Settings*. Click on button *Floorplan Settings* and select option *Application Controller Settings*. On the following popup enter the following settings and click on button *Ok*:

Picture: Specifying the Application Controller Settings.

Field	Value
Web Dynpro Component	/BOFU/WDC_FBI_CONTROLLER
Configuration Name	/SCMTMS/WDCC_APPCC

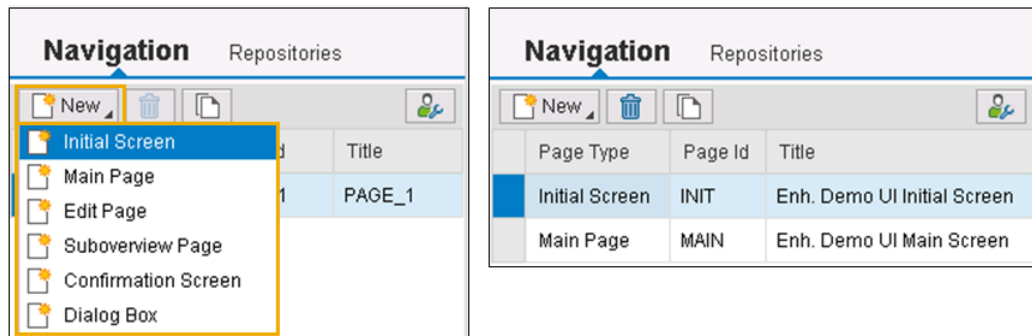
Picture: Application-Specific Parameters.

- 5) On the right side of the Component Configuration Editor navigate to section *Navigation*. In the displayed list, the system will already default an entry for a Main Page. Click on button *New*, choose option *Initial Screen* and specify the following parameters for this new page:

Field	Value
Page ID	INIT
Page Type	Initial Page
Title	Enh. Demo UI Initial Screen

For the Main Page entry in the list specify the following parameters:

Field	Value
Page ID	MAIN
Page Type	Main Page
Title	Enh. Demo UI Main Screen

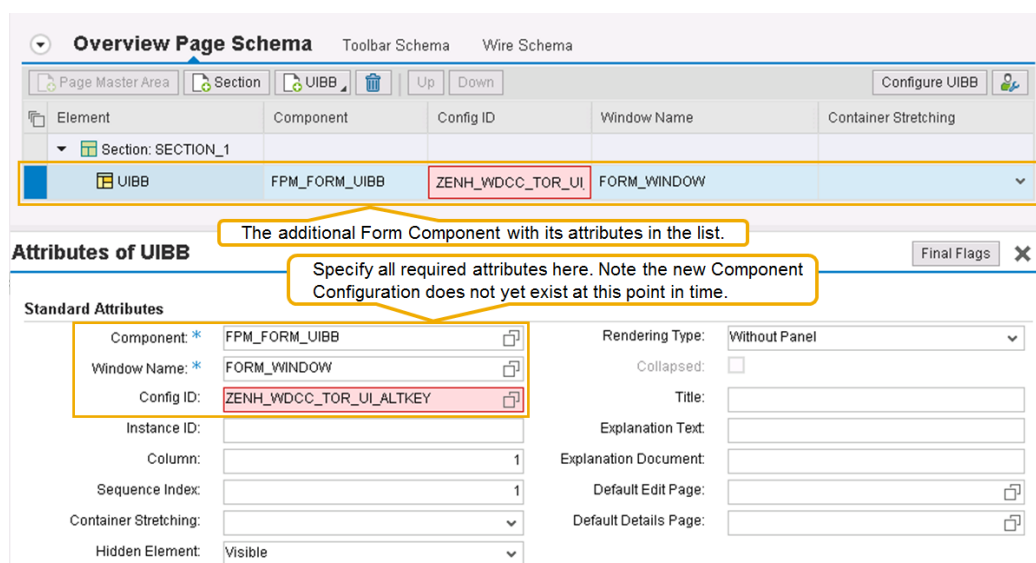


Picture: Defining the Initial Screen and the Main Page.

- On the left side of the Component Configuration Editor double click on the list entry for the Initial Screen. Now navigate to the related Overview Page Schema. Here you should now find a section **SECTION\_1**. Select it in the list, display its attributes and make sure that it has the following settings:

Field	Value
Section ID	SECTION_1
Layout Type	One Column Layout (Standard Layout)

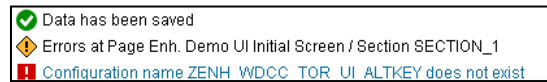
Now add a new Form UIBB as a subcomponent of the displayed **SECTION\_1**. For this click on button **UIBB** and choose option *Form Component* to add a new Form UIBB to the Initial Screen of the new UI.



Picture: Adding a new Form UIBB to the Initial Screen.

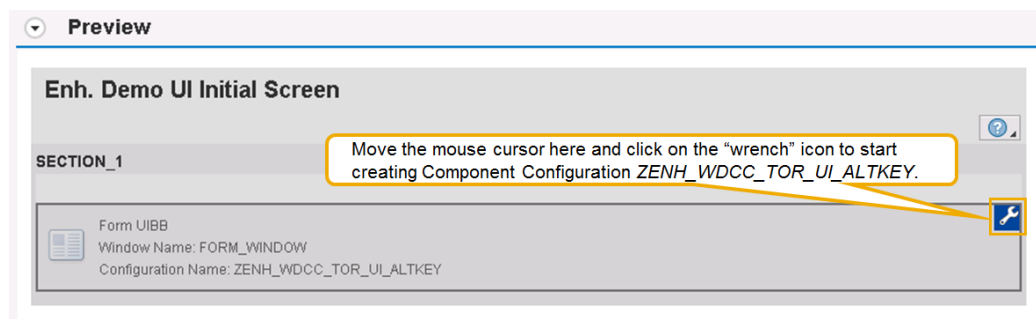
Field	Value
Component	FPM_FORM_UIBB
Window Name	FORM_WINDOW
Configuration ID	ZENH_WDCC_TOR_UI_ALTKEY

Maintain the attributes for the new Form UIBB and then save the Component Configuration. As you can see in the message list below, the current Component Configuration is saved but the new one (*ZENH\_WDCC\_TOR\_UI\_ALTKEY*) we just created does not yet exist. So we have to create it in the next step.

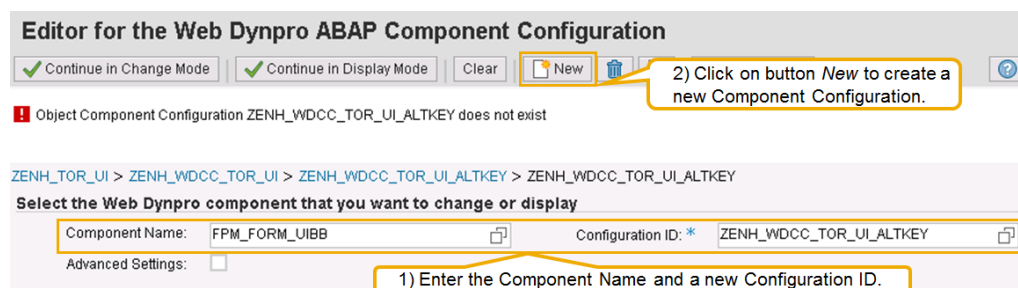


Picture: Messages after Save.

- 7) Open the *Preview* section. Here you can find the Form UIBB and its Configuration Name. When you move over this Form UIBB with the mouse you should see the “wrench” icon. Click on this icon to start creating the Component Configuration for the Form UIBB (Alternative: Click on button *Configure UIBB* in the Overview Page Schema).



Picture: The new Component Configuration (not yet created) in the Preview.



Picture: Creating the new Component Configuration

Specify the following data and click on button *New* to create the Component Configuration:

Field	Value
Configuration Name	FPM_FORM_UIBB
Configuration ID	ZENH_WDCC_TOR_UI_ALTKEY

On the following popups specify a description and a package where to store the Component Configuration (which represents the content for the initial screen).

Field	Value
Description	Enhancement Demo UI Alternative Key Configuration
Package	\$TMP (or your own package)

- 8) Enter class `/BOFU/CL_FBI_GUIBB_ALTKEY_FDR` as the feeder class. Then click on button *Edit Parameters* and maintain the following attributes. Afterwards click on button *Ok*.

Field	Value
Business Object	/SCMTMS/TOR
Node	ROOT
Alternative Key	TOR_ID

Picture: Specifying the Feeder Class and its parameters.

Navigate to the Form UIBB Schema, click on button *Element* and add a new Group with title *Document Number*. Then select the new group in the list, click on button *Child Elements* and from the list of available fields add field *TOR\_ID* to the group. Maintain the following attributes for the field:

Field	Value
Field Name	TOR_ID → Is defaulted and cannot be changed.
Label Visibility	Is visible
Display Type	Input Field
Label	Document
FPM Event ID	FPM_LEAVE_INITIAL_SCREEN() → This will allow hitting <i>ENTER</i> in the field instead of having to click on button <i>Continue</i> to get to the main screen.

Picture: The final configuration of the Initial Screen.

Check and save this Component Configuration. Then navigate back to component configuration `ZENH_WDCC_TOR_UI`. This can be achieved by simply clicking the corresponding link in the header section of the Component Configuration Editor.

- 9) Add a second Form UIBB as a subcomponent of SECTION\_1 for the Initial Screen. For this click again on button *UIBB* and choose option *Form Component* to add a new Form UIBB to the Initial Screen of the new UI. Specify the following attributes:

Field	Value
Component	FPM_FORM_UIBB_GL2
Window Name	FORM_WINDOW
Configuration ID	ZENH_WDCC_TOR_UI_INIT
Sequence Index	2

The additional Form Component with its attributes in the list.

Specify all required attributes here. Note: the new Component Configuration does not yet exist at this point in time.

**Standard Attributes**

Component *	FPM_FORM_UIBB_GL2	Rendering Type:	Without Panel
Window Name *	FORM_WINDOW	Collapsed:	<input type="checkbox"/>
Config ID:	ZENH_WDCC_TOR_UI_INIT	Title:	
Instance ID:		Explanation Text:	
Column:	1	Explanation Document:	
Sequence Index:	2	Default Edit Page:	
Container Stretching:		Default Details Page:	
Hidden Element:	Visible		

Picture: Adding a second Form UIBB to the Initial Screen.

Again you can find the new Form UIBB already in the Preview but its configuration does not yet exist and is now created just like the first Form UIBB in step 7). So navigate to the Preview section and click on the “wrench” icon for the second Form UIBB to create it. On the entry screen of the Component Configuration Editor enter the following data and then click on button *New*:

Field	Value
Component Name	FPM_FORM_UIBB_GL2
Configuration ID	ZENH_WDCC_TOR_UI_INIT

On the following popups specify a description and a package where to store the Component Configuration (which represents the Bootstrap configuration for the new UI).

Field	Value
Description	Enh. Demo UI Bootstrap Configuration
Package	\$TMP (or your own package)

- 10) Enter class */BOFU/CL\_FBI\_GUIBB\_BOOTSTRAP* as the feeder class. Click on button *Edit Parameters* and specify the following attributes. Afterwards click on button *Ok*.



Picture: Specifying the Feeder Class and its parameters.

Field	Value
Business Object	/SCMTMS/TOR
Node	ROOT
URL Key Provider	/BOFU/CL_FBI_URL_KEYPROVIDER
Preselection Key Provider	/BOFU/CL_FBI_PSEL_KEYPROVIDER

Check and save this configuration. Then navigate back to component configuration `ZENH_WDCC_TOR_UI`.

- 11) Navigate to the Wire Schema, create a new Wire by clicking on button *Wire* and specify the following attributes with corresponding values:

Picture: Creating a new Wire on the Wire Schema.

Field	Value
Component	FPM_FORM_UIBB_GL2
Configuration Name	ZENH_WDCC_TOR_UI_INIT
Source Component	FPM_FORM_UIBB_GL2
Source Configuration Name	ZENH_WDCC_TOR_UI_ALTKEY
Port Type	Collection
Port Identifier	CO
Connector Class	/BOFU/CL_FBI_CONNECTOR

Picture: The attributes for the first Wire.

Click again on button *Wire* to specify a second required wire just like shown before. Specify the following attributes with corresponding values:



Field	Value
Component	/BOFU/WDC_FBI_CONTROLLER
Configuration Name	/SCMTMS/WDCC_APPCC
Source Component	FPM_FORM_UIBB_GL2
Source Configuration Name	ZENH_WDCC_TOR_UI_INIT
Port Type	Collection
Port Identifier	CO
Connector Class	/BOFU/CL_FBI_CONNECTOR

With this step we have finalized the general Application Configuration, the Bootstrap Configuration for the new UI as well as its Initial Screen Configuration. On the initial screen you will be able to enter a document number to be displayed. In the next steps we will add content to the Main Page where the data of the corresponding document will be shown.

- 12) On the left side of the Component Configuration Editor double click on the list entry for the Main Page (in section *Navigation*). Navigate to the related Overview Page Schema. Here you should now find a section *SECTION\_1* associated with the Main Page. Select it in the list, display its attributes and make sure that it has the following settings:

Field	Value
Section ID	SECTION_1
Layout Type	One Column Layout (Standard Layout)

Now add a new Form UIBB as a subcomponent of the displayed SECTION\_1. For this click on button *UIBB* and choose option *Form Component* to add a new Form UIBB to the Main Page of the new UI.

The screenshot shows the 'Overview Page Schema' interface. Under 'Section: SECTION\_1', a new UIBB 'FPM\_FORM\_UIBB\_GL2' is added with configuration 'ZENH\_WDCC\_TOR\_UI\_ROOT'. The 'Attributes of UIBB: Root Node Information' dialog is open, showing various attributes. The 'Config ID' field is highlighted with a red box, and a note states: 'Specify all required attributes here. Note: The new Component Configuration does not yet exist at this point in time.'

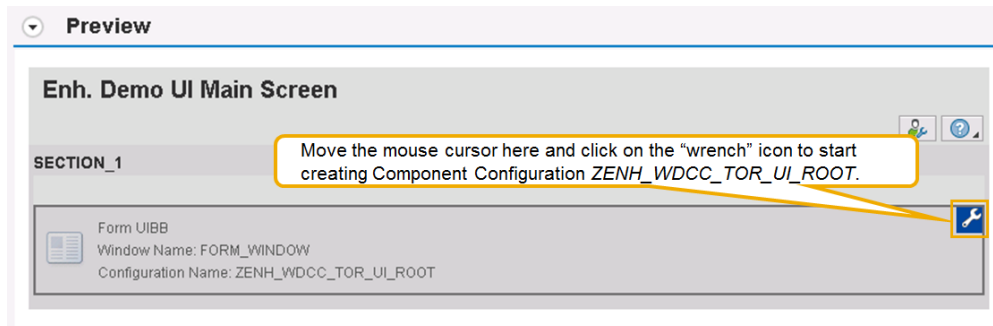
Picture: Adding a new Form UIBB to the Main Page.

Specify the following attributes for this new Form UIBB:

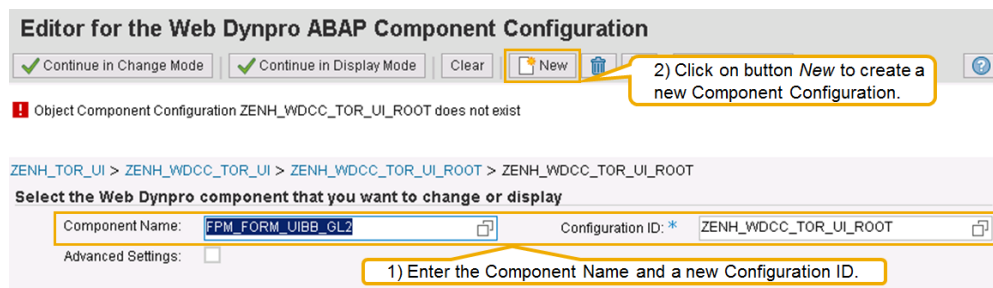
Field	Value
Component	FPM_FORM_UIBB_GL2
View	FORM_WINDOW
Configuration Name	ZENH_WDCC_TOR_UI_ROOT
Sequence Index	1
Title	Root Node Information

As described in the previous steps, the new Form UIBB will be visible in the Preview of the Main Page but its Component Configuration still has to be created.

- 13) Open the *Preview* section and use the mouse to move over the Form UIBB with the Configuration Name `ZENH_WDCC_TOR_UI_ROOT`. You should see the related “wrench” icon. Click on it to start creating the Component Configuration for the Form UIBB.



Picture: The new Component Configuration (not yet created) in the Preview.



Picture: Creating the new Component Configuration

Specify the following data and click on button *New* to create the Component Configuration:

Field	Value
Configuration Name	FPM_FORM_UIBB_GL2
Configuration ID	ZENH_WDCC_TOR_UI_ROOT

On the following popups specify a description and a package where to store the Component Configuration (which represents the content for the initial screen).

Field	Value
Description	Enh. Demo UI Root Data Configuration
Package	\$TMP (or your own package)

- 14) Enter class `/BOFU/CL_FBI_GUIBB_FORM` as the feeder class. Click on button *Edit Parameters* and specify the following attributes. Afterwards click on button *Ok*.

Picture: Specifying the Feeder Class and its parameters.

Field	Value
Business Object	/SCMTMS/TOR
Node	ROOT
Handles Toolbar	Yes (flag shall be set)

Navigate to the Form UIBB Schema, click on button *Element* and add a new Group with title *Root Node Information*. Maintain the following attributes for the new group:

Field	Value
Text	Root Node Information

Then select the new group in the list, click on button *Child Elements* and from the list of available fields add the following fields: *TOR\_ID*, *TOR\_CAT* and *TOR\_TYPE*. Maintain the following attributes for the fields:

Field	Attribute	Value
TOR_ID	Label Text	Document (defaulted)
	Display Type	Text View
TOR_CAT	Label Text	Document Category (defaulted)
	Display Type	Text View
TOR_TYPE	Label Text	Document Type (defaulted)
	Display Type	Text View

Add another new Group with title *Administrative Data*. Maintain the following attributes for the new group:

Field	Value
Text	Administrative Data

Select the new group in the list, click on button *Child Elements* and from the list of available fields add the following fields: *CREATED\_BY*, *CREATED\_ON*, *CHANGED\_BY* and *CHANGED\_ON*. Maintain the following attributes for the fields:

Field	Attribute	Value
CREATED_BY	Label Text	Created By (defaulted)
	Display Type	Text View
CREATED_ON	Label Text	Created On (defaulted)
	Display Type	Text View
CHANGED_BY	Label Text	Changed By (defaulted)
	Display Type	Text View

	Display Type	Text View
<i>CHANGED_ON</i>	Label Text	Changed On (defaulted)
	Display Type	Text View

Picture: The final two groups with their fields on the Form UIBB Schema.

Check and save this configuration. Then navigate back to component configuration *ZENH\_WDCC\_TOR\_UI*.

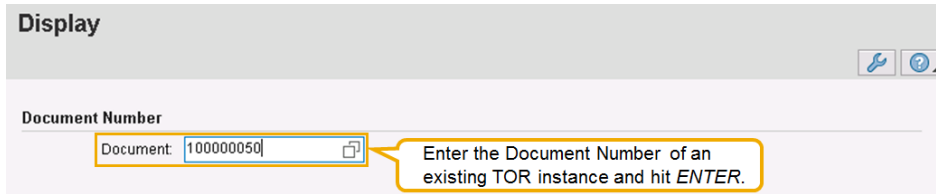
- 15) Navigate to the Wire Schema, create a new Wire by clicking on button *Wire* and specify the following attributes with corresponding values:

Field	Value
Component	FPM_FORM_UIBB_GL2
Configuration Name	ZENH_WDCC_TOR_UI_ROOT
Source Component	FPM_FORM_UIBB_GL2
Source Configuration Name	ZENH_WDCC_TOR_UI_INIT
Port Type	Collection
Port Identifier	CO
Connector Class	/BOFU/CL_FBI_CONNECTOR

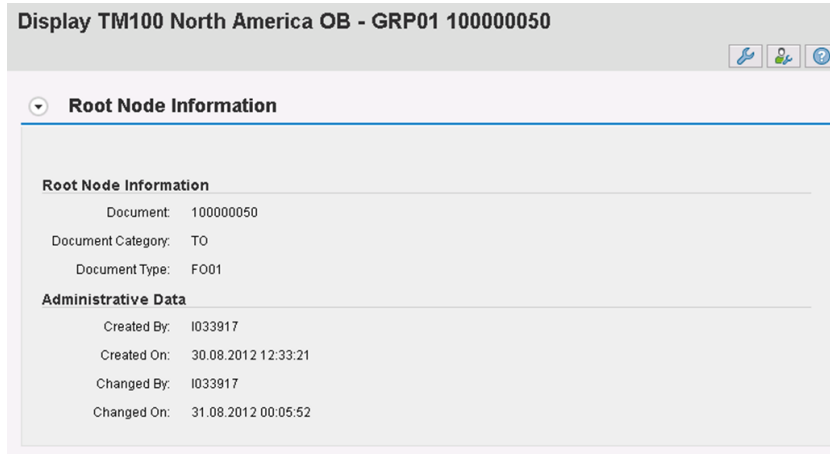
Picture: Creating a new Wire on the Wire Schema.

Check and save component configuration *ZENH\_WDCC\_TOR\_UI* and navigate to Application Configuration *ZENH\_TOR\_UI*.

Our new UI is now ready for a first test. On the Application Configuration screen click on button *Test* to start the new user interface. Enter an existing TOR ID and hit *ENTER* to continue. You should now see the following initial and main screen:



Picture: The initial screen of the new UI.

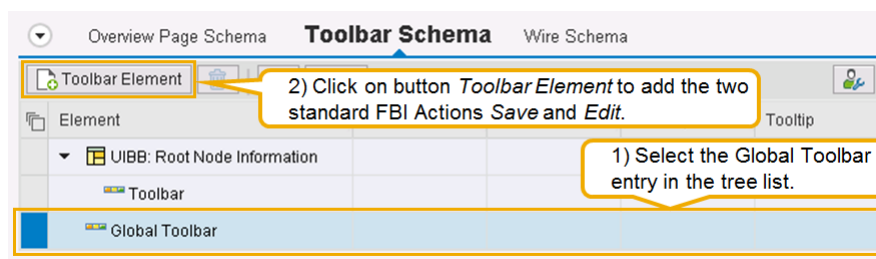


Picture: The main screen of the new UI.

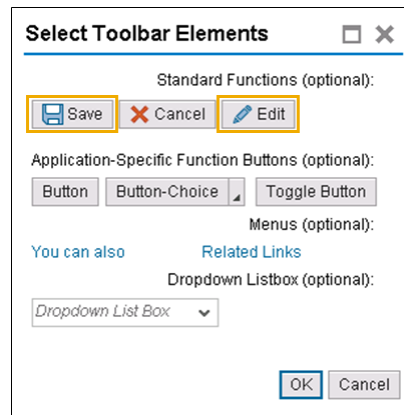
You can now add further UIBBs with corresponding configurations and wires between them to add more functionality to your user interface. The steps 1 – 15 show the very basic steps how to make use of the FBI feeder classes that are already implemented and available. In the next steps we will add further functionality.

- 16) The next step is adding buttons in the toolbar of the main screen that allow saving a changed document and switching a document into Edit Mode. Go back to Component Configuration *ZENH\_WDCC\_TOR\_UI* and in section *Navigation* on the left side of the Component Configuration Editor double click on the entry for the Main Screen.

Go to the Toolbar Schema of the Main Screen and select the entry for the Global Toolbar in the tree list. Click on button *Toolbar Element* to add the Standard Functions *Save* and *Edit*.



Picture: Adding the actions Save and Edit to the Global Toolbar.



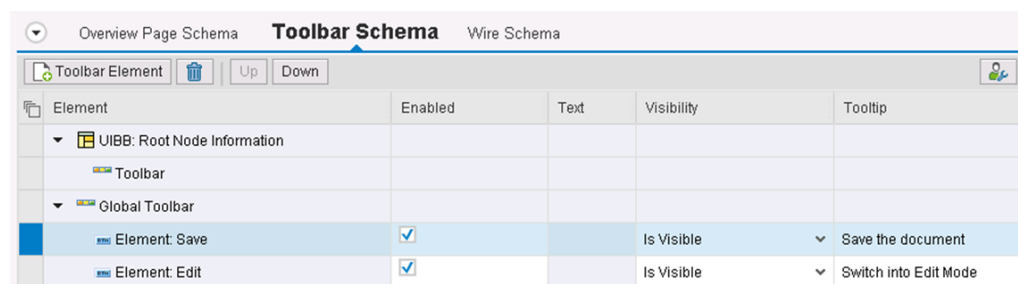
Picture: Adding a Toolbar Element.

On the following popup click on button *Save* to add the save button. Click again on button *Toolbar Element* to add the second button *Edit* to the Global Toolbar. Specify the following attributes for the two new buttons:

Field	Value
Element ID	FPM_SAVE_1 (defaulted)
Add Separator	Yes
FPM Event ID	FPM_SAVE (is set and used automatically)

Field	Value
Element ID	FPM_EDIT_1 (defaulted)
Add Separator	Yes
FPM Event ID	FPM_EDIT (is set and used automatically)

When data is displayed on the main screen you can now click on button *Edit* to switch the displayed document into Edit Mode. Input fields can now be entered. Text View fields remain Display-Only fields. Moreover you can now click on button *Save* to save changed data.



Picture: The final configuration of the Main Screen Global Toolbar.

- 17) Let's add a second Form UIBB on the Main Screen to display Business Partner Data coming from the TOR Root node. Proceed as described in the steps 12 – 15. On the Overview Page Schema, click on button *UIBB* , add a new *Form Component* and maintain the following attributes for it:

Field	Value
Component	FPM_FORM_UIBB_GL2
Window Name	FORM_WINDOW
Configuration ID	ZENH_WDCC_TOR_UI_BUPA
Sequence Index	2
Title	Business Partner Data

Again you can find this new Form UIBB in the Preview section where you can start creating the required Component Configuration `ZENH_WDCC_TOR_UI_BUPA`. Assign class `/BOFU/CL_FBI_GUIBB_FORM` as the feeder class for the Form UIBB and specify the following Feeder Class Parameters.

Field	Value
Business Object	/SCMTMS/TOR
Node	ROOT
Handles Toolbar	Yes

On the Form UIBB Schema click on button *Element* and add a new Group with title *Root Business Partner Data*. Maintain the following attributes for the new group:

Field	Value
Text	Root Business Partner Data

Select the new group in the list, click on button *Child Elements* and from the list of available fields add the following fields: `CONSIGNEEID`, `SHIPPERID` and `TSPID`. Maintain the following attributes for the fields:

Field	Attribute	Value
<code>CONSIGNEEID</code>	Label Text	Consignee (defaulted)
	Display Type	Input Field
<code>SHIPPERID</code>	Label Text	Shipper (defaulted)
	Display Type	Input Field
<code>TSPID</code>	Label Text	Carrier (defaulted)
	Display Type	Input Field

Check and save this configuration. Then navigate back to component configuration `ZENH_WDCC_TOR_UI`.

- 18) Navigate to the Wire Schema, create a new Wire by clicking on button *Wire* and specify the following attributes with corresponding values:

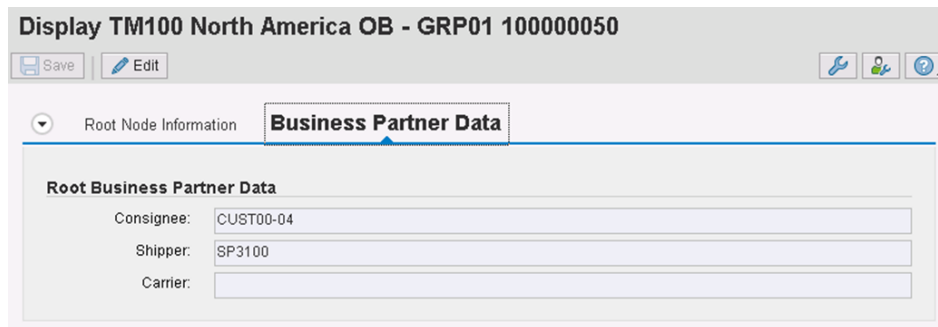
Field	Value
Component	FPM_FORM_UIBB_GL2
Configuration Name	ZENH_WDCC_TOR_UI_BUPA
Source Component	FPM_FORM_UIBB_GL2
Source Configuration Name	ZENH_WDCC_TOR_UI_INIT
Port Type	Collection
Port Identifier	CO
Connector Class	/BOFU/CL_FBI_CONNECTOR

Picture: Creating a new Wire on the Wire Schema.

Check and save component configuration `ZENH_WDCC_TOR_UI` and navigate to Application Configuration `ZENH_TOR_UI`. Here you can click on button *Test* again to start the UI again. You should now see the second Form UIBB represented as a tab strip



containing Business Partner Data that is stored on the TOR Root node along with the two standard functions *Save* and *Edit* that we had added to the Global Toolbar.



Picture: The additional tab strip for Business Partner Data on the new UI.

You can now switch into Edit Mode and e.g. enter a carrier in the corresponding field. Then save the document, refresh the browser (press *F5*), display the same document again and verify that this change has been persisted in the document.

- 19) As a last step we now add a List UIBB on the Main Screen to display Item Data associated with the TOR Root node. Again proceed as described in the steps 12 – 15. Go to the Overview Page Schema, click on button *UIBB* , add a new *List Component* and maintain the following attributes for it:

Field	Value
Component	FPM_LIST_UIBB_ATS
Window Name	LIST_WINDOW
Configuration ID	ZENH_WDCC_TOR_UI_ITEM
Sequence Index	3
Title	Item Node Information

Go to the Preview section and start creating the required Component Configuration *ZENH\_WDCC\_TOR\_UI\_ITEM*. Assign class */BOFU/CL\_FBI\_GUIBB\_LIST* as the feeder class for the List UIBB and specify the following Feeder Class Parameters.

Field	Value
Business Object	/SCMTMS/TOR
Node	ITEM_TR
Handles Toolbar	Yes

Go to the List UIBB Schema and click on button *Column*. From the list of available columns choose the following ones and add them to the List UIBB: *ITEM\_ID*, *ITEM\_DESCR*, *PRODUCT\_ID*, *GRO\_VOL\_VAL*, *GRO\_VOL\_UNI*, *GRO\_WEI\_VAL* and *GRO\_WEI\_UNI*.

Check the attributes for each of these fields. Initially all added fields are defined to be *Text Views* in attribute *Display Type*, i.e. they are defined to be Display-Only fields. If you want any of the listed fields to be input fields where data can be entered or adjusted in Edit Mode change the attribute *Display Type* of the field to *Input Field*. Define e.g. the fields *GRO\_VOL\_VAL* and *GRO\_WEI\_VAL* as input fields.

Now go to the Toolbar Schema of the List UIBB, click on button *Toolbar Element* and from the list of available actions add the standard FBI Actions *FBI\_CREATE* and *FBI\_DELETE* to the toolbar of the list UIBB. They can then be used to add or remove items to a document in Edit Mode (of course you can any of the other actions from the list of available actions).

Picture: The final configuration of the List UIBB.

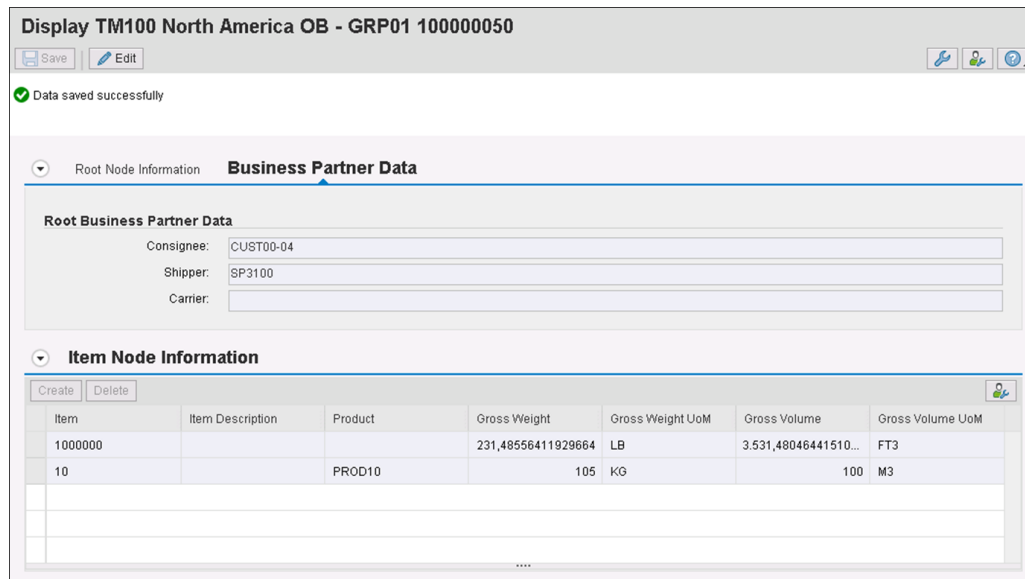
Check and save this configuration. Then navigate back to component configuration *ZENH\_WDCC\_TOR\_UI*.

- 20) Navigate to the Wire Schema, create a new Wire by clicking on button *Wire* and specify the following attributes with corresponding values:

Field	Value
Component	FPM_LIST_UIBB_ATS
Configuration Name	ZENH_WDCC_TOR_UI_ITEM
Source Component	FPM_FORM_UIBB_GL2
Source Configuration Name	ZENH_WDCC_TOR_UI_INIT
Port Type	Collection
Port Identifier	CO
Connector Class	/BOFU/CL_FBI_CONNECTOR
Source Node Association	ITEM_TR

Picture: Creating a new Wire on the Wire Schema.

Check and save component configuration *ZENH\_WDCC\_TOR\_UI*. For testing the UI again navigate to Application Configuration *ZENH\_TOR\_UI* and click on button *Test*. You should now see the final UI with the configured List UIBB represented as a tab strip containing also the Item data associated with the TOR Root node of the displayed document. You can use the personalization to make the Item List e.g. a separate section, switch into Edit Mode and save changes done to the document.



Display TM100 North America OB - GRP01 100000050

Save Edit

Data saved successfully

Root Node Information **Business Partner Data**

Root Business Partner Data

Consignee: CUST00-04

Shipper: SP3100

Carrier:

Item Node Information

Create Delete

Item	Item Description	Product	Gross Weight	Gross Weight UoM	Gross Volume	Gross Volume UoM
10000000			231,48556411929664	LB	3.531,48046441510...	FT3
10		PROD10	105	KG	100	M3

Picture: The final Demo UI that was built from scratch.

Suggest playing around a bit with this UI that we built from scratch and add a few more things based on the UI enhancement use cases that were described so far.

## 5.4.8 Copying a complete FPM-based Application

The first 7 UI Enhancement Use Cases described examples how to customize existing standard Component Configurations as well as how to create and include your own Component Configurations into the standard SAP TM UI. In the last section we built a complete UI from scratch with quite some steps required to get it up and running (nevertheless, these steps can be repeated and executed in less than 45 minutes!).

There is a Web Dynpro Application available, the so called *FPM Configuration Hierarchy Browser*, which allows not only browsing through the Component Configuration Hierarchy of an existing FPM-based application. In addition it also allows creating a deep copy of this Component Configuration Hierarchy. The resulting copy represents again a running FPM-based application.

The FPM Configuration Hierarchy Browser can be started with the following general link which has to be enhanced with information on server and port, depending on where you want to start the tool.

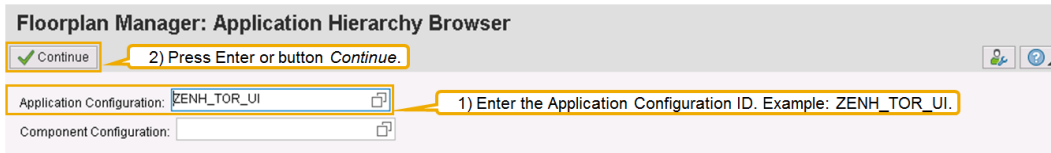
[http://\[server\]:\[port\]/sap/bc/webdynpro/sap/fpm\\_cfg\\_hierarchy\\_browser](http://[server]:[port]/sap/bc/webdynpro/sap/fpm_cfg_hierarchy_browser)

Example:

[http://ukwtr9x.wdf.sap.corp:80089/sap/bc/webdynpro/sap/fpm\\_cfg\\_hierarchy\\_browser](http://ukwtr9x.wdf.sap.corp:80089/sap/bc/webdynpro/sap/fpm_cfg_hierarchy_browser)

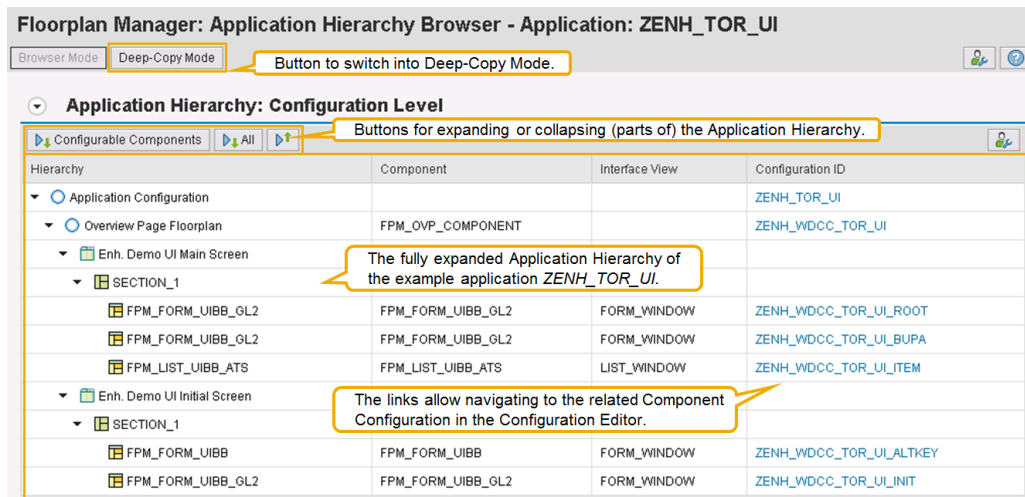
On the initial screen you can enter the technical name of an Application Configuration and display its Component Configuration Hierarchy. Remember the steps to create a UI from scratch in the previous section 5.4.7. Here you can now pretty nice see the hierarchy of Component Configurations that we created. Let's take a look at the example UI that we created.

- 1) Start the tool as mentioned above and enter the Application Configuration ID ZENH\_TOR\_UI in input field *Application Configuration*. Then press *Enter* or click on button *Continue*.



Picture: Start screen of the Application Hierarchy Browser.

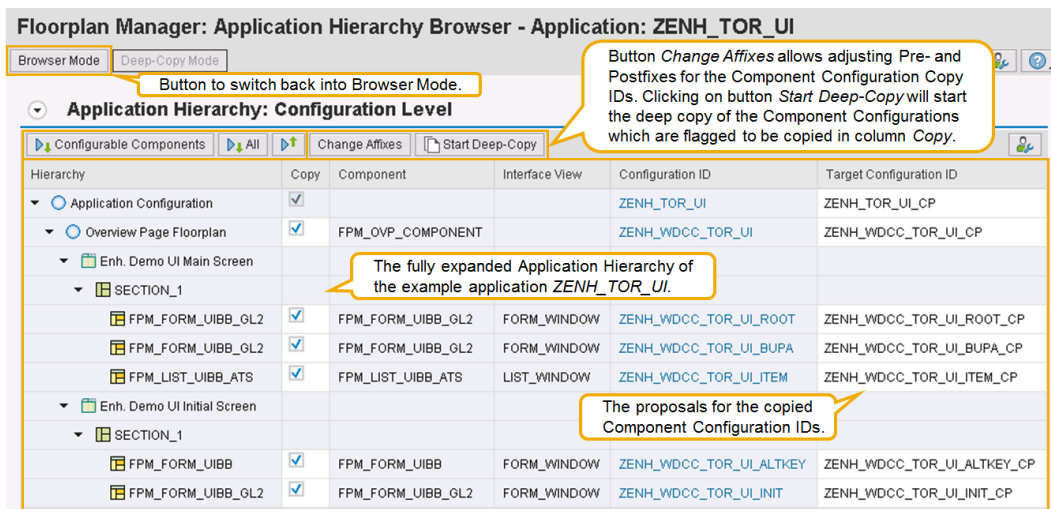
- 2) On the following screen you will first of all see the Application Hierarchy in *Browser Mode*. This mode allows browsing through the hierarchy of Component Configurations that make up the given FPM-based application. Each Component Configuration can be reached from here by clicking on the corresponding link on the right side of the tree list. So this tool is actually very helpful with getting an overview of the Component Configurations involved in a complete FPM-based application.



Picture: The Application Hierarchy in Browser Mode.

- 3) Click on button *Deep-Copy Mode* to switch into the deep-copy mode. Here you can now determine which Component Configurations you want to copy by a deep-copy and take it over to the new Application Configuration that the copy will represent.

Clicking on button *Change Affixes* allows you specifying Pre- and Postfixes for the Component Configuration Copy IDs. Here you need to make sure of course that nothing is copied with the same name or ID of an already existing Component Configuration. In the example on the picture below the default proposal for a postfix *CP* was simply taken over. As a result all Target Configuration IDs end with *CP* ("copy").

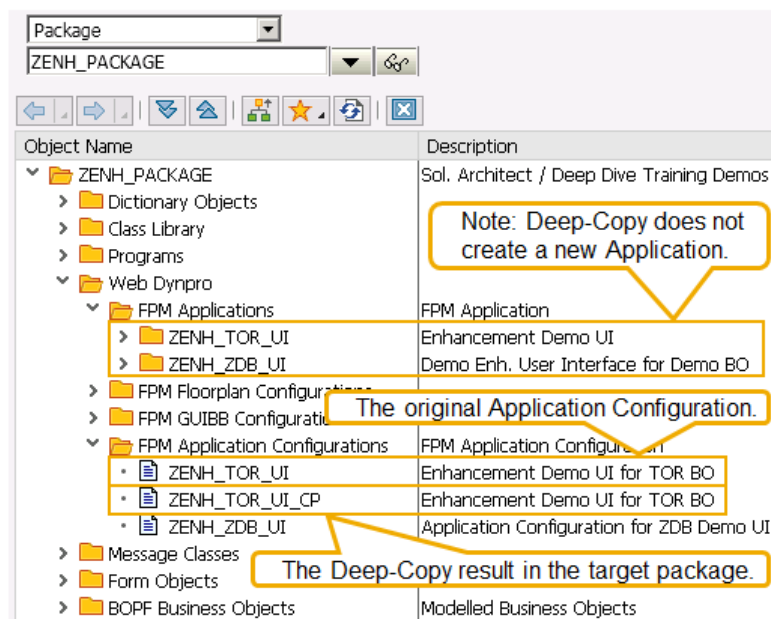


Picture: The Application Hierarchy in Deep-Copy Mode.

Before you actually start the deep-copy of the Application Configuration you can define for each involved Component Configuration whether it shall be copied over with a corresponding target Component Configuration ID to the new Application Configuration.

In the example shown in the picture above the tree list is fully expanded and all the involved Component Configurations are visible. In column *Copy* of the tree list you can set or reset the flag indicating that the Component Configuration shall be copied (set) or not (reset).

- 4) Click on button Start Deep-Copy once you have finally selected the Component Configurations to be taken over into the copy. On the following two popups you will be asked to specify a package and (if required) a transport request for the new objects that will be created by the copy process. The result is a new Application Configuration ZENH\_TOR\_UI\_CP with all the copied sub-configurations in its hierarchy.



Picture: The Deep-Copy result in the target package (SE80).

Note: The application in the example is *ZENH\_TOR\_UI* which we created in section 5.4.7 from scratch. What is now copied is not the application but the Application Configuration (which in this case has also has the name *ZENH\_TOR\_UI*). The result of the deep-copy is another, alternative Application Configuration of Application *ZENH\_TOR\_UI*.

- 5) If you want to also have an explicit new Web Dynpro Application you can create one as described in section 5.4.7 steps 1 and 2. You can then assign the Deep-Copy result, i.e. the Application Configuration *ZENH\_TOR\_UI\_CP* as the Application Configuration of your new Application.

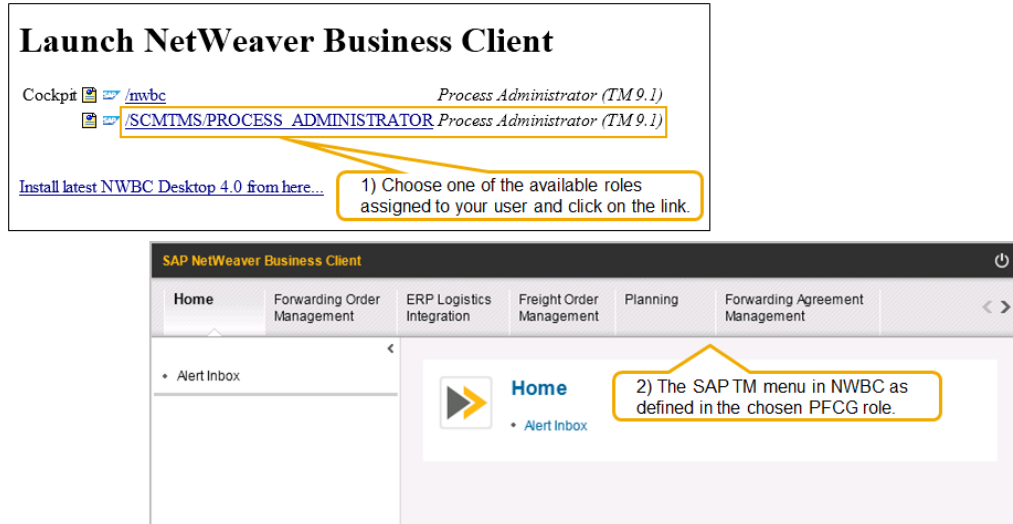
## 5.4.9 Adding a Web Dynpro Application to NWBC

In the previous sections a FPM-based application was created. Of course users will not start the application from within transaction *SE80* or others but you may want to make it accessible for a specific user role in the NWBC environment as part of either an existing functional area or as a new, independent functional area. The following sections show how this can be done.

The following steps describe how to use transaction *PFCG* for adjusting the role that you use along with your user to start the SAP TM User Interface in NWBC.

- 1) Usually you start the SAP TM User interface by starting transaction *NWBC*, i.e. the NetWeaver Business Client. On the initial screen you get a list of user roles that you can use in the next step to start the user interface.

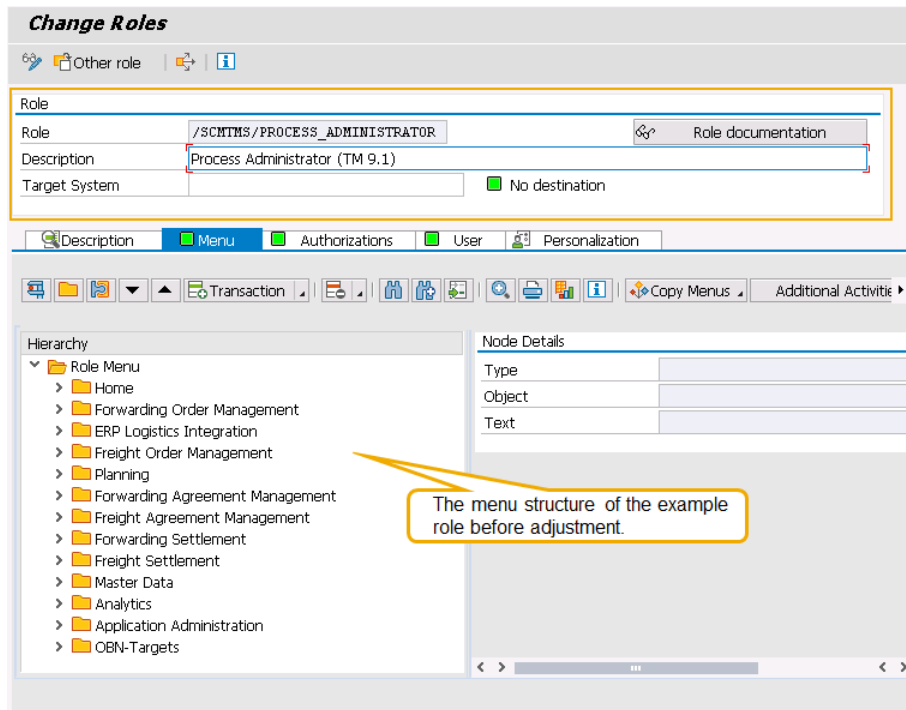
Start transaction *NWBC* and note down the role that you use to start the user interface. This should look as indicated on the following pictures:



Picture: Starting NWBC and choosing a user role.

- 2) In this example we choose PFCG Role */SCMTMS/PROCES\_ADMINISTRAOR* to start the SAP TM User Interface. Start transaction *PFCG* and enter the mentioned role in field *Role* of the initial screen.

Change into Change Mode to enable adjusting the role menu and then navigate to tab strip *Menu*. Here you can first of all see the current definition of the folder structure that contains the menu content.

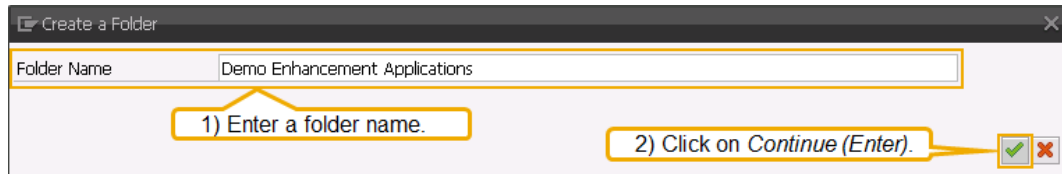


Picture: The menu structure of the example role.



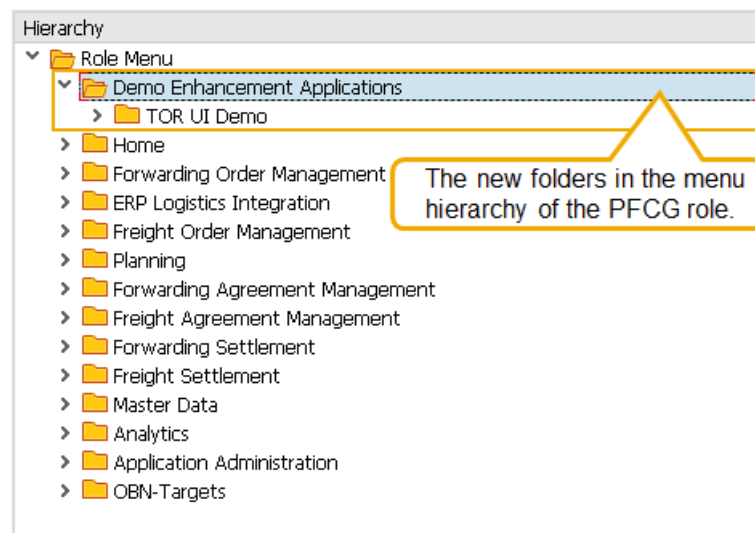
- 3) We now create our own folder substructure in the menu hierarchy to define our own menu area that will contain the FPM-based application created in section 5.4.7.

In the menu hierarchy mark the topmost folder *Role Menu* and click on button *Create Folder*. On the following popup enter a folder name in field *Folder Name*. Example: *Demo Enhancement Applications*.



Picture: Creating a new folder in the menu hierarchy.

Now mark the new folder, click on button *Create Folder* again and create an additional folder (i.e. a subfolder) below the first one. On the popup enter a folder name in field *Folder Name*. Example: *TOR UI Demo*. Finally the



Picture: The adjusted menu hierarchy.

- 4) For both new folders a few further properties shall be specified. Mark the new folder *Demo Enhancement Applications* in the menu hierarchy and click on button *Other Node Details* on the toolbar above. Make sure that the following properties are set for the folder:

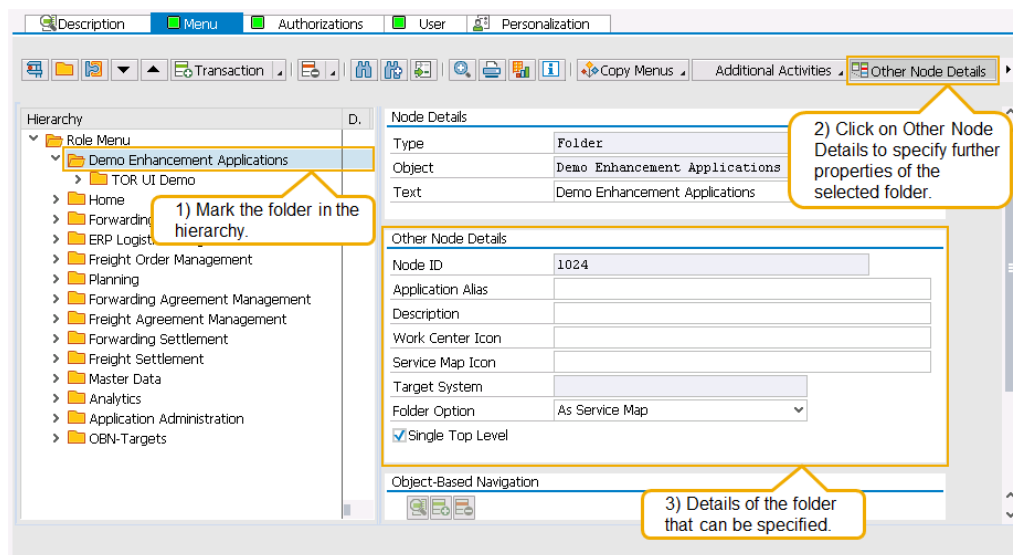
Field	Value
Text	<i>Demo Enhancement Applications</i> (this is actually defaulted already from the folder object name; you may want to adjust it)
Folder Option	<i>As Service Map</i> .  A service map shows an overview of a navigation structure with the entries of the next navigational level in the navigation tree, which provides the end user with a better overview of the business process that is represented by the folder.
Single Top Level	Yes.
Service Map Icon	(Optional) Here you could define/change the icon that is used to visualize a folder within a service map.

Repeat the same steps for the second new folder *TOR UI Demo* with the following settings:



Field	Value
Text	<i>TOR UI Demo</i> (this is actually defaulted already from the folder object name; you may want to adjust it)
Folder Option	<i>As Service Map.</i>  A service map shows an overview of a navigation structure with the entries of the next navigational level in the navigation tree, which provides the end user with a better overview of the business process that is represented by the folder.
Single Top Level	<i>(is always blank for folders that are not root folders)</i>
Service Map Icon	(Optional) Here you could define/change the icon that is used to visualize a folder within a service map.

With these settings you can influence the way how the new menu hierarchy parts will later on be represented in NWBC. For more options provided with these settings check out the F1-Help for each property in your system. The documentation provided there explains available values and settings in more detail.



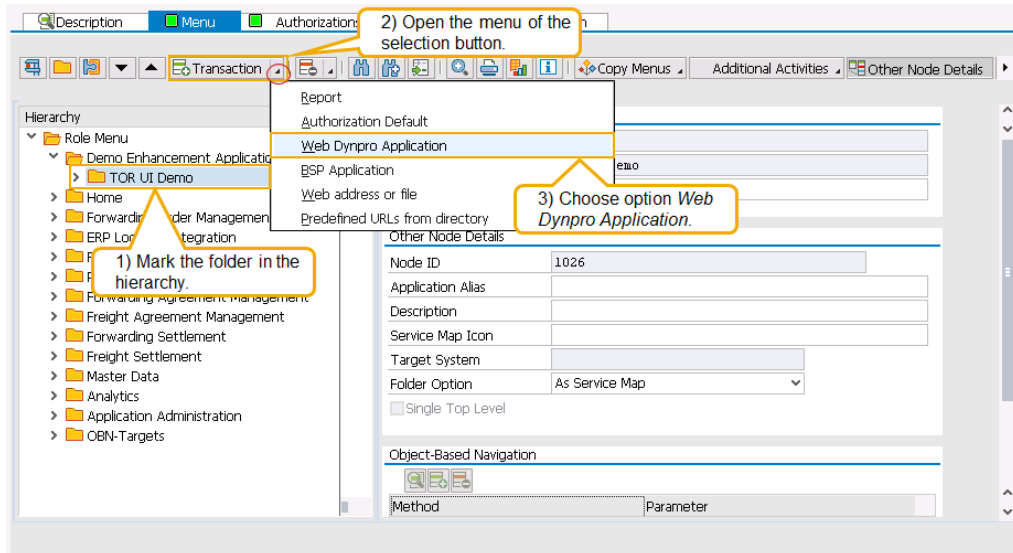
Picture: Specifying further folder properties.

- 5) Add an FPM-based application to example folder *TOR UI Demo*. Mark this folder in the menu hierarchy. Open the menu of selection button Insert Node and choose option *Web Dynpro Application*.

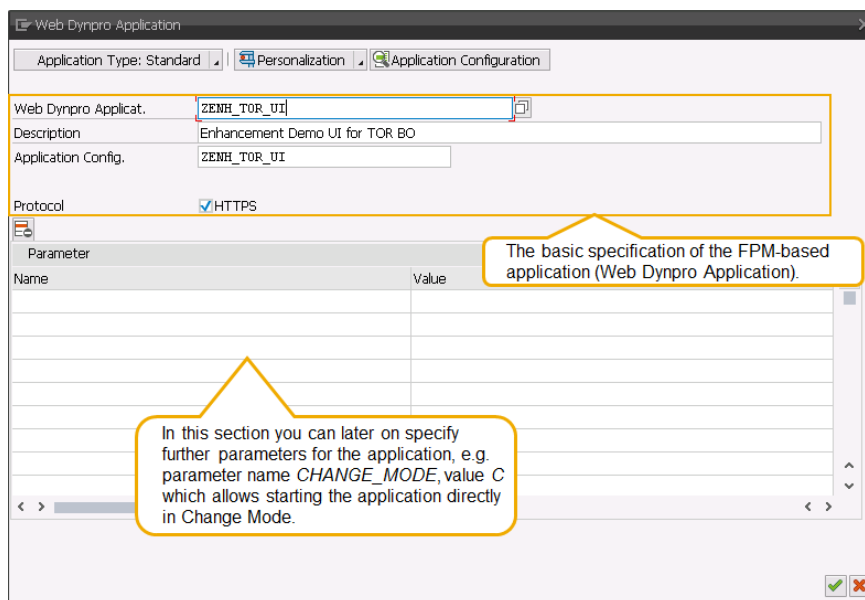
On the following popup specify the following properties:

Field	Value
Web Dynpro Application	ZENH_TOR_UI
Description	Enhancement Demo UI for TOR BO
Application Configuration	ZENH_TOR_UI
Protocol - <i>HTTPS</i>	Yes.

Then click on Enter to add the application to the chosen folder. You should now see a corresponding entry for the FPM-based application under folder *TOR UI Demo*.



Picture: Adding the FPM-based application to the new folder.

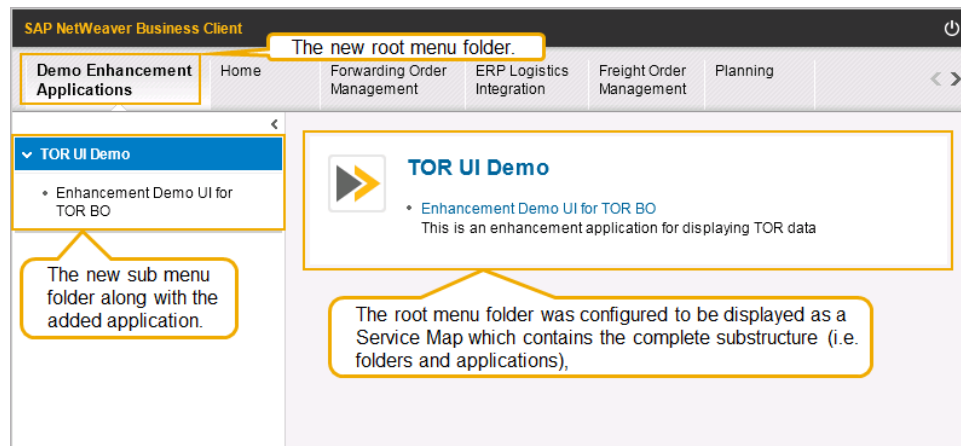


Picture: Specifying the properties of the FPM-based application.

- 6) In the next step we can specify a few more properties for this new application entry. Mark the application in folder *TOR UI Demo* and click again on button *Other Node Details* as shown already in step 4 for the folders. Make sure that the following properties are set for the application:

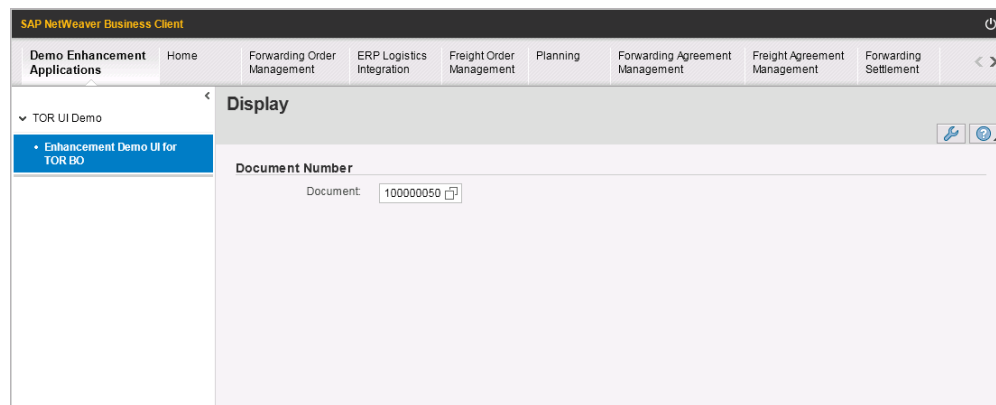
Field	Value
Text	<i>Enhancement Demo UI for TOR BO</i> (this is actually defaulted already from the folder object name; you may want to adjust it)
Description	This is an enhancement application for displaying TOR data.
Visibility	Visible
Launch Application	Standard

- 7) Start transaction *NWBC* in your system and chose the user role that were adjusted in the previous steps. You should then see the following result:

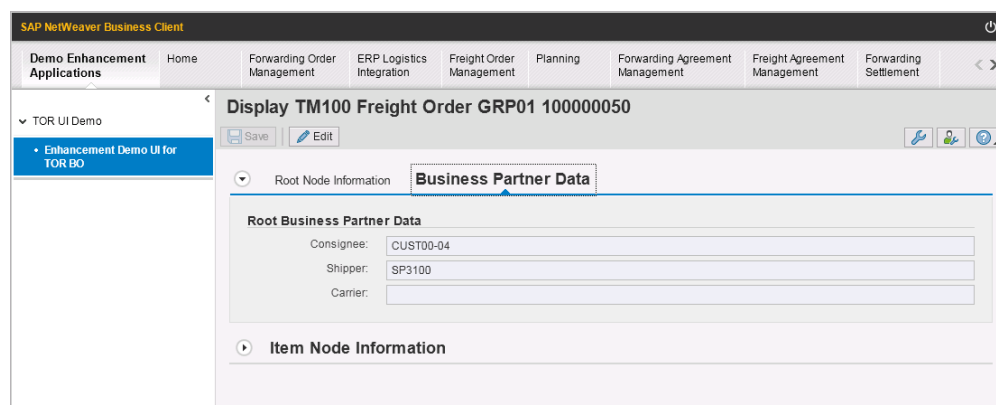


Picture: The new menu structure and application in NWBC.

- 8) You can now start your application just like any other standard application in NWBC.



Picture: The initial screen of the new application in NWBC.



Picture: An example document displayed with the new application in NWBC.

## 5.5 Transporting or removing UI enhancements

Transporting as well as deleting created UI customizing is possible via a corresponding Web Dynpro application that can be started with the following general link which has to be enhanced with information on server and port, depending on where you want to start the tool.

[http://\[server\]:\[port\]/sap/bc/webdynpro/sap/wd\\_analyze\\_config\\_comp](http://[server]:[port]/sap/bc/webdynpro/sap/wd_analyze_config_comp)

Example: [http://ukwtr9x.wdf.sap.corp:80089/sap/bc/webdynpro/sap/wd\\_analyze\\_config\\_comp](http://ukwtr9x.wdf.sap.corp:80089/sap/bc/webdynpro/sap/wd_analyze_config_comp)

Here you can find the standard Component Configuration as well as your own Component Configurations. The tool allows searching Component Configurations by Component Name (e.g. search all Component Configurations for *FPM\_LIST\_UIBB\**), Configuration (e.g. search for all Component Configuration IDs starting with */SCMTMS/WDCC\_FRE\_ORD\**) and by Author (e.g. search all Component Configurations created by user *XYZ*).

You can mark an entry in the result list and either transport it from here through your system landscape or delete it if required. Moreover it allows navigating to the related Personalization and start the Configuration Editor for the selected entry. Example:

- 1) Start the tool as mentioned above. In field *Find by* select *Configuration* and in the input field next to the drop down list enter the name of a configuration that you want to take a look at. Example: */SCMTMS/WDCC\_FRE\_ORDER*.

Picture: The initial screen of the tool.

- 2) On following screen mark the Component Configuration of interest in the search result list and click on button *Goto Personalization*. In the list of Component Configurations select the entry with Component Configuration ID */SCMTMS/WDCC\_FRE\_ORDER*. You can expand this entry to display further detailed properties of this Component Configuration.

Picture: Navigating to the Personalization of the Component Configuration.

The button *Delete* allows deleting a selected Component Configuration. With button *Transport* you can trigger the transport of the selected Component Configuration. A following popup will ask you to provide a corresponding transport request.

- 3) Click on the link *Goto Personalization*. On the following screen you can see the list of personalization entries that are currently available for the selected Component Configuration (see List of Component Personalizations).

Personalization can be deleted and transported from here if required. Moreover, the Customizing Editor can be started here.

There is a Personalization available that is valid for all users (marked list entry shows \* in column User)

The XML Display for the existing personalization shows all details in an XML representation.

Example for an Explicit Personalization: An additional List UIBB Component Configuration that was added via the standard Component Configuration /SCMTMS/WDCC\_FRE\_ORDER

User	User Scope	Component	Config ID	Config. Variant	Initial View	Description	Configuration Type	Last Changed By	Changed On
*	A	FPM_OVP_COMPONENT	/SCMTMS/WDCC_FRE_ORDER			Demo Enh. Field Extension	00	SCMSUPPORT	10.10.2013
BHAT	U	FPM_OVP_COMPONENT	/SCMTMS/WDCC_FRE_ORDER			Demo Enh. Field Extension	00	BHAT	17.10.2012
BROEKEMIER	U	FPM_OVP_COMPONENT	/SCMTMS/WDCC_FRE_ORDER			Demo Enh. Field Extension	00	BROEKEMIER	27.11.2013
C5125344	U	FPM_OVP_COMPONENT	/SCMTMS/WDCC_FRE_ORDER			Freight Order	00	C5125344	19.09.2012
C5161703	U	FPM_OVP_COMPONENT	/SCMTMS/WDCC_FRE_ORDER			Freight Order	00	C5161703	25.09.2012

Explicit Personalization

Path	Attribute	Value
CONFIGURATION_CONTEXT.000001.OVP_APPLICATION.000001.CONTENT_AREA.000001.SECTION.000001.UIBB.000001	LOCATION	
CONFIGURATION_CONTEXT.000001.OVP_APPLICATION.000001.CONTENT_AREA.000001.SECTION.000001.UIBB.000001	TYPE	T
CONFIGURATION_CONTEXT.000001.OVP_APPLICATION.000001.CONTENT_AREA.000001.SECTION.000001.UIBB.000001	ROW	1
CONFIGURATION_CONTEXT.000001.OVP_APPLICATION.000001.CONTENT_AREA.000001.SECTION.000001.UIBB.000001	CONFIG_ID	ZENH_WDCC_DEMO_LIST_UIBB
CONFIGURATION_CONTEXT.000001.OVP_APPLICATION.000001.CONTENT_AREA.000001.SECTION.000001.UIBB.000001	TITLE	1

Picture: Deleting and transporting personalization.

Again the buttons *Transport* and *Delete* allows transporting or deleting personalization. Moreover you can follow the link *Call Customizing Editor* to start the Component Customizing Editor for the related Component Configuration ID.

In the example shown on the picture above you can see that there is a personalization record selected that has *User \** and *User Scope A*. This indicates a personalization of Component Configuration /SCMTMS/WDCC\_FRE\_ORDER which is valid for all users while the other shown entries have an individual user and User Scope U, i.e. the personalization associated with these records is only valid for that particular user.

In general, the described tool allows you to search for Component Configurations and then navigating to the related personalization. You can of course also search here for your very own Component Configurations and delete or transport them from here. Moreover, you can delete transport associated personalization, e.g. the enhancement of a standard Component Configuration with an additional List UIBB as shown in the picture above. Moreover you can start the Component Configuration Editor in Configuration or Customizing mode to further edit e.g. your own Component Configurations or customize existing standard Component Configurations.

## 6 Enhancing Queries and POWL

### 6.1 Queries

#### 6.1.1 General concept

As described in section 3.3 the BOBF Enhancement Workbench does not support enhancing existing standard BO queries. It only supports adding new such queries. The only way to extend standard Node Attribute Queries in the standard BOPF environment is to add further query node attributes, i.e. adding extension fields to the query node. For standard Custom Queries and Generic Result Queries the following section describes a concept how to enhance them.

The majority of SAP TM Queries that require an implementation (i.e. Custom Queries or Generic Result Queries) are derived from super class `/SCMTMS/CL_Q_SUPERCLASS`. This super class not only helps making these queries work in a unified way but also provides an enhancement mechanism which makes such standard SAP TM queries extensible.

The query super class provides an API for creating an optimized database `SELECT` statement to execute the query and returning the requested result table. For this the class implements the BOPF interface `/BOBF/IF_FRW_QUERY` for queries and further methods that are frequently used in query implementations.

Remember that the result of a standard Node Query will be a list of keys for instances of the BO node that the query is assigned to. In case of a Generic Result Query the result will return the data in a table having a predefined result structure.

The enhancement mechanism is based on the so called Query Enhancement Table (*QET*) `/SCMTMS/C_QENH`. At runtime the super class creates an optimized database `SELECT` statement from the content of the query structure and the query enhancement table. The Query Enhancement Table has the following structure:

Attribute		Description
MANDT	Filled automatically	Client.
QUERY_CAT	Mandatory	Value "space" = Standard Node Query, "G" = Generic Result Query (Custom Query).
BO_NAME	Mandatory	Name of the Business Object.
NODE_NAME	Mandatory	Name of the Business Object node.
QUERY_NAME	Mandatory	Name of the query assigned to the specified BO node. The first five components of the table identify the query to be enhanced.
QUERY_ATTRIBUTE	Mandatory	Name of the new query attribute.
ATTR_NODE_NAME	Mandatory	Name of the node to which the additional query attribute belongs to.  If an external BO node or a database table (or database view) is used as the source of the additional query attribute, this field contains the name of the query BO node which contains the attribute used for the JOIN between the query BO and the external BO node or database table ("source node key of the Cross BO Association").
NODE_ATTRIBUTE	Mandatory	Name of the additional query attribute as it is named on its node.
EXT_BO_NAME	Optional	Name of Target Business Object in Cross BO Association
EXT_NODE_NAME	Optional	Name of Node Containing Cross BO Association: Together with EXT_BO_NAME this component

		describes the node of an external BO to which the additional query attribute belongs to.
EXT_DB_NAME	Optional	Table Name: Instead of an external BO node it is also possible to define a database table (or view) directly, to which the additional query attribute belongs to.
EXT_JOIN_ATTR	Optional	External Join Attribute (on Cross BO Assoc. Node or DB Table): This component is needed if an external BO node or a database table (or view) contains the additional query attribute. It specifies the attribute within this external BO node or database table (or view) which is used to join the ATTR_NODE_NAME node and the external BO node or database table (or view) together.
NODE_JOIN_ATTR	Optional	Source Join Attribute (on Cross BO Assoc. Node of Query BO). This component is needed if an external BO node or a database table (or view) contains the additional query attribute. It specifies the attribute of the ATTR_NODE_NAME node of the query BO which is used to join the ATTR_NODE_NAME node and the external BO node or database table (or view) together.
NODE_JOIN_ATTR2	Optional	Target Join Attribute (on Target DB Table, such as LANGU).
ND_JN_ATTR2_VAL	Optional	Target Node Join Attribute Value.
ND_JN_ATTR2_C	Optional	Target Node Join Attribute Value from Constant or Type.

### 6.1.2 Maintaining the standard query enhancement table

You can use report `/SCMTMS/MAINT_QUERY_ENH` for specifying the required entries in the Query Enhancement Table (QET) or start transaction `/SCMTMS/QUERY_ENH` which starts the same report. It allows specifying entries depending on the use case and the type of enhancement that you want to realize.

The screenshot shows the 'Enhance TM Queries' report selection screen. It has two main sections: 'Type of Query Enhancement' and 'Query Enhancement Category'. In the first section, 'Query Structure' is selected. In the second section, 'Attribute of Same BO' is selected. Three numbered callouts are present: 1) points to the 'Type of Query Enhancement' section, 2) points to the 'Query Enhancement Category' section, and 3) points to the information icon in the top left corner.

**Enhance TM Queries**

3) Press F8 (or the info button to display further documentation). The corresponding maintenance view for the QET will be displayed where the required entries can be specified.

Type of Query Enhancement

☒ Query Structure

☐ Result Structure

1) Choose the type of Query Enhancement.

Query Enhancement Category

☒ Attribute of Same BO

☐ Attribute of Another BO

☐ Attribute of a Database Table

2) Choose the Query Enhancement Category.

Picture: Report for QET maintenance.

The following list shows the specific QET views available for each possible use case. The options available on the selection screen of the report allow specifying the enhancement use case. Press F8 to get to the corresponding maintenance view and specify the required entries for the query enhancement.



View	Function / Use Case
/SCMTMS/V_QENH1	View for Query Extensions - additional attributes
/SCMTMS/V_QENH2	View for Query Extensions - additional attributes XBO
/SCMTMS/V_QENH3	View for Query Extensions - additional attributes from table
/SCMTMS/V_QENH4	View for Generic Result Query Extensions - additional attributes
/SCMTMS/V_QENH5	View for Generic Result Query Extensions - additional attributes XBO
/SCMTMS/V_QENH6	View for Generic Result Query Extensions - additional attributes from table.

Before we take a look at some discrete examples for standard query enhancements some general remarks on how to use the report to maintain query extensions:

#### 1) Enhancing Node Attribute Queries:

The data type (i.e. the structure providing the selection criteria) of the most simple Node Attribute Query is simply the node data structure of the node that the query is assigned to. There is no implementing class involved and the BOPF Framework handles such queries automatically. Alternatively the data type is a separate structure that represents a subset of the node data structure.

- You don't have to specify any entry in the QET to enhance such queries.
- If the data type corresponds to the node data structure (i.e. the structure with exactly the same technical name is used to define the query data type), you can simply add extension fields to the node data structure as described in section 3.3.4. The extension fields will then automatically be part of the data type and therefore also available as search criterion for the query.
- If the data type is a separate structure representing a subset of the node data structure, you need to add the extension field not only on the node (see section 3.3.4) but also add it to the separate structure used for the data type of the query.
- In both cases the extension fields will be part of the node that the query is assigned to and therefore will also be returned in the query result when the data is requested along with the BO node instance keys that match the search criteria.
- An example for such a query is e.g. *ROOT\_ELEMENTS* on the *TOR* Root Node. When you have placed enhancement fields at this node, the mentioned Node Attribute Query will directly contain these fields as search criteria.

#### 2) Enhancing Custom Queries:

A standard Custom Query is always related to a specific node of the corresponding business object. It has a data type representing the search criteria of the query as well as an implementing class. The set of search criteria is not restricted to attributes of the node that it is assigned to, i.e. the search criteria can come from another node of the same BO, from a node of another BO or even from a completely different database table.

If such Queries do not have a specific result type and result table type (assumed in this case) they just return the node instance keys of those records matching the search criteria specified in the request structure.

- In the section *Type of Query Enhancement*, you only need to select option Query Structure to define/declare additional attributes that shall serve as selection criteria.
- Chose a corresponding Query Enhancement Category and press *F8*.

- The corresponding view of the *QET* will open and allow specifying entries for the additional selection attributes. Specify values in all required fields and save the new entry.
- In this use case you just need to enhance the query data structure by your additional selection attributes as we do not have an explicit result type involved but only return keys & data of the node that the query is assigned to.

**Change View "View for Query Extensions - additional attributes": Detail**

New Entries

BO Name	/SCMTMS/TOR
Query Node Name	ROOT
Query Name	FO_DATA_BY_ATTR
Attribute Node	ZENH_ROOT_SUBNODE

The query to be enhanced.  
Example: A query from the TOR Root

View for Query Extensions - additional attributes

Add. Target Join Attr.	
Add.Trgt.Nd.Join Value	
Add.Trgt.Nd.Join Const.	
Node Attribute	ZENH_CHNG_DATE
Query Attribute	ZENH_CHNG_DATE

Example: A BO node extension field is added as an additional attribute for the query structure.

Picture: An example entry for the *QET* View /SCMTMS/V\_QENH1.

- Note that you do not need such an entry for enhancement fields which are located on the same node that the query is assigned to. They will be implicitly available as selection criteria.
- An example for such a query is e.g. *PLANNING\_ATTRIBUTES* which is assigned to the *TOR* Root Node. In the picture above you can see an example entry in the *QET* that enhances the query by an additional selection attribute *ZENH\_CHNG\_DATE* which located on an Enhancement Node *ZENH\_ROOT\_SUBNODE* (see section 3.3.5 how to create this example enhancement node and section 5.4.3 how to add this subnode to the UI as a List UIBB).

In field *Node Attribute* the field *ZENH\_CHNG\_DATE* is specified. This field will be available in the query search structure with the attribute name specified in field *Query Attribute* (here it is the same attribute name *ZENH\_CHNG\_DATE*). At runtime, the *Query attribute* (search criteria) will be mapped onto the *Node Attribute* (attribute to be searched on).

- 3) Enhancing Generic Result Queries: A Generic Result Query (e.g. used for *POWLs*) returns its results as a table with a specific result structure, i.e. it can return more than just data of the node that it is assigned to. Just like for Custom Queries, the set of search criteria is not restricted to attributes of the node that the Generic Result Query is assigned to, i.e. the search criteria can come from another node of the same BO, from a node of another BO or even from a completely different database table. In the first step we enhance the query/filter structure, i.e. the list of selection criteria as follows:

- In the section *Type of Query Enhancement*, you first need to select option *Query Structure* to define/declare additional attributes that shall serve as selection criteria.
- Chose a corresponding Query Enhancement Category and hit F8.
- The corresponding view of the *QET* will open. Here you can maintain entries for the additional selection attributes. Maintain all required fields and save the new entry.

- Finally, you need to add the additional selection attributes in the corresponding *DDIC* object (see example 2 in section 6.1.5). This is done by defining an Append for the filter structure with the new field included.

In case of Generic Result Queries being used e.g. for the POWL selection, also the origin of the attributes of the result table needs to be specified. This is done the same way as shown for the attributes of the query/filter structure:

- In the section *Type of Query Enhancement*, you first need to select option *Result Structure* to define/declare additional attributes that shall be returned as additional result attributes.
- Chose a corresponding Query Enhancement Category and hit *F8*.
- The corresponding view of the *QET* will open. Here you can maintain entries for the additional result attributes. Maintain all required fields and save the new entry.
- Finally, you need to add the additional result attributes in the corresponding *DDIC* object for the structure of the result table (see example 2 in section 6.1.5). This is done by defining an Append for the result structure with the new fields included.

The following must be kept in mind when enhancing such queries and especially when creating and implementing your very own Generic Result Queries from scratch:

- Assumption is, that all attributes of the result table structure for which no origin and mapping was defined, belong to the query node (i.e. the node that the query is assigned to).
- The result table structure must contain attribute *DB\_KEY* (corresponding to the *DB\_KEY* attribute of the query node database table) as the leading (i.e. the very first!) attribute. This is required to enable finding the link between the query node instances and the found result records.
- The result table structure should moreover contain all attributes that are relevant for authority checks. This helps quite a bit to get the query optimized from a performance perspective. Such a query first of all selects all data that can be found with the given selection criteria. Then the authority check is executed which reduces the initial result. Note that any attribute which is relevant for authority check and not part of the result table structure causes additional effort for reading/determining this data and checking it. Assuming you have provided a maximum number of rows to be returned by the query, the result will contain this number of records with only authorized data, i.e. the query will stop selection when it found the maximum number of authorized data records or earlier when there aren't that many.
- If for the result table only the mandatory attribute *DB\_KEY* was defined, it is assumed that method *GET\_RESULT\_DATA* was overwritten. In this case only the *DB\_KEY* attribute will be selected from database per default. All further attributes have to be selected within the *GET\_RESULT\_DATA* method (with its corresponding alternative implementation).
- In general, it is always allowed and possible to overwrite the *PROTECTED* methods of the query super class to realize individual query logic. Nevertheless it is important to keep the query extensible. This can be accomplished if additional attributes are always defined within the query and result enhancement tables. The best place to do this is within method *GET\_QUERY\_ENHANCE\_TABLE*. But as this is not always possible or feasible, there are further methods called during generating the query *SELECT* statement at runtime:

- *EXTEND\_SELECT\_CLAUSES*: This method is called right before the database *SELECT* statement is executed. It provides the possibility to extend or modify all parts of the *SELECT* statement to realize any kind of possible selection on the TM backend.
- *EXTEND\_RESULT\_DATA*: This method is called after the query was completely executed and the result data has been collected following the definitions from the result enhancement table. It provides the possibility to select further data to extend the result structure or to modify the result table in any way.
- *SPLIT\_SELECTION\_PARAMETERS*: This method is called before the real query execution starts. It can be used to take parts of the selection parameters out into an extra selection parameters table which shall not be taken into account by the standard query. The selection parameters from the extra table can be processed in method *POST\_KEY\_FILTERING*.
- *POST\_KEY\_FILTERING*: This method is called after the database selection of the query has been done. It can be used to filter the selected instance keys before they are returned to the consumer. It is called with the selection parameters that were processed by the query so far, and with the table of extra selection parameters.

### 6.1.3 BAdI for creation of query enhancement table entries

Instead of using the mentioned report to provide entries for the query enhancement table, these entries can also get provided by an implementation of BAdI */SCMTMS/FRW\_QUERY*. While the report allows creation of the entries without coding, the use of the BAdI requires implementation of corresponding code that provides additional entries.

The BAdI allows a programmatic provisioning of Query Enhancement information. Customers and partners can implement the BAdI to provide corresponding information. With an implementation you can e.g. use other tables than the standard *QET* (e.g. your own customer specific tables) that contain the enhancement data. The default implementation of the BAdI takes the information from the mentioned table */SCMTMS/C\_QENH* whose records were created with report */SCMTMS/MAINT\_QUERY\_ENH* (i.e. this is the “standard” enhancement data source).

### 6.1.4 Example 1: Enhancing a Custom Query

The following first example shows how to enhance Custom Query *PLANNING\_ATTRIBUTES* defined for the Root node of the Freight Order BO (*TOR*). In this example the query shall be enhanced by a field *ZENH\_CHNG\_DATE* that is part of an enhancement node *ZENH\_ROOT\_SUBNODE*.

See section 3.3.5 how to create this example enhancement node and section 5.4.3 how to add this subnode to the UI as a List UIBB. Moreover, the example includes the example enhancement fields added by the example in section 3.3.4. The related exercises are prerequisite for creating example data and making the following query enhancement example work. Of course you can also try all this out in a similar way with other already existing standard nodes and fields.

- 1) Start report */SCMTMS/MAINT\_QUERY\_ENH*: Choose Type of Query Enhancement *Query Structure*, Query Enhancement Category *Attribute of the same BO* and press F8.
- 2) On the next screen you can see the corresponding maintenance view (in this case it is */SCMTMS/V\_QENH1*). Click on button *New Entries* and enter the required data for the query extension field:

Field	Value	Comment
BO Name	/SCMTMS/TOR	The technical name for the Freight Order BO.
Query Node Name	ROOT	The node of the BO that the query to be enhanced is assigned to.
Query Name	PLANNING_ATTRIBUTES	The name of the query.
Attribute Node	ZENH_ROOT_SUBNODE	The node where the extension field can be found.
Node Attribute	ZENH_CHNG_DATE	The name of the extension field which will be available as a selection criterion.
Query Attribute	ZENH_CHNG_DATE	The name of the extension attribute as it shall be used in the query. This name can be different from the Node Attribute.

- 3) Save the new entry. The standard query has now an additional query attribute that can be used.
- 4) Test the enhanced query. Note that the query enhancements cannot be tested via the BOBF Test tool */BOBF/TEST\_UI*. As the query enhancement concept is not part of the BOBF framework itself, any query enhancements will not appear in the list of selection fields when testing with the test tool. Instead, the query can be tested with a simple test report that executes the query with the new query attribute. Code example:

```

*&-----
*& Report ZREP_CUSTOM_QUERY_TEST
*&-----
*& Demonstration of a standard query enhancement
*&-----
REPORT zrep_custom_query_test.

DATA: lo_srv          TYPE REF TO /bobf/if_tra_service_manager,
      lt_selpar       TYPE /bobf/t_frw_query_selparam,
      ls_selpar       TYPE /bobf/s_frw_query_selparam,
      lo_message      TYPE REF TO /bobf/if_frw_message,
      lt_data         TYPE /scmtms/t_tor_root_k,
      ls_key          TYPE /bobf/s_frw_key,
      lt_key          TYPE /bobf/t_frw_key,
      ls_query_inf    TYPE /bobf/s_frw_query_info.

CLEAR: ls_selpar,
       lt_selpar.

* Get instance of service manager for TRQ
lo_srv = /bobf/cl_tra_serv_mgr_factory=>get_service_manager(
        /scmtms/if_tor_c=>sc_bo_key ).

*Get instances of TOR via Query
ls_selpar-attribute_name = /scmtms/if_tor_c=>sc_query_attribute-
                           root-planning_attributes-changed_by.

ls_selpar-option         = 'EQ'.
ls_selpar-sign           = 'I'.
ls_selpar-low            = 'SCMSUPPORT'.
APPEND ls_selpar TO lt_selpar.

ls_selpar-attribute_name = 'ZENH_ENTRY_DATE'.
ls_selpar-option         = 'EQ'.
ls_selpar-sign           = 'I'.

```

```

ls_selpar-low           = '20140610072530'.
APPEND ls_selpar TO lt_selpar.

ls_selpar-attribute_name = 'ZENH_CHNG_DATE'.
ls_selpar-option         = 'EQ'.
ls_selpar-sign           = 'I'.
ls_selpar-low            = '20140610072530'.
APPEND ls_selpar TO lt_selpar.

BREAK-POINT.

* Execute the query
lo_srv->query(
  EXPORTING
    iv_query_key          = /scmtms/if_tor_c=>sc_query-root-
                          planning_attributes " Query
    it_selection_parameters = lt_selpar " Query Sel.Parameters
    iv_fill_data          = abap_true
  IMPORTING
    eo_message            = lo_message " Message Object
    es_query_info         = ls_query_inf " Query Information
    et_data              = lt_data
    et_key               = lt_key ).

BREAK-POINT.

```

The query in this example should return just the keys of those instances of BO /SCMTMS/TOR which were changed by user SCMSUPPORT and ZENH\_ENTRY\_DATE equal to 10.06.2014 07:25:30 (represented in the coding above as timestamp 20140610072530) and have an existing data record on the enhancement node ZENH\_ROOT\_SUBNODE, attribute ZENH\_CHNG\_DATA equal to 10.06.2014 07:25:30 (also represented as time stamp in the above coding).

The extension field ZENH\_ENTRY\_DATA was added to the Root Node (see section 3.3.4), i.e. to the same node that the query is assigned to. Therefore it does not need an explicit QET entry. But field ZENH\_CHNG\_DATA is located on node ZENH\_ROOT\_SUBNODE and therefore needs a corresponding entry.

Of course you could e.g. also specify a range for the new extension fields to be used for the query to e.g. search for all instances within a given time frame. In general, all selection parameter options can be used for an enhancement selection field just like for any other standard selection field.

### 6.1.5 Example 2: Enhancing a Generic Result Query

In the second example the generic result query FO\_DATA\_BY\_ATTR is extended. Again this query is defined and assigned for the Root node of the Freight Order BO. Again the extension fields ZENH\_ENTRY\_DATE of the Root node and field ZENH\_CHNG\_DATA of the enhancement node ZENH\_ROOT\_SUBNODE shall be added to the query as a selection criterion. Both attributes shall also be returned in the result table.

- 1) Start report /SCMTMS/MAINT\_QUERY\_ENH: Choose Type of Query Enhancement *Query Structure*, Query Enhancement Category *Attribute of the same BO* and press F8.
- 2) On the next screen you can see the corresponding maintenance view (in this case it is /SCMTMS/V\_QENH1). Click on button *New Entries* and enter the required data for the query extension field:



Field	Value	Comment
BO Name	/SCMTMS/TOR	The technical name for the Freight Order BO.
Query Node Name	ROOT	The node of the BO that the query to be enhanced is assigned to.
Query Name	FO_DATA_BY_ATTR	The name of the query.
Attribute Node	ZENH_ROOT_SUBNODE	The node where the extension field can be found.
Node Attribute	ZENH_CHNG_DATE	The name of the extension field which will be available as a selection criterion.
Query Attribute	ZENH_CHNG_DATE	The name of the extension attribute as it shall be used in the query. This name can be different from the Node Attribute.

Note that such an entry is not required for the field `ZENH_ENTRY_DATE` as this is defined at the Root Node of BO `/SCMTMS/TOR`, i.e. the node that also the query is assigned to.

With this entry in the Query Enhancement Table, the implementing class of the query (class `/SCMTMS/CL_TOR_Q_FO`, inheriting from the mentioned query super class) is enabled to consider the enhancement field `ZENH_CHNG_DATE` of enhancement node `ZENH_ROOT_SUBNODE` for building the dynamic select statement at runtime. But to prevent misunderstandings, this does not automatically also enhance the query's request structure in the DDIC. This is required and done in step 4.

- 3) Go back to the initial screen of report `/SCMTMS/MAINT_QUERY_ENH`: Choose Type of Query Enhancement *Result Structure*, Query Enhancement Category *Attribute of the same BO* and press F8.

On the next screen you can see the corresponding maintenance view (in this case it is `/SCMTMS/V_QENH4`). Click on button *New Entries* and enter the required data for the query extension field:

Field	Value	Comment
BO Name	/SCMTMS/TOR	The technical name for the Freight Order BO.
Query Node Name	ROOT	The node of the BO that the query to be enhanced is assigned to.
Query Name	FO_DATA_BY_ATTR	The name of the query.
Attribute Node	ZENH_ROOT_SUBNODE	The node where the extension field can be found.
Node Attribute	ZENH_CHNG_DATE	The name of the extension field which will be available as a selection criterion.
Query Attribute	ZENH_CHNG_DATE	The name of the extension attribute as it shall be used in the query. This name can be different from the Node Attribute.

Again, such an entry is not required for the field `ZENH_ENTRY_DATE` as this is defined at the Root Node of BO `/SCMTMS/TOR`, i.e. the node that also the query is assigned to.

With this entry in the Query Enhancement Table, the implementing class of the query is enabled to return the content of enhancement field `ZENH_CHNG_DATE` of enhancement node `ZENH_ROOT_SUBNODE` as an attribute of the result structure. Again, this does not automatically also enhance the query's result structure in the DDIC. This is required and done in step 5.



- 4) The Filter Structure of the query is defined by DDIC structure `/SCMTMS/S_TOR_Q_FO` and represents the set of selection criteria that can be used with the query. This structure does not correspond to a Node Structure but is in this case an arbitrary structure. So in the context of Generic Result Queries we have to enhance this structure as well with the additional fields (i.e. both enhancement fields need to be added here).

In transaction `/BOBF/CONF_UI` navigate to business object `/SCMTMS/TOR` (Freight Order) and then navigate to the query `FO_DATA_BY_ATTR` under the node elements of the Root node. Double click on the query to display its details. Now double click on the displayed Filter Structure `/SCMTMS/S_TOR_Q_FO` of the query and create an append structure with the following information:

Append Structure	ZENH_TOR_Q_SEARCH
Description	Gen. Result Query Enh. Selection Criteria

Add the following component to the append structure.

Component	Typing Method	Component Type
ZENH_ENTRY_DATE	Types	/SCMTMS/DATETIME
ZENH_CHNG_DATE	Types	/SCMTMS/DATETIME

Save and activate the append structure.

- 5) The next step is to enhance the query result structure with the additional fields that shall be returned by the query. Again, a `QET` entry is only required for the attribute `ZENH_CHNG_DATE` as it is located on a different node than the node that the query is assigned to.

In transaction `/BOBF/CONF_UI` navigate to business object `/SCMTMS/TOR` (Freight Order) and then navigate to the query `FO_DATA_BY_ATTR` under the node elements of the Root node. Double click on the query to display its details. Now double click on the displayed Result Type `/SCMTMS/S_TOR_Q_FO_R` of the query and create an append structure with the following information:

Append Structure	ZENH_TOR_Q_RESULT
Description	Gen. Result Query Enh. Result Attributes

Add the following components to the append structure.

Component	Typing Method	Component Type
ZENH_ENTRY_DATE	Types	/SCMTMS/DATETIME
ZENH_CHNG_DATE	Types	/SCMTMS/DATETIME

Note that again both enhancement fields are added, this time in the DDIC structure that defines the result structure of the query.

Save and activate the append structure.

- 6) Test the enhanced query. As mentioned, the query enhancements cannot be tested via the BOBF Test tool `/BOBF/TEST_UI` as the query enhancement concept is not part of the BOBF framework itself. Any query enhancement will not appear in the list of selection fields when testing with the test tool. Instead the query can be tested with a simple report executing the query with the new query attributes. Code example:

```

*&-----*
*& Report   ZREP_GENRES_QUERY_TEST
*&-----*
*& Demonstartion of a generic result query enhancement
*&-----*
REPORT   zrep_genres_query_test.

DATA:    lo_srv          TYPE REF TO /bobf/if_tra_service_manager,
         lt_selpar       TYPE /bobf/t_frw_query_selparam,
         ls_selpar       TYPE /bobf/s_frw_query_selparam,
         lt_req_attr     TYPE /bobf/t_frw_name,
         lo_message      TYPE REF TO /bobf/if_frw_message,
         lt_data         TYPE /scmtms/t_tor_q_fo_r,
         ls_key          TYPE /bobf/s_frw_key,
         lt_key          TYPE /bobf/t_frw_key,
         ls_query_inf    TYPE /bobf/s_frw_query_info.

CLEAR:   ls_selpar,
         lt_selpar,
         lt_req_attr.

* Get instance of service manager for TRQ
lo_srv = /bobf/cl_tra_serv_mgr_factory=>get_service_manager(
                                         /scmtms/if_tor_c=>sc_bo_key ).

*Get instances of TOR via Query
ls_selpar-attribute_name = /scmtms/if_tor_c=>sc_query_attribute-
                           root-fo_data_by_attr-changed_by.

ls_selpar-option          = 'EQ'.
ls_selpar-sign            = 'I'.
ls_selpar-low             = 'SCMSUPPORT'.
APPEND ls_selpar TO lt_selpar.
APPEND ls_selpar-attribute_name TO lt_req_attr.

ls_selpar-attribute_name = 'ZENH_ENTRY_DATE'.
ls_selpar-option          = 'EQ'.
ls_selpar-sign            = 'I'.
ls_selpar-low             = '20140610133000'.
APPEND ls_selpar TO lt_selpar.
APPEND ls_selpar-attribute_name TO lt_req_attr.

ls_selpar-attribute_name = 'ZENH_CHNG_DATE'.
ls_selpar-option          = 'EQ'.
ls_selpar-sign            = 'I'.
ls_selpar-low             = '20140610141800'.
APPEND ls_selpar TO lt_selpar.
APPEND ls_selpar-attribute_name TO lt_req_attr.

BREAK-POINT.

* Execute the query
lo_srv->query(
  EXPORTING
    iv_query_key          = /scmtms/if_tor_c=>sc_query-root-
                           fo_data_by_attr " Query
*   it_filter_key         = " Key Table
  it_selection_parameters = lt_selpar " Query Sel.      Parameters
*   is_query_options      = " Query Options
  iv_fill_data            = abap_true
  it_requested_attributes = lt_req_attr " List of Names (e.g. Fi
                                         " Fieldnames)

```

**IMPORTING**

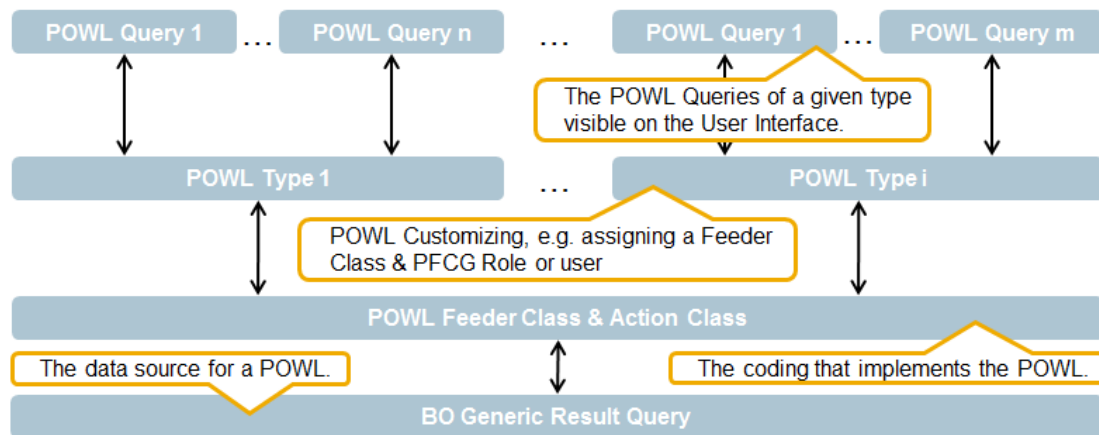
```
eo_message      = lo_message  " Message Object
es_query_info   = ls_query_inf" Query Information
et_data         = lt_data
et_key          = lt_key ).
```

**BREAK-POINT.**

Remark: The example query *FO\_DATA\_BY\_ATTR* is implemented in a way that parameter *IT\_REQUESTED\_ATTRIBUTES* of the *QUERY* method must be provided with all attributes that the query shall return. This is done for performance reasons, i.e. it allows reading only the information that is really required without unnecessary overhead. You can see in the example code how this parameter (*LT\_REQ\_ATTR*) is build up while defining the selection criteria that will be passed to the query. The mentioned query is e.g. used as the data provider for the Freight Order POWL. When called in this context, it will only return those fields of the POWL result structure that are configured to be visible. This helps to reduce the runtime.

## 6.2 POWL (Personal Object Work Lists)

This section describes how to create a new TM specific Personal Object Work List (POWL) and how to enhance existing standard POWLs provided with the standard TM application. In section 6.1 we have already seen how Generic Result Queries can be enhanced which are the data source for any POWL. As a first step we will take a look at how to create a new POWL. This allows exploring all involved technical concepts and components which are also relevant for enhancing existing POWLs.



Picture: POWL components and their relation.

The picture above shows an overview of the different POWL components and their relation. You will get to know all of these components and their usage in the following sections.

Note that there is actually no explicit enhancement concept available for existing standard POWLs. In this case you will have to work with implicit enhancements (see section 4.5) to add required elements to a standard POWL. Creating a new POWL as a first example provides useful hints at which places standard POWLs have to be adjusted to get enhancements done.

### 6.2.1 Creating a new POWL

As mentioned we will first of all create a new POWL to get to know the most important elements and basic technical concepts of a POWL.

The following example will use the Generic Result Query *FO\_DATA\_BY\_ATTR* as data source which was already enhanced by the additional fields *ZENH\_ENTRY\_DATE* and *ZENH\_CHNG\_DATE* in section 6.1.5. In general all POWLs use a Generic Result Query as data source. Besides the standard selection criteria and result attributes, the new example POWL will also allow using the enhancement fields as selection criteria and result attributes.

### 6.2.2 The POWL Feeder Class

The main access point for a POWL is the *POWL Feeder Class*. It contains the definition of the POWL's selection criteria, the field catalog (i.e. the result structure) and the actions that can be executed from the POWL Toolbar for a selected set of object instances from the POWL result list.

In TM a *POWL Feeder Class* is usually based on a query of a Business Object. The relationship between a Business Object query (a Generic Result Query) and a *POWL Feeder Class* is always one to one. So whenever there is no *POWL Feeder Class* making use of an existing Generic Result Query of a BO you will need a new *POWL Feeder Class*.

You should also keep this in mind when creating BO Queries for POWL usage. If you have two completely different requirements leading to different BO Queries, of course two different *POWL Feeder Classes* are required.

If you have different business categories or usages for POWLs which share many common parts from a technical perspective, it makes sense to design one (technical) BO query which has one common technical POWL Feeder Class in the end. Based on this feeder class, you can separate your different usages or categories with corresponding POWL types. An example is the TOR Feeder class `/SCMTMS/CL_UI_POW_FD_TOR`, with separate POWL types for each BO Category. This example POWL Feeder Class provides selection criteria, result attributes and actions depending on the TOR Category, e.g. TO for Freight Order or FU for Freight Units.

The following example implements a new POWL for Freight Orders.

- 1) Create a new POWL Feeder Class with e.g. transaction `SE24`: The general naming convention followed in TM is `/SCMTMS/CL_UI_POW_FD_[object name or abbreviation]`.

Example class: `ZCL_ENH_UI_POW_FD_TOR`.

Class `/SCMTMS/CL_UI_POW_FD_BASE` must be defined as the super class for the POWL Feeder Class. Save and activate the new class.

- 2) In the POWL Feeder Class constructor constants are used to specify the names and keys of all general attributes and elements of a POWL. In the standard TM POWL implementations these constants are usually defined in the standard constants interface `/SCMTMS/IF_UI_POW_CONST`.

In our example we create a separate new constants interface to represent the constants required for the new POWL. With this separate constants interface it is not required using implicit enhancements for adjusting the standard constants interface with your own constants. Create the example constants interface as follows:

- Start transaction `SE24`, enter the interface name `ZIF_ENH_UI_POW_CONST` in field *Object Type* and click on button *Create*.
- On tab strip *Attributes* enter the following constants:

Attribute	Level	Description
CO_OUTPUT_STRUCTURE_NAME	Constant	Output list Structure Names
CO_SELCRIT_STRUCTURE_NAME	Constant	Selection Criteria Structure names
CO_ACTION_CLASS_NAME	Constant	Action class names
CO_POWL_TYPE	Constant	POWL Types

- Switch to the *Source Code-based view* (toggle button in the Class Builder, transaction `SE24` tool bar) and make sure that the example constants interface is defined by the following code:

```

*-----*
*  INTERFACE ZIF_ENH_UI_POW_CONST
*-----*
*  Constants for the Demo Enhancement POWL
*-----*
INTERFACE zif_enh_ui_pow_const

PUBLIC .

CONSTANTS:
    BEGIN OF co_output_structure_name,
        zc_tor_resp TYPE struname VALUE 'ZENH_S_UI_POW_R_TOR',
    END OF co_output_structure_name .
CONSTANTS:
    BEGIN OF co_selcrit_structure_name,
```

```

        zc_tor_req TYPE struname VALUE 'ZENH_S_UI_POW_S_TOR',
    END OF co_selcrit_structure_name .
CONSTANTS:
    BEGIN OF co_action_class_name,
        zc_tor_act TYPE seoclsname VALUE 'ZCL_ENH_UI_ACTION_TOR',
    END OF co_action_class_name .
CONSTANTS:
    BEGIN OF co_powl_type,
        BEGIN OF enh_type,
            zenh_tor TYPE powl_type_ty VALUE 'ZENH_TOR_POWL',
        END OF enh_type,
    END OF co_powl_type.

ENDINTERFACE.                                "ZIF_ENH_UI_POW_CONST

```

- 3) In the next step copy the following required structures from the standard TOR POWL and create a new action class. This will help a bit reducing the effort and time for creating the example. For your own implementations you can of course create and use your very own corresponding objects.

- **Selection Criteria:** The structure will represent the Search Structure for the new POWL, i.e. it contains the list of attributes that will be available as selection criteria for the POWL queries.
  - Start transaction *SE11* and enter */SCMTMS/S\_UI\_POW\_S\_TOR* in field *Data type* on the initial screen.
  - Copy (*Ctrl+F5*) this structure to *ZENH\_S\_UI\_POW\_S\_TOR*.
  - Save and activate the new structure.
- **Result Structure:** The structure will represent the Result Structure for the new POWL, i.e. it contains the list of attributes that will be available to be displayed in the result list of the POWL.
  - Start transaction *SE11* and enter */SCMTMS/S\_UI\_POW\_R\_TOR* in field *Data type* on the initial screen.
  - Copy (*Ctrl+F5*) this structure to *ZENH\_R\_UI\_POW\_R\_TOR*.
  - Save and activate the new structure.
- **Actions:** The class will represent the Action Class for the new POWL, i.e. it contains the coding to handle all actions that can be executed on entries selected in the POWL result list.
  - Start transaction *SE24* and enter class name *ZCL\_ENH\_UI\_ACTION\_TOR* in field *Object type* and click on button *Create (F5)* to create this class.
  - On tab *Properties* specify class */SCMTMS/CL\_UI\_ACTION\_BASE* as the super class for the new class.
  - Save and activate the new class.

Alternative: Copy standard action class */SCMTMS/CL\_UI\_ACTION\_TOR* to the new class *ZCL\_ENH\_UI\_ACTION\_TOR*. With this procedure you simply reuse the action handling as implemented in the standard. In section 6.2.3 you can find an implementation example that shows how to implement the action class yourself.

- 4) Go back to the POWL Feeder Class *ZCL\_ENH\_UI\_POW\_FD\_TOR* and create a method *CONSTRUCTOR* as a public instance method. In the implementation of this POWL Feeder Class constructor, the following information is defined and provided:

Attribute	Description
BO_NAME	The business object that is to be processed with the POWL.
BO_QUERY_KEY	The key of the business object query that will provide the data for the POWL.
BO_ALTID_ATTRIBUTE_NAME	Describes the attribute name of the result structure which is the alternative identifier of the business object. This attribute will be displayed as the first column in the POWL result list. Moreover, this column is fixed and will be displayed as a link (for further object based navigation).
POW_OUTPUT_STRUCTURE_NAME	Represents the output structure for the field catalog (based on the result structure of the output query or the data structure of the root node). All Standard POWL result structures start with /SCMTMS/S_UI_POW_R*
POW_SELCRIT_STRUCTURE_NAME	Represents the selection criteria structure for the selection criteria catalog (based on the query structure of the BO query). All Standard POWL query structures start with /SCMTMS/S_UI_POW_S*
ACTION_CLASS_NAME	(Optional) The class can be specified in case you want to use your very own action class to handle the actions on the POWL.
BO_CATEGORY_ATTRIBUTE_NAME	(Optional) Describes the attribute name of the result structure which contains a BO category (e.g. required for TRQ and TOR as these objects are used for different purposes represented by a corresponding category → serves as a “filter”).
BO_KEY_ATTRIBUTE_NAME	Describes the attribute name of the result structure which is used for generic navigation. This starts after you clicked on the generic link which is based on <b>BO_ALTID_ATTRIBUTE_NAME</b> .
CONVERSION_CLASS_NAME	(Optional) The class can be specified in case you want to use your very own conversion class.
MV_REQUESTED_ATTRIBUTES	Set to <i>true</i> in the example to indicate that only requested attributes (i.e. the attributes of the defined field catalog are requested and read from the database. This helps improving the performance.

Use the following example code to implement the constructor method. Then save and activate the class. Within the coding you can see that it makes use of the constants defined in step 2 to specify the above listed attributes of the feeder class.

**METHOD** constructor.

```
* call the constructor of the super class
CALL METHOD super->constructor.

* define BO to be represented by the POWL
ms_pow_profile-bo_name = /scmtms/if_tor_c=>sc_bo_name.

* define the Generic Result Query to be used with the POWL
ms_pow_profile-bo_query_key = /scmtms/if_tor_c=>sc_query-root-fo_data_by_attr.
```



```

* define the attribute of the BO to represent the alternative
* id/key on the POWL
ms_pow_profile-bo_altid_attribute_name = /scmtms/if_tor_c=>
                                         sc_node_attribute-root-tor_id.

* define the attribute to represent the category of BO instances
* Use Case e.g.: Only TOR instances of category Booking shall be
* listed
ms_pow_profile-bo_category_attribute_name = /scmtms/if_tor_c=>
                                         sc_node_attribute-root-tor_cat.

* define the DDIC structure that represents the set of attributes
* to be listed in the POWL result list
ms_pow_profile-pow_output_structure_name =
                                         zif_enh_ui_pow_const=>
                                         co_output_structure_name-zc_tor_resp.

* define the DDIC structure that represents the set of selection
* criteria attributes to be used for the POWL Query
ms_pow_profile-pow_selcrit_structure_name =
                                         zif_enh_ui_pow_const=>
                                         co_selcrit_structure_name-zc_tor_req.

* define the class that represents the action class for the POWL,
* i.e. the operations that can be triggered from the POWL for
* selected Business Document in the POWL result list
ms_pow_profile-action_class_name =
                                         zif_enh_ui_pow_const=>
                                         co_action_class_name-zc_tor_act.

* Conversion Class definition. Similar to an FBI View Conversion
* Class it allows the conversion of time stamps into the three
* parts date, time and time zone to be available as selection
* criteria or it allows converting codes like e.g. Life Cycle
* Status = 01 into a readable text representation, e.g. "open"
ms_pow_profile-conversion_class_name =
                                         /scmtms/if_ui_pow_const=>
                                         co_conversion_class_name-tor

* make sure that only requested attributes are passed to the query
* which helps improving performance
mv_requested_attributes = abap_true.

* Important(!): initialize the POWL
CALL METHOD init( ).

ENDMETHOD.

```

In the example code for the constructor you can see that also a conversion class was defined (class `/SCMTMS/CL_UI_POW_TOR`). In this case it is the standard TOR conversion class that is referenced / assigned via the standard POWL constants interface. For the selection criteria it supports a conversion from time stamp fields into the corresponding parts date, time and time zone similar to the conversions available in FBI Views described in section 5.2.4. Example:

- In the selection criteria structure you have a field `CHANGED_ON` defined as a time stamp.
- The conversion class will check if there are fields available in the selection criteria structure with the name `CHANGED_ON_D` (date), `CHANGED_ON_T` (time) and `CHANGED_ON_TZ` (time zone). These fields have to be placed in a

corresponding Include with group name *DATS* to indicate that these fields will be involved in a date conversion as described (the group name is used to define the type of conversion).

In the output structure, the same date conversion can be used as described above (i.e. using an Include with fields that follow the same kind of naming convention and the group name *DATS*). Moreover, for the output structure you can also use a code list conversion. Example:

- In the output structure you have a field *LIFECYCLE* which represents the life cycle status of e.g. a Freight Order with value 00 = Draft, 01 = New, etc. (the values are e.g. defined in the domain of the related data element).
  - The conversion class will check if there is a field available in the output structure with the name *LIFECYCLE\_TXT*. This field has to be placed in a corresponding Include with group name *CODL* to indicate that it will be involved in a code list conversion (again the group name is used to define the type of conversion). Instead of unclear values like 00, 01, etc. the clear text of a life cycle status will be displayed in the POWL result.
- 5) Before we can redefine and implement some of the methods of the example feeder class *ZCL\_ENH\_UI\_POW\_FD\_TOR* create the following new methods in this class. They encapsulate the coding for building the list of selection criteria and the field catalog for the POWL result list at runtime.

- Create a new method *GET\_SEL\_CRITERIA\_FO* in class with the following settings:

Method	Level	Visibility	Description
GET_SEL_CRITERIA_FO	Instance Method	Protected	Return selection criteria for category Freight Order.

Parameter	Type	Typing Method	Associated Type
C_SELCRIT_DEFS	Changing	Type	POWL_SELCRIT_TTY
C_DEFAULT_VALUES	Changing	Type	RSPARAMS_TT

Use the following example coding to implement it:

```
METHOD get_sel_criteria_fo.
```

```

DATA: ls_selcrit_defs      LIKE LINE OF c_selcrit_defs,
      ls_default_values    LIKE LINE OF c_default_values,
      ls_selcrit_ddfields  LIKE LINE OF mt_selcrit_ddfields,
      ls_pow_selcrit_mapp  LIKE LINE OF mt_pow_selcrit_mapp.

LOOP AT mt_selcrit_ddfields INTO ls_selcrit_ddfields.
*   define the general properties of each selection field
  CLEAR ls_selcrit_defs.
  ls_selcrit_defs-param_type = /scmtms/if_ui_pow_const=>
                                co_param_type-input_field.
  ls_selcrit_defs-kind = 'S'.
*   P = Parameter, S = Select Option
  ls_selcrit_defs-allow_admin_change = abap_true.
  ls_selcrit_defs-ref_table = ms_pow_profile-
                                pow_selcrit_structure_name.
  ls_selcrit_defs-ref_field = ls_selcrit_ddfields-fieldname.
  ls_selcrit_defs-crittext = ls_selcrit_ddfields-scrtext_m.
  ls_selcrit_defs-tooltip = ls_selcrit_ddfields-scrtext_l.
  ls_selcrit_defs-datatype = ls_selcrit_ddfields-rollname.
  ls_selcrit_defs-allow_admin_change = abap_true.

```

```

*   define specific properties for each field
CASE ls_selcrit_ddfields-fieldname.
*   Fields for identifying the TOR instance
WHEN /scmtms/if_tor_c=>sc_query_attribute-root-
    fo_data_by_attr-tor_id.
    ls_selcrit_defs-selname = 'P0001'.
    ls_selcrit_defs-header = cl_wd_utilities=>
    get_otr_text_by_alias( '/SCMTMS/UI_CMN/GENERAL_DATA' ).
    ls_selcrit_defs-quicksearch_crit = abap_true.
WHEN /scmtms/if_tor_c=>sc_query_attribute-root-
    fo_data_by_attr-tor_type.
    ls_selcrit_defs-selname = 'P0002'.
    ls_selcrit_defs-header = cl_wd_utilities=>
    get_otr_text_by_alias( '/SCMTMS/UI_CMN/GENERAL_DATA' ).
    ls_selcrit_defs-quicksearch_crit = abap_true.
WHEN /scmtms/if_tor_c=>sc_query_attribute-root-
    fo_data_by_attr-tor_cat.
    ls_default_values-selname = 'P0003'.
    ls_default_values-kind     = 'P'.
    ls_default_values-sign     = 'I'.
    ls_default_values-low     = /scmtms/if_tor_const=>
    sc_tor_category-active.
    INSERT ls_default_values INTO TABLE c_default_values.
    ls_selcrit_defs-read_only = abap_true.
    ls_selcrit_defs-hidden    = abap_true.
    ls_selcrit_defs-selname = 'P0003'.
    ls_selcrit_defs-header = cl_wd_utilities=>
    get_otr_text_by_alias( '/SCMTMS/UI_CMN/GENERAL_DATA' ).
    ls_selcrit_defs-param_type = /scmtms/if_ui_pow_const=>
    co_param_type-input_field.

*   Organisational Data
WHEN /scmtms/if_tor_c=>sc_query_attribute-root-
    fo_data_by_attr-exec_org_id.
    ls_selcrit_defs-selname = 'P0100'.
    ls_selcrit_defs-header = cl_wd_utilities=>
    get_otr_text_by_alias( '/SCMTMS/UI_CMN/ORGANIZATIONAL_DATA' ).
WHEN /scmtms/if_tor_c=>sc_query_attribute-root-
    fo_data_by_attr-exec_grp_id.
    ls_selcrit_defs-selname = 'P0101'.
    ls_selcrit_defs-header = cl_wd_utilities=>
    get_otr_text_by_alias( '/SCMTMS/UI_CMN/ORGANIZATIONAL_DATA' ).
WHEN /scmtms/if_tor_c=>sc_query_attribute-root-
    fo_data_by_attr-purch_org_id.
    ls_selcrit_defs-selname = 'P0102'.
    ls_selcrit_defs-header = cl_wd_utilities=>
    get_otr_text_by_alias( '/SCMTMS/UI_CMN/ORGANIZATIONAL_DATA' ).
WHEN /scmtms/if_tor_c=>sc_query_attribute-root-
    fo_data_by_attr-purch_grp_id.
    ls_selcrit_defs-selname = 'P0103'.
    ls_selcrit_defs-header = cl_wd_utilities=>
    get_otr_text_by_alias( '/SCMTMS/UI_CMN/ORGANIZATIONAL_DATA' ).

*   Admin data
WHEN /scmtms/if_tor_c=>sc_query_attribute-root-
    fo_data_by_attr-created_by.
    ls_selcrit_defs-selname = 'P2000'.
    ls_selcrit_defs-header = cl_wd_utilities=>
    get_otr_text_by_alias( '/SCMTMS/UI_CMN/ADM_INFO' ).
    ls_selcrit_defs-quicksearch_crit = abap_true.
WHEN 'CREATED_ON_D'.

```

```

        ls_selcrit_defs-selname = 'P2001'.
        ls_selcrit_defs-header = cl_wd_utilities=>
            get_otr_text_by_alias('/SCMTMS/UI_CMN/ADM_INFO').
        WHEN /scmtms/if_tor_c=>sc_query_attribute-root-
            fo_data_by_attr-changed_by.
        ls_selcrit_defs-selname = 'P2002'.
        ls_selcrit_defs-header = cl_wd_utilities=>
            get_otr_text_by_alias('/SCMTMS/UI_CMN/ADM_INFO').
        WHEN 'CHANGED_ON_D'.
        ls_selcrit_defs-selname = 'P2003'.
        ls_selcrit_defs-header = cl_wd_utilities=>
            get_otr_text_by_alias('/SCMTMS/UI_CMN/ADM_INFO').

*      Enhancement Selection Attribute
        WHEN 'ZENH_ENTRY_DATE'.
        ls_selcrit_defs-selname = 'P3003'.
        ls_selcrit_defs-header = 'Enh. Entry Date'.
        ls_selcrit_defs-quicksearch_crit = abap_true.

        WHEN 'ZENH_CHNG_DATE'.
        ls_selcrit_defs-selname = 'P3004'.
        ls_selcrit_defs-header = 'Enh. Change Date'.
        ls_selcrit_defs-quicksearch_crit = abap_true.

        WHEN OTHERS.
            CONTINUE.
        ENDCASE.

*      fill mapping table
        ls_pow_selcrit_mapp-selname = ls_selcrit_defs-selname.
        ls_pow_selcrit_mapp-fieldname = ls_selcrit_ddfields-fieldname.
        APPEND ls_pow_selcrit_mapp TO mt_pow_selcrit_mapp.
        APPEND ls_selcrit_defs TO c_selcrit_defs.
    ENDMETHOD.

*      call standard parameters - to add e.g. Max. Number of Rows
*      as an additional generic selection parameter
        add_std_sel_crits( CHANGING
            c_selcrit_defs = c_selcrit_defs
            c_default_values = c_default_values ).

ENDMETHOD.

```

The method loops over the list of all potential selection criteria from the corresponding structure defined in the constructor of the feeder class. Within this loop the *WHEN* sections of the *CASE* instruction show how to set the properties of the selection attributes. More examples on how to set such properties can be found in the standard feeder classes. Finally the method returns the set of selection criteria that can be used to define POWL queries.

The term “POWL query” is not to be mixed up with the technical Generic Result Query that represents the data source for the POWL. Here this term refers to the POWL queries that you can define on the User Interface, based on the POWL (see section 6.2.5).

- Create a new method `GET_FIELDCATALOG_FO` with the following settings:

Method	Level	Visibility	Description
GET_FIELDCATALOG_FO	Instance Method	Protected	Return output list for category Freight Order

Parameter	Type	Typing Method	Associated Type
C_FIELDCAT	Changing	Type	POWL_FIELDCAT_TTY

Use the following example coding to implement it:

```
METHOD get_fieldcatalog_fo.
  DATA: ls_output_ddfields LIKE LINE OF mt_output_ddfields,
         ls_fieldcat        LIKE LINE OF c_fieldcat.

  LOOP AT mt_output_ddfields INTO ls_output_ddfields.
    CLEAR ls_fieldcat.
    * now set default layout for all other columns
    ls_fieldcat-colid          = ls_output_ddfields-fieldname.
    ls_fieldcat-col_visible   = abap_true.
    ls_fieldcat-header_by_ddic = abap_true.
    ls_fieldcat-enabled       = abap_true.
    ls_fieldcat-allow_filter   = abap_true.
    ls_fieldcat-allow_sort     = abap_true.
    ls_fieldcat-display_type   = /scmtms/if_ui_pow_const=>
                                co_display_type-textview.

    CASE ls_output_ddfields-fieldname.
      WHEN 'TOR_ID'.
        ls_fieldcat-colpos = 1.
        ls_fieldcat-col_visible = abap_true.
        ls_fieldcat-fixed = abap_true.
        ls_fieldcat-display_type = /scmtms/if_ui_pow_const=>
                                co_display_type-link_to_action.
        ls_fieldcat-text_ref = ms_pow_profile-
                                bo_altid_attribute_name.
        ls_fieldcat-sort_order = '01'.
      WHEN 'TOR_TYPE'.
        ls_fieldcat-colpos = 2.
        ls_fieldcat-col_visible = abap_true.

    * Organizational Units
      WHEN 'PURCH_ORG_ID'.
        ls_fieldcat-colpos = 3.
        ls_fieldcat-col_visible = abap_true.
      WHEN 'PURCH_GRP_ID'.
        ls_fieldcat-colpos = 4.
        ls_fieldcat-col_visible = abap_true.
      WHEN 'EXEC_ORG_ID'.
        ls_fieldcat-colpos = 5.
        ls_fieldcat-display_type = /scmtms/if_ui_pow_const=>
                                co_display_type-dropdown_by_key.
        ls_fieldcat-col_visible = abap_true.
      WHEN 'EXEC_GRP_ID'.
        ls_fieldcat-colpos = 6.
        ls_fieldcat-display_type = /scmtms/if_ui_pow_const=>
                                co_display_type-dropdown_by_key.
        ls_fieldcat-col_visible = abap_true.

    * Administrative Data
      WHEN 'CREATED_BY'.
```

```

        ls_fieldcat-colpos = 7.
        ls_fieldcat-col_visible = abap_true.
    WHEN 'CREATED_ON'.
        ls_fieldcat-colpos = 8.
        ls_fieldcat-col_visible = abap_true.
    WHEN 'CHANGED_BY'.
        ls_fieldcat-colpos = 9.
        ls_fieldcat-col_visible = abap_true.
    WHEN 'CHANGED_ON'.
        ls_fieldcat-colpos = 10.
        ls_fieldcat-col_visible = abap_true.

*      Enhancement Fields
    WHEN 'ZENH_ENTRY_DATE'.
        ls_fieldcat-colpos = 11.
        ls_fieldcat-col_visible = abap_true.

    WHEN 'ZENH_CHNG_DATE'.
        ls_fieldcat-colpos = 12.
        ls_fieldcat-col_visible = abap_true.

    WHEN OTHERS.
        ls_fieldcat-col_visible = abap_false.

    ENDCASE.

    INSERT ls_fieldcat INTO TABLE c_fieldcat.
    ENDLOOP.

ENDMETHOD.

```

Analog to the method for the selection criteria the method loops over the list of all potential fields which can be shown on the POWL result list (defined via the output structure in the constructor of the feeder class). Again the WHEN sections of the CASE instruction show how to set the different properties of the fields that will be added to the field catalog at runtime. Finally the method returns the desired field catalog.

The corresponding fields will also be available in the POWL personalization where you can individually define which fields shall be visible in the POWL result list and which not (see section 6.2.5).

- 6) In the next step we redefine some of the methods of the feeder class which have been inherited from the feeder super class /SCMTMS/CL\_UI\_POW\_FD\_BASE:

- Class method *IF\_POWL\_FEEDER~GET\_SEL\_CRITERIA*: The coding of this POWL Feeder Class method specifies the set of selection criteria that will be available for defining POWL queries.

The generic implementation provided in the mentioned super class will just take all fields from the structure *ZENH\_S\_UI\_POW\_S\_TOR* to create the list of selection criteria at runtime.

For the example POWL, redefine the method with the following coding which uses method *GET\_SEL\_CRITERIA\_FO* created in step 5. Note that this method is then only called in case of the POWL Type being *ZENH\_TOR\_POWL* which is later defined in customizing and assigned to our new feeder class (see section 6.2.4).

```

METHOD if_powl_feeder~get_sel_criteria.
*  depending on the POWL type the corresponding selection
*  criteria will be defined

```

```

CASE i_type.

    WHEN zif_enh_ui_pow_const=>co_powl_type-enh_type-zenh_tor.
        get_sel_criteria_fo(
            CHANGING
                c_selcrit_defs      = c_selcrit_defs
                c_default_values    = c_default_values ).

    WHEN OTHERS.
        *   in case given powl type is not to be treated specifically
        *   use the generic implementation of the super class, i.e.
        *   take all attributes from the selection criteria structure
        CALL METHOD super->if_powl_feeder~get_sel_criteria
            EXPORTING
                i_username          = i_username
                i_applid            = i_applid
                i_type               = i_type
                i_langu             = sy-langu
            IMPORTING
                e_selcrit_defs_changed = e_selcrit_defs_changed
                e_default_val_changed  = e_default_val_changed
            CHANGING
                c_selcrit_defs      = c_selcrit_defs
                c_default_values    = c_default_values.

ENDCASE.

ENDMETHOD.

```

The CASE instruction in the above method implementation shows how you can reuse this method for different POWL types, i.e. depending on the POWL type the selection criteria is build up differently at runtime. This allows a reuse of one and the same POWL Feeder Class for different POWLs.

You will see in the later customizing steps that POWL Feeder Class can be assigned to different POWL Types. E.g. standard class /SCMTMS/CL\_UI\_POW\_FD\_TOR serves for Freight Order, Bookings, Freight Units and others, i.e. all POWLs related to the technical BO TOR that is used to represent these kinds of business documents.

- Class method `IF_POWL_FEEDER~GET_FIELD_CATALOG`: The coding of this POWL Feeder Class method specifies the set of fields that will be available as columns in the POWL result list.

The generic implementation provided with the super class will just take all fields from the structure `ZENH_S_UI_POW_R_TOR` to create the field catalog at runtime.

For the example POWL redefine the method with the following coding which uses method `GET_FIELDCATALOG_FO` created in step 5. Again this method is then only called in case of the POWL Type being `ZENH_TOR_POWL` which is later defined in customizing and assigned to our new feeder class (see section 6.2.4).

```

METHOD if_powl_feeder~get_field_catalog.
    * depending on the POWL type the corresponding field
    * catalog will be defined
    CASE i_type.
        WHEN zif_enh_ui_pow_const=>co_powl_type-enh_type-zenh_tor.
            get_fieldcatalog_fo( CHANGING c_fieldcat = c_fieldcat ).

        WHEN OTHERS.
            *   in case given powl type is not to be treated specifically
            *   use the generic implementation of the super class, i.e.
            *   take all attributes from the field catalog structure

```



```

CALL METHOD super->if_powl_feeder~get_field_catalog
EXPORTING
    i_username      = i_username
    i_applid        = i_applid
    i_type          = i_type
    i_langu         = i_langu
    i_selcrit_values = i_selcrit_values
IMPORTING
    e_fieldcat_changed = e_fieldcat_changed
    e_visible_cols_count = e_visible_cols_count
    e_visible_rows_count = e_visible_rows_count
    e_default_technical_col = e_default_technical_col
CHANGING
    c_fieldcat = c_fieldcat.
ENDCASE.

* add default Header for descriptions fields
add_std_description_label( CHANGING c_fieldcat = c_fieldcat )
.

* set default output values
e_default_technical_col = abap_true.
e_fieldcat_changed = abap_true.
e_visible_cols_count = 10.
e_visible_rows_count = 15.

ENDMETHOD.

```

In both methods implemented so far in this step the **CASE** instruction shows how you can reuse this method for different POWL types, i.e. depending on the POWL type (that you can define in customizing and associate with the same feeder class) the set of selection criteria and the field catalog is build up differently at runtime.

This helps implementing POWLs that are based on the same BO and Generic Result Query but shall return different BO Category Instances (e.g. one POWL type just handles Freight Orders category A while another POWL type uses the same feeder class but handles Freight Orders of category B).

- Class method **IF\_POWL\_FEEDER~GET\_ACTIONS**: Within this method you place coding that defines the set of actions that will be available to be executed via the tool bar of the POWL result list.

For the example POWL redefine the method with the following coding.

```

METHOD if_powl_feeder~get_actions.
* get the actions to be available on the POWL tool bar
DATA: ls_action_def LIKE LINE OF c_action_defs,
      ls_act_choices TYPE powl_act_choice_sty,
      lv_index       TYPE int4.

* Define the availability of some standard actions. Their
* properties are finally defined in method GET_ACTIONS of
* the super class.
mv_action_open      = abap_true.
mv_action_display   = abap_true.
mv_action_copy      = abap_true.
mv_action_create     = abap_true.
mv_action_delete     = abap_true.
mv_action_open_bd    = abap_true.

* Disable or enable certain standard actions

```

```

CASE i_type.
  WHEN zif_enh_ui_pow_const=>co_powl_type-enh_type-zenh_tor.
    * there shall be e.g. no DELETE action on this POWL
    mv_action_delete = abap_false.

  WHEN OTHERS.

ENDCASE.

* call super class to take over default actions
CALL METHOD super->if_powl_feeder~get_actions
EXPORTING
  i_username      = i_username
  i_applid        = i_applid
  i_type          = i_type
  i_selcrit_para  = i_selcrit_para
  i_langu         = i_langu
IMPORTING
  e_actions_changed = e_actions_changed
CHANGING
  c_action_defs     = c_action_defs.

* keep in mind, how many actions are already available
* at this point in time.
lv_index = lines( c_action_defs ).

* Define further actions depending on the POWL Type
CASE i_type.
  WHEN zif_enh_ui_pow_const=>co_powl_type-enh_type-zenh_tor.
    * add the action to calculate charges
    CLEAR ls_action_def.
    lv_index      = lv_index + 1.
    ls_action_def-placementindx = lv_index.
    ls_action_def-cardinality   = 'S'.
    ls_action_def-placement     = 'B'.
    ls_action_def-actionid      = /scmtms/if_ui_tor_c=>
                                sc_action-calc_transp_charges.
    ls_action_def-text          = cl_wd_utilities=>
                                get_otr_text_by_alias(
                                  '/SCMTMS/UI_CMN/CALC_TRANSP_CHARGES' ).
    ls_action_def-tooltip       = cl_wd_utilities=>
                                get_otr_text_by_alias(
                                  '/SCMTMS/UI_CMN/CALC_TRANSP_CHARGES' ).
    ls_action_def-enabled       = abap_true.
    INSERT ls_action_def INTO TABLE c_action_defs.

    * add an enhancement action that was added to the TOR BO
    CLEAR ls_action_def.
    lv_index      = lv_index + 1.
    ls_action_def-placementindx = lv_index.
    ls_action_def-cardinality   = 'S'.
    ls_action_def-placement     = 'B'.
    ls_action_def-actionid      = 'ZENH_MAINTOOLBAR_ACTION'.
    ls_action_def-text          = 'Enhancement Action'.
    ls_action_def-tooltip       =
                                'Tooltip for the Enhancement Action'.
    ls_action_def-enabled       = abap_true.
    INSERT ls_action_def INTO TABLE c_action_defs.

  WHEN OTHERS.

```

```
ENDCASE.
```

```
ENDMETHOD.
```

First of all this method calls method `GET_ACTIONS` of the super class to define some standard actions on the POWL tool bar. Moreover it adds two specific actions to the tool bar, again depending on the POWL Type. In the example, an action to start the Charge Calculation on found Freight Orders as well as an Enhancement Action (see example from section 5.4.4) is added to show how you can add required actions to the POWL tool bar.

### 6.2.3 The POWL Action Class

The POWL Action Class is used to handle the execution of the actions assigned to the tool bar of the POWL result list. The name of the class to be used is defined in the constructor of the related POWL feeder class (see step 2 & 4 in the last section 6.2.1).

In general, method `HANDLE_ACTION` is the only method to be implemented here. At runtime, the Action ID and the data of the objects selected on the POWL result list is available. This can be used to execute e.g. related BOBF actions that are either executed e.g. for all selected or only the first selected object on the POWL result list.

In section 6.2.1 Action Class `ZCL_ENH_UI_ACTION_TOR` was created which inherits from the Action Super Class `/SCMTMS/CL_UI_ACTION_BASE`. While standard actions like New, Copy, Edit, Display, etc. are handled in method `START_POWL_ACTION` of the supper class, any other actions care handled directly in method `HANDLE_ACTION` of the Action Class.

The following lines of code provide an example on how to implement the method to handle the action `CALCULATE_TRANSPORTATION_CHARGES` and the Enhancement Action `ZENH_MAINTOOLBAR_ACTION` that was described in the UI Enhancements section of this document. Both actions have been also prepared in method `GET_ACTIONS` of the POWL Feeder Class.

```
METHOD handle_action.
```

```
    FIELD-SYMBOLS: <ls_tor_root_key> TYPE /bobf/s_frw_key.
```

```
    DATA: lv_error_exists TYPE boole_d,
           lv_do_save      TYPE boole_d,
           lo_message      TYPE REF TO /bobf/if_frw_message,
           lt_key          TYPE /bobf/t_frw_key.
```

```
    CASE iv_action_id.
```

```
        WHEN /scmtms/if_ui_tor_c=>sc_action-calc_transp_charges.
```

```
        *   Action / Button: Calculate Transportation Charges
        *   The calculation is executed for all documents marked
        *   in the POWL result list
```

```
        IF it_keys IS INITIAL.
```

```
            RETURN.
```

```
        ENDIF.
```

```
        *   Execute the TOR action CALCULATE TRANSPORTATION CHARGES
```

```
        do_action(
```

```
            EXPORTING
```

```
                iv_act_key      = /scmtms/if_tor_c=>sc_action-root-
                                calc_transportation_charges
```

```
                it_key         = it_keys
```

```
            IMPORTING
```

```
                ev_error       = lv_error_exists ).
```

```
        *   If all calculations were successfull,
```

```
        *   save the corresponding changes
```

```

    IF lv_error_exists IS INITIAL AND iv_source = co_source_powl.
        lv_do_save = abap_true.
    ENDIF.

    WHEN 'ZENH_MAINTOOLBAR_ACTION'.
        * Action / Button: Enhancement Action. The action is only
        * executed for the first document marked in the POWL result list
        READ TABLE it_keys ASSIGNING <ls_tor_root_key> INDEX 1.
        IF sy-subrc IS INITIAL.
            CLEAR lt_key.
            APPEND <ls_tor_root_key> TO lt_key.
        * Execute the TOR action CALCULATE TRANSPORTATION CHARGES
        do_action(
            EXPORTING
                iv_act_key      = zif_enh_tor_c=>sc_action-root-
                               zenh_maintoolbar_action
                it_key          = lt_key
            IMPORTING
                ev_error        = lv_error_exists ).
        ENDIF.

    WHEN OTHERS.

    ENDCASE.

    * In case a SAVE is required
    IF lv_do_save IS NOT INITIAL.
        mo_tramgr->save( IMPORTING eo_message = lo_message ).
        mo_tramgr->cleanup( ).
        add_message( io_message = lo_message ).
    ENDIF.

    ENDMETHOD.

```

## 6.2.4 The basic POWL Customizing

In the previous sections, the coding foundation for a new POWL has been described and created. To really see and use the new POWL on the TM UI, customizing settings need to be done which will ensure that the POWL will be visible for users with a specific role and in the correct application area. The POWL created in section 6.2 handles Freight Orders. We will therefore assign it to the corresponding application area Freight Order Management.

- 1) **Transaction *POWL\_TYPE*:** The transaction registers the POWL Feeder Class and adds a description that is used in the POWL dialog for creating user specific POWL query definitions.

- Start transaction *POWL\_TYPE* and create a new entry as follows:

Field	Value
Type	ZENH_TOR_POWL
Description	Enhancement TOR POWL Type
Feeder Class	ZCL_ENH_UI_POW_FD_TOR
Sync. Call	(not set)
No Msg. Wrapping	(not set)

- Save the new entry.
- 2) **Transaction *POWL\_TYPER*:** The transaction assigns the POWL Type (see step 1) to a PFCG Role, i.e. all users assigned to this role will be able to see the new POWL in the application area to be specified.

- Start transaction *POWL\_TYPER* and create a new entry as follows:

Field	Value
Application	SCMTMS_POWL_FO
Role	SAP_QAP_TRANSPORTATION_MANAGER
Type	ZENH_TOR_POWL
Description	Demo Enh. TOR POWL Type

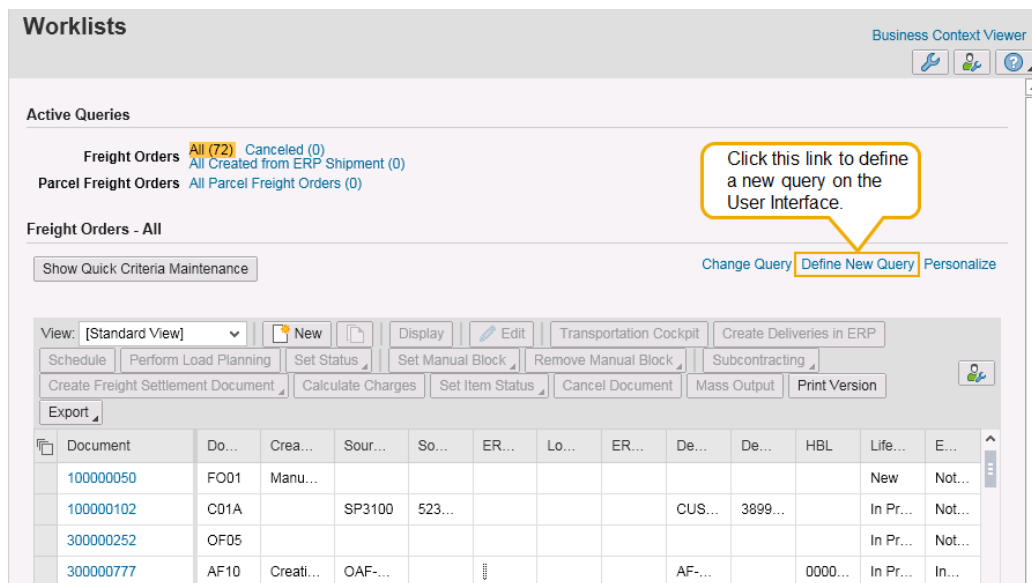
- Save the new entry.

Note that you will only see the new POWL on the SAP TM User Interface if your user has the PFCG role assigned (→ *SU01*) that is used in the above customizing entry. There is also transaction *POWL\_TYPEU* available which allows assigning the POWL to a specific user, i.e. it allows further restricting the visibility of the POWL to a specific set of users.

With this two customizing entries in place you can already specify a new POWL Query on the TM User Interface. How to do this is described in the next section.

### 6.2.5 Creating POWL Queries

Start the NetWeaver Business Client (transaction *NWBC*) and choose the role that the example POWL Type *ZENH\_TOR\_TYPE* is assigned to (make sure that your user has this role assigned in e.g. transaction *SU01*). Navigate to *Freight Order Management* → *Road* → *Overview Road Freight Orders*. In the POWL Result List section you can now click on the link *Define New Query*. The following guided procedure will request you to enter all relevant parameters for specifying a new POWL Query.



Picture: Define a new POWL Query.

- 1) Select Object Type: In field Select Object Type choose the entry *Enhancement TOR POWL Type* (which is nothing else but the description of the POWL type that was customized in section 6.2.4. Click on button *Next*.

**Worklists** Business Context Viewer

**Define New Query**

1 Select Object Type 2 Maintain Criteria 3 Finish

Select Object Type: Demo Enh. TOR POWL Type

Select Existing Query as Template:

< Previous Next > Cancel

1) In the list of Object Types you can find the Object Type that corresponds to the POWL Type that was created in the example.

2) Click on button Next to get to the next step

Picture: Select Object Type (POWL Type).

- 2) **Maintain Criteria:** On this screen of the guided procedure you can define default values for a set of selection criteria. Here, all selection criteria are visible that were specified in method `GET_SEL_CRITERIA` of the underlying POWL Feeder Class. Note that the two enhancement fields that were added to the query are now available as selection criteria too.

**Worklists** Business Context Viewer

**Define New Query**

1 Select Object Type 2 Maintain Criteria 3 Finish

**General Data**

Document: To

Document Type: To

**Organizational Data**

Plan. Exec. Org.: To

Plan. Exec. Group: To

Purch. Organization: To

Purchasing Group: To

**Administrative Data**

Created By: To

Changed By: To

**Enh. Entry Date**

Date/Time: To

**Enh. Change Date**

Date/Time: To

**General Selection Settings**

Max. No. of Records: 500

Preview Criteria Personalization

< Previous Next > Cancel

All available Selection Criteria for the POWL Query can be provided with default values.

The enhancement fields that were added to the Generic Result Query are now available as Selection for the POWL Query too.

2) Click on button Next to get to the next step

Picture: Maintain Selection Criteria.

In the example we just leave all fields blank by default except the *Maximum Number of Hits* which is set to 500. As indicated you can define default values here which can also be changed later on. When all required default / predefined values are specified, click on button *Next* again.

- 3) Query Description and Category: On the final screen you enter a Query Description, e.g. *New Enhancement Demo Query* and set the flag *Activate Query*.

Picture: Query description and new category.

In the example we want to assign the new query to its own category. Click on button *Create New Category* to create a new category for the query.

Picture: Entering a new category.

Make sure that the new category is then selected on the previous screen.

- 4) Now you can click on button *Finish*. The new POWL Query is now assigned to the Freight Order Management application and can be seen on the User Interface. The final result of this example looks as follows:

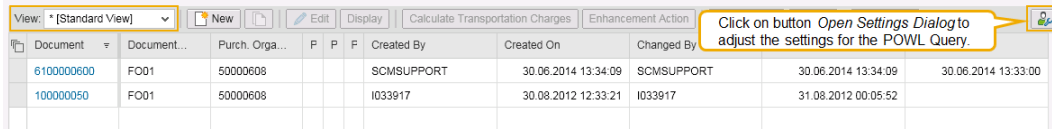
Document	Document...	Purch. Orga...	Purchasing...	Plan. Exec....	Plan. Exec. G...	Created By	Created On	Changed By	Changed On	Date/Time
6100000600	F001	50000608				SCMSUPPO...	30.06.2014 1..	SCMSUPPO...	30.06.2014 1..	30.06.2014 1..
6100000551	COPU	50001582				D057773	02.04.2014 0..	D057773	02.04.2014 0..	02.04.2014 0..
6100000550	COPU	50001582								
6100000502	COPU	50001582								
6100000501	COPU	50001582								
6100000457	F000	50001425				D056218	09.01.2014 1..	D056218	09.01.2014 1..	09.01.2014 1..
6100000456	F000	50001425				D056218	09.01.2014 1..	D056218	09.01.2014 1..	09.01.2014 1..
6100000455	RFSC	50001425				D056218	09.01.2014 1..	D056218	09.01.2014 1..	09.01.2014 1..

Picture: The final new POWL Query.

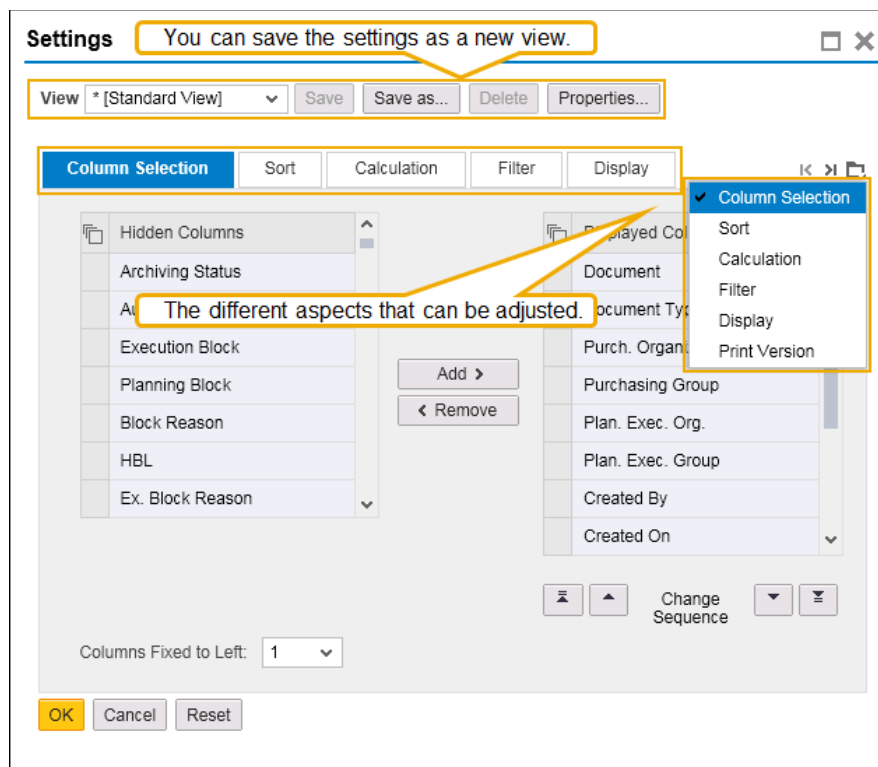


In the picture above you can see the POWL Query displaying a result where one found document is selected. The data of this document also contains entries for the extension fields that were added to the involved Generic Result Query. Moreover, in the POWL tool bar you can see the two actions that were added in addition to the standard actions. The example POWL can now be personalized like any other standard POWL.

- 5) Once the new POWL Query is up and running you can further adjust and personalize it via a settings dialog. Click on the settings button on the right side of the POWL tool bar (the last button) to start the dialog:



Picture: Starting the Settings Dialog for a POWL Query.



Picture: Settings Dialog for POWL Query.

On the dialog you can find tab strips that allow setting different aspects of the POWL Query. In the topmost toolbar you can define and manage (save, delete) your settings as views that you can then directly choose on the POWL Queries result list. So you can e.g. create such views to show documents which are relevant in a specific business case. The following settings can be done:

- Column Selection: You can define which available columns from the field catalog shall be shown on the result list. Moreover you can define or adjust the sequence of the columns.
- Sort: Allows defining a set of sort criteria, i.e. a list of columns that the sorting of the results shall be based on. For each sorting column you can define whether sorting shall be done ascending or descending.

- **Calculation:** This allows setting up a calculation for available columns, e.g. the minimum, maximum, totals or mean value of a column in the result list. The calculation figures are displayed as additional rows in the result list that are marked with a different background color (similar to an ALV control).
- **Filter:** Similar to Sort this allows setting up filters on a set of columns to restrict the result list to specific entries.
- **Display:** This allows setting e.g. the number of visible rows and columns in the result list and some general layout settings.
- **Print Version:** These settings determine parameters that shall be used when printing a POWL result list, e.g. the scaling, margins and page size of a printout.

## 6.2.6 Additional POWL Customizing

In section 6.2.4 the basic customizing was described required to display the example POWL on the UI. In transaction *POWL\_TYPE* a new POWL Type was defined and assigned to the newly created POWL Feeder Class. Then in transaction *POWL\_TYPER* the POWL Type was assigned to a specific PFCG role to make it visible for users that have this role assigned and logon with in the SAP TM System.

There are some more customizing settings that can be created in the context of a POWL and related POWL queries:

1. Assign a POWL to a specific User with transaction *POWL\_TYPEU*: This allows an even more fine granular assignment of a POWL to a specific user.

Picture: Assigning a POWL Type to a specific user.

2. Create a so called Admin Work List. They can be defined to be visible in the standard overviews, i.e. they can be used to create a standard set of POWL Queries which are always available and visible in contrast to a POWL Query that a User configures for himself as described in section 6.2.5. Use transaction *POWL\_QUERY* to define such Admin Work Lists.

Picture: Defining an Admin Work List Query in *POWL\_QUERY*.

In transaction *POWL\_QUERY* you can also define the refresh behavior of the POWL Query (e.g. manual refresh only or refresh on every page visit, synchronous or asynchronous refresh). Moreover a predefined layout can be provided with a layout variant that you can e.g. create via the button *Layout Variant* in the tool bar of the transaction.

3. Similar to the mentioned transactions *POWL\_TYPER* and *POWL\_TYPEU*, the transactions *POWL\_QUERYR* and *POWL\_QUERYU* allow assigning a Admin Work List query to a PFCG Role or a user respectively.

**New Entries: Details of Added Entries**

Application: SCHTMS\_POWL\_F0  
 Role: /SCHTMS/TRANSPORTATION\_MGR\_V2  
 Query ID: ZENH\_TOR\_ALL

Assignment of the POWL Query to the application and PFCG role.

View: Query - Role assignment

Category: ZENH\_POWL\_CAT  
 Description: All Enhancement FOs  
 Category sequence no:   
 Query sequence no:   
 Tab sequence no:   
☐ Activate

Picture: Assigning an Admin Work List Query to a PFCG Role (*POWL\_QUERYR*).

When assigning a query to a PFCG Role in transaction *POWL\_QUERYR* you can also define the position of the query within a set of queries of the same category by specifying a corresponding *Category Sequence Number*, *Query Sequence Number* and a *Tab Sequence Number*.

The *Category Sequence Number* defines the category that you want to display the query in. *Query Sequence Number* and *Tab Sequence Number* set the query position within the POWL Category.

A Category can be defined in customizing via the following path: *SPRO* → *Cross Application Components* → *General Application Functions* → *Generic SAP Business Suite Functions* → *Personal Object Worklist* → *Cockpit for POWL Administration*.

In transaction *POWL\_QUERYU* you provide the same entries but provide a specific user name instead of a PFCG role. Again this allows a more fine granular assignment of such a POWL Query to a specific user.

## 6.2.7 Enhancing a standard POWL

Based on the example POWL that was created from scratch in sections 6.2.2 to 6.2.4 you could already see the most important parts and places in the coding of a POWL. The example indicates how to implement a completely new POWL. But many customers and partners ask for a way how to extend existing standard POWLs without necessarily having to create all the code for feeder- and action classes and even implement a completely new Generic Result Query, etc. (this makes of course sense when none of the standard POWLs really fulfills the requirements).

Well, the first answer to this question is that there is actually no explicit enhancement concept for existing standard POWLs. But nevertheless, also standard POWLs can be enhanced by using the Query Enhancement mechanism described in section 6.1 and using implicit enhancement at certain places in the POWL coding (see section 4.5). The example with the

completely new POWL contains all parts and coding places that you need to know and take into consideration for enhancing a standard POWL. The basic enhancement steps:

1. Enhance the Generic Result Query of the standard POWL (you can find the used query in the POWL Feeder Class Constructor) as described in section 6.1.5).
2. Identify the selection criteria and result structure of the standard POWL (remember that this again can be found in the constructor of the POWL Feeder Class) and place the enhancement fields into these structures via a new (or an existing) Append.
3. Enhance method *IF\_POWL\_FEEDER~GET\_SEL\_CRITERIA* of the POWL Feeder Class. The coding of this method specifies the set of selection criteria that will be available for defining POWL queries.
  - It returns the defined selection criteria Meta Description in changing parameter *C\_SELCRIT\_DEFS*. You can use e.g. a Post- Method (implicit enhancement) to add required code for adding the new enhancement fields to the list of selection criteria in the mentioned changing parameter.
  - You can also adjust or remove the Meta Description of the standard selection criteria there, e.g. adjusting the grouping, representation, label texts, etc.
  - How this is done is indicated in the example of section 6.2.2, step 5 where e.g. two additional enhancement fields are added to the selection criteria list. In your coding you should make sure that the additional fields are only added for the required POWL Types (in the example and other standard implementations the POWL Type is distinguished via a *CASE* statement).
  - Changing parameter *C\_DEFAULT\_VALUES* allows e.g. programmatically adding default values for the new (and existing) selection criteria attributes.
4. Enhance class method *IF\_POWL\_FEEDER~GET\_FIELD\_CATALOG* of the POWL Feeder Class: The coding of this method specifies the set of fields that will be available as columns in the POWL result list.
  - It returns the defined output field catalog Meta Description in changing parameter *C\_FIELDCAT*. Again you can use e.g. a Post Method (implicit enhancement) to add the required code for adding the new enhancement fields to the list of output attributes in the mentioned changing parameter.
  - You can also adjust or remove the Meta Description of the standard output attributes there, e.g. adjusting the display type, header texts, column position, visibility, etc. Take a look at the constants defined in the standard POWL Constants Interface */SCMTMS/IF\_UI\_POW\_CONST*. Here you can find constants for e.g. display types (*CO\_DISPLAY\_TYPE*) and parameter types (*CO\_PARAM\_TYPES*) which you can reuse to set the properties of output attributes.
  - Review the example of section 6.2.2, step 6 which indicates how this can be done. Here the same enhancement fields are added to also have them available as attributes in the result list of the POWL. Just like for the selection criteria your code should make sure that the additional fields are only added for the required POWL Types (in the example and other standard implementations the POWL Type is distinguished via a *CASE* statement).
5. Enhance class method *IF\_POWL\_FEEDER~GET\_ACTIONS*: Within this method you can add coding (e.g. via a Post Method) that defines a set of additional actions to be available on the POWL tool bar. Note that this method only defines what actions shall be available and not how they are executed. This is done in the POWL Action Class as indicated in the example of section 6.2.3.

- It returns the defined actions and their properties in changing parameter `C_ACTION_DEFS`. You may use e.g. a Post Method (implicit enhancement) to add further required action definitions in the mentioned changing parameter.
  - You can also adjust properties of the standard actions, e.g. adjusting the text tool tip display type, header texts, the sequence of the actions on the tool bar, etc. For each action you need to make sure that it gets an Action ID assigned. This will be used by the POWL Action Class implementation to identify the action to be executed at runtime.
  - Again the example of section 6.2.2, step 5 indicates how this can be done. Here in addition to some standard actions, two more actions are added to the POWL tool bar. Just like before you can use the POWL Type (available in the mentioned Feeder Class method) in a `CASE` statement to e.g. show certain actions only for specific POWL types.
6. Finally you can enhance `HANDLE_ACTION` to react on the execution of newly added actions, i.e. here you identify the Action ID and specify the coding that shall be executed for each action. Again you may use e.g. a Post Method (implicit enhancement) to add the coding for your own actions. The example for the implementation of a POWL Action Class from section 6.2.3 indicates how to implement this.

With these basic steps you can also enhance or adjust standard POWLs to your specific needs. As mentioned there is no explicit enhancement concept but implicit enhancements help in this case to get required additional coding implemented. Depending on the use case it makes sense to just enhance a standard POWL as described. On the other hand, in more complicated use cases it might be reasonable to implement your very own POWL with a Generic Result Query and other parts that are specifically designed for this.

Performance is always an important (if not the most important) aspect of any enhancement or implementation of a POWL. Make sure that you enhance or implement in this area in a way that you provide performant coding to your users.

## 6.2.8 POWL Maintenance Reports

For administration purposes there is a list of reports available that allow resetting or displaying certain information of POWLs. They can be useful during development or e.g. when having to clear inconsistencies in POWL configurations, buffered data, etc. Nevertheless, they should be handled with care and only be used by experienced administrators.

Report	Comment
POWL_D01	Delete Queries from Database.
POWL_D02	Show POWL Design Information.
POWL_D03	Check the consistency of POWL Table Entries.
POWL_D04	Delete cached selection criteria for Admin Queries.
POWL_D05	Delete POWL Check Results.
POWL_D06	Activate Derived Queries
POWL_D07	Delete Shadowing Entries
POWL_D08	Delete Admin Layouts
POWL_D09	Delete Default Layout Mapping

## 6.2.9 POWL Performance

POWLs are the main entry point for end users to access data in SAP TM. They use the selection criteria of POWL Queries to e.g. select the documents they are responsible for. Depending on the selection criteria the result of a POWL Query can be quite large and may contain e.g. documents that are not required any longer as they are already finalized, etc.

Experience shows that many times the POWL Queries return more result records than actually necessary with the effect that users experience a rather slow performance of the POWLs. To make sure that users get the right amount of data in a performant way, consider the following measures for improving the performance.

## Indexes for the most common selection criteria

Although the standard TM delivers a set of (secondary) indexes for most common (standard) attributes, your POWL Query might be defined in a way that it selects its result data mainly via attributes which are not covered by a secondary index.

For improving the POWL Query performance it can make sense to add secondary indexes on additional attributes. For identifying the right database table attribute connected with the POWL Query selection criteria you can do e.g. the following (provided that the BOPF Query used by the POWL is implemented using the super class `/SCMTMS/CL_Q_SUPERCLASS`).

In method `DO_SELECT_NO_SUBQUERY` or `DO_SELECT_WITH_SUBQUERY` the actual `SELECT`-Statement of the query is implemented and executed. Set a break-point here and take a look at the `SELECT` that is being executed at runtime. This helps finding the right attribute of the right database table. In the simplest case, the used BOPF Query only accesses a single BO node, i.e. a single database table. It might be a bit more difficult the right database table attribute in case the `SELECT`-Statement contains joins over multiple tables.

When you have identified the right database table and attribute, you can create a corresponding secondary index for it which helps accelerate the POWL Query execution.

But keep in mind that a database table should not have too many indexes. When attributes of database table records are changed or records are added, the affected indexes are updated too. With too many indexes in place it can actually happen that the performance even slows down due to the significant overhead of keeping the indexes consistent and up-to-date. Moreover, it might make sense to do an SQL Trace with e.g. transaction ST05 to determine and verify that your new index is actually used by the database to access the data.

## Using Calculated Dates in POWL Query definition

An additional index on a database table attribute can help to improve the performance of the POWL Query. But in many cases customers, partners and end users define POWL Queries that simply select too many data records. For example, a user would like to see only the Freight Order of the last 30 days in his or her responsibility. Instead of changing the POWL Query frequently by adjusting the provided date attributes, the POWL Framework provides a feature that allows defining so called Calculated Dates.

The following example is based on the Road Freight Order POWL and shows how to use the Calculated Dates feature which can be used for restricting the amount of records returned by a POWL Query automatically to only those records (documents) that are within a certain time frame (and fulfill other provided selection criteria) which in turn helps improving the performance.

1. Start transaction NWBC to start the NetWeaver Business Client and choose a corresponding role to get to the SAP TM User Interface. Then navigate to *Freight Order Management* → *Road* → *Overview Road Freight Orders*.
2. Click on the link for POWL Query *All Road Freight Orders* (make sure that it returns a reasonable amount of result records). The query should now come back with a set of documents that were found based on the given selection criteria.
3. Now click on the link *Change Query* to adjust the given selection criteria.





## 7 Enhancing Print Forms

SAP Transportation Management makes use of Output Management to print, fax or email various documents such as Forwarding Instructions, Pro Forma Freight Invoices, Air Waybills and many more. The TM standard provides predefined, PDF-based forms (SAP Interactive Forms) for this kind of documents that you can find in transaction *SFP* (Form Builder) by using the F4-Help in field Form on the initial screen and searching for */SCMTMS/\**.

The data to be used on a form is read from the TM backend via a so called printing class that passes this data on to a predefined form interface. This interface contains the definition of fields (e.g. header attributes of a Forwarding Order) and deep structures (e.g. item data of a Forwarding Order) to be placed on the form. The layout and look & feel of such forms are defined with the help of the Adobe LiveCycle Designer.

SAP Transportation Management uses an Output Management Adapter for BOPF Business Objects to automate the output (automated backend or manual front end output) of the print forms. This adapter makes use of the Post Processing Framework (PPF) for generating and processing outputs. How e.g. an external communication (print, fax, email) is done is defined in related customizing (Output Management is moreover used for A2A, B2B, NetWeaver Alert and Workflow Task communication).

The provided standard forms may not contain all information required by customers or the layout might not match the requirements. For example there are customer specific extension fields on business object level that shall be also printed with a form or the layout needs to be adjusted, e.g. changing the sequence of fields or adding a company logo.

In the following sections we see how to enhance existing forms and how to create completely new forms. Moreover the required configuration and customizing of Output Management and PPF will be shown based on a working example.

### 7.1 Enhancing a standard form

The standard forms delivered with SAP Transportation Management should only serve as a template, i.e. you should not change the standard form itself. It is recommended to copy the standard form to be used and add all required content to this new form. In the following simple example we will use a copy of the standard print form */SCMTMS/FP\_FFDOC* (TM Forwarding Instructions). We will keep the standard printing interface and printing class and not replace it with corresponding copies (→ the second example will show how to create a completely new form from scratch including the creation of a new printing interface and a printing class).

#### 7.1.1 Enhancing the involved BO(s)

A form might require additional standard fields that are not yet used on the form or customer-specific fields were added to a business object that shall be e.g. printed with a related form. In our example the Forwarding Order BO (*/SCMTMS/TRQ*) is the source of data from where the form will get the content to be printed.

In general, multiple BOs and/or database tables could serve as data source for a form but corresponding Output Management Actions will be only assigned with a leading BO (node) from where you can trigger them).

As a first step we add the following customer-specific extension fields to the Root node of the BO (review section 3.3.4 for details on how to create extension fields on a business object). This is of course an optional step. Of course you could also add existing standard fields to a form:

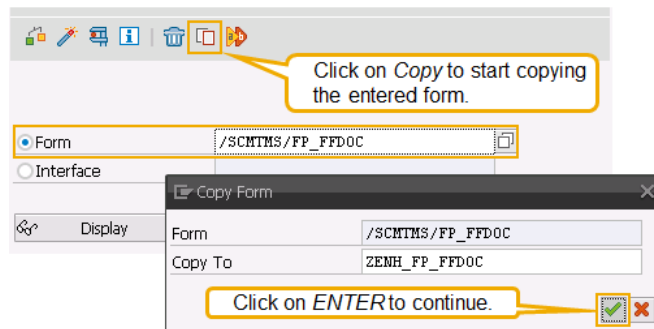
Component	Typing Method	Component Type
ZENH_ENTRY_DATE	Types	/SCMTMS/DATETIME
ZENH_SALESORG_ID	Types	/SCMTMS/SALES_ORG_ID
ZENH_APPROVED	Types	FLAG

In the following steps we will bring these extension fields onto a copy of print form `/SCMTMS/FP_FFDOC` (TM Forwarding Instructions). In the described example, we follow a bottom-up approach, i.e. we start with some additional fields on a BO to get them to the print form..

### 7.1.2 Copying the standard form

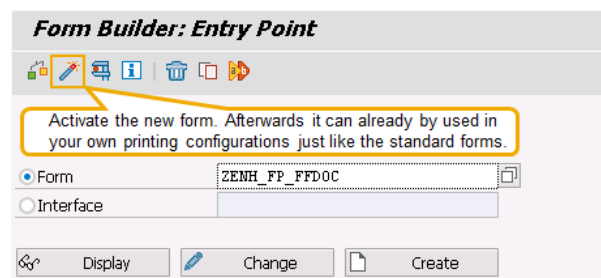
First of all create a copy of the standard form `/SCMTMS/FP_FFDOC` as follows:

- 1) Start transaction `SFP`, select radio button *Form* and enter the form to be copied in the related input field (or use the F4-Help of the input field to find the form).
- 2) In the menu bar select *Form Object* → *Copy...* or press `CTRL+F5`.
- 3) On the following popup screen, enter the new name for the copy of the form. Example: `ZENH_FP_FFDOC`.



Picture: Creating a copy of a standard form.

- 4) On the following popup, define a package where to store the new copy. Then save the form and go back to the initial screen of transaction `SFP`, activate the new form.



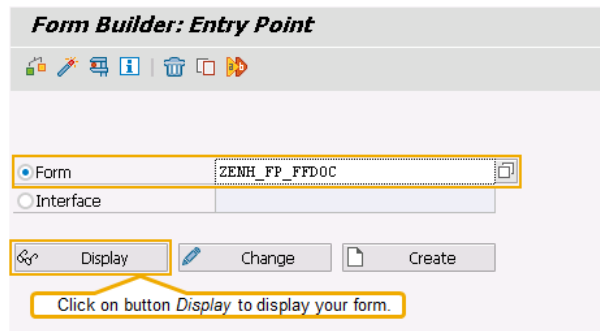
Picture: Activating the new form.

- 5) Such a new copied form can now already be used in a printing configuration (Output Management / PPF) just like the original standard form (We'll see how to do this kind of configuration in section 7.4).

### 7.1.3 Enhancing the Print Structure of a Form

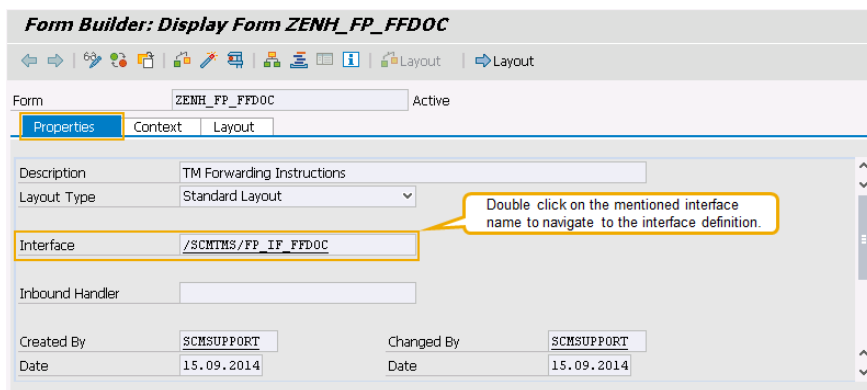
In the second step we place the extension fields in the print structure of the form. The print structure is represented by a corresponding DDIC structure that defines the data which can be placed on the form. The definition can be found via the form interface. To identify the underlying technical DDIC structure of the form's print structure execute the following steps:

- 1) Start transaction *SFP* and display your form.



Picture: Display a form via transaction SFP.

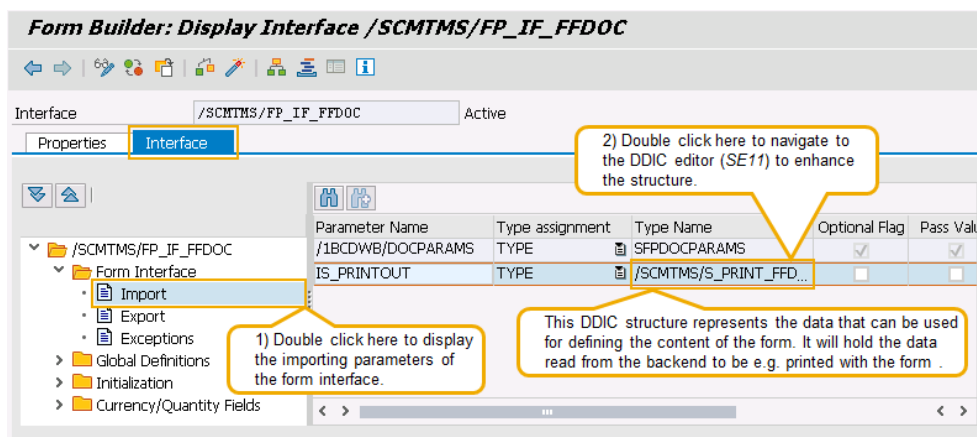
- 2) On the following screen, navigate to tab strip *Properties* and double click on the interface mentioned there. The interface contains the data structures that the form uses to represent its content.



Picture: Navigate to the interface of the form.

- 3) You can now see the details of the interface. Navigate to tab strip *Interface*. On the left side you can find the content and details of the interface represented as a tree structure. The example interface is */SCMTMS/FP\_IF\_FFDOC*. In the tree open the following path: */SCMTMS/FP\_IF\_FFDOC* → *Form Interface*. Then double click on the entry *Import*.

On the right side you can see now the parameter *IS\_PRINTOUT* with its corresponding DDIC Type */SCMTMS/S\_PRINT\_FFDOC* in column *Type Name*. Double click on this DDIC structure to edit it as described in the next step.



Picture: Identifying the content DDIC structure of the form.

- 4) After having double clicked on the DDIC structure `/SCMTMS/S_PRINT_FFDOC`, it will open in the DDIC editor (transaction `SE11`). Here we can now add the customer specific fields to be placed on the form. On the button bar, click on *Append Structure...* for creating a new Append. On the following popup enter the Append name. Example:

Append Name	ZENH_FP_FFDOC
Short Description	Append for additional fields on form /SCMTMS/FP_FFDOC

Add following fields in the new Append. Then save and activate the Append.

Component	Typing Method	Component Type
ZENH_ENTRY_DATE	Types	/SCMTMS/DATETIME
ZENH_SALESORG_ID	Types	/SCMTMS/SALES_ORG_ID
ZENH_APPROVED	Types	FLAG

**Dictionary: Display Structure**

Structure: `/SCMTMS/S_PRINT_FFDOC` Active  
Short Description: Print structure FF Document for Adobe interface

Append Structure...

1) Click here to create a new Append structure with the additional fields.

Create Append Structure for `/SCMTMS/S_PRINT_FFDOC`

Append Name: `ZENH_FP_FFDOC`

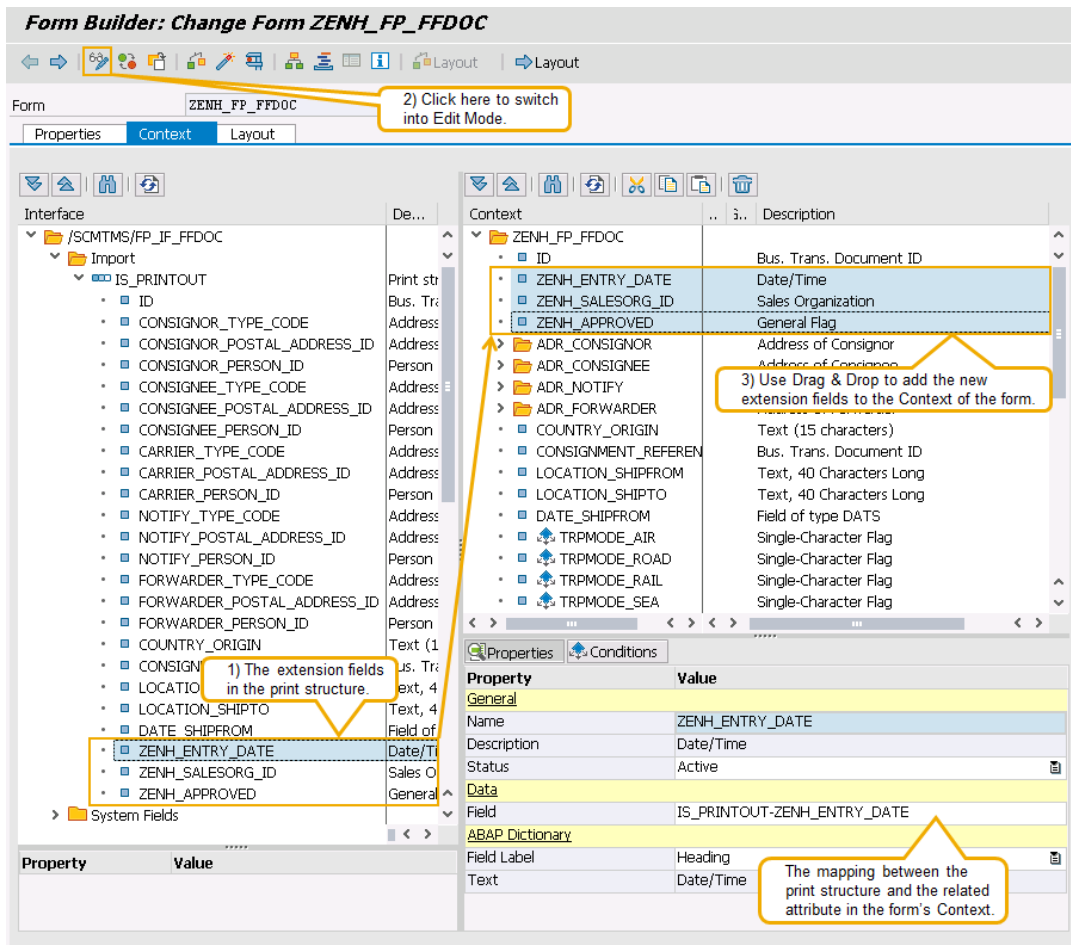
2) Enter the Append name here and click on Enter.

3) The new Append with the additional fields.

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
QUOT_VALID_FROM	Types	DATS	DATS	8	0	Field
QUOT_VALID_TO	Types	DATS	DATS	8	0	Field
UNLOADING_POINT	Types	/SCMTMS/UNLOADI...	CHAR	25	0	Unloading Point Name
.APPEND	Types	ZENH_FP_FFDOC		0	0	Append for additional fields on form /SCMT...
ZENH_ENTRY_DATE	Types	/SCMTMS/DATETIME	DEC	15	0	Date/Time
ZENH_SALESORG_ID	Types	/SCMTMS/SALES_O...	NUMC	8	0	Sales Organization
ZENH_APPROVED	Types	FLAG	CHAR	1	0	General Flag

Picture: The DDIC structure of the form with the new Append.

- 5) Now navigate back from the DDIC Editor to the Form Interface and further back to transaction `SFP`. Here click on tab strip *Context*. On this tab strip you can see the form interface again. In the tree open the path `/SCMTMS/FP_IF_FFDOC` → *Import* → *IS\_PRINTOUT*. The fields of the new Append should now be visible here as well.



Picture: Adding the new fields to the Context of the form.

Switch into edit mode (**CTRL + F1**). Now mark the added extension fields and place them into the form context via drag & drop as shown in the picture above. Drag & drop allows initial placement of the fields as well as subsequently adjusting their position in the hierarchy of the form context. Finally, save and activate the form again.

### 7.1.4 Providing data to enhanced fields

Now the new fields added in the print structure need to be provided with data to be printed. For this, we first of all need to identify the corresponding print document class which provides the data to the form in standard (remember that in this example we have not created our own printing class).

- 1) Start transaction **SE24** (Class Builder) and use the F4-Help on the initial screen to search for classes with the pattern **/SCMTMS/\*PRINT\***. For our example print form **/SCMTMS/FP\_FFDOC** (Forwarding Order) the following class is the right one:

**/SCMTMS/CL\_PRINTOUT\_FWO**

- 2) All print classes provided with SAP Transportation Management inherit from super class **/SCMTMS/CL\_PRINTOUT** which provides in general two methods:

- **FILL\_PRINTSTRUCTURE**: This method is overwritten by the implementation of print class **/SCMTMS/CL\_PRINTOUT\_FWO**. It contains the coding that reads the data to be printed at runtime.

- *PRINT\_DOCUMENT*: Contains coding to prepare the printing of a form and calls method *FILL\_PRINTSTRUCTURE* to get the data from the application backend. Its implementation is taken over from the mentioned super class.

In printing class */SCMTMS/CL\_PRINTOUT\_FWO* you can find further implemented methods that serve as data providers or helper methods for accessing required data.

- 3) In our example the fields added to the Root node of business object */SCMTMS/TRQ* are already read within method *BUFFER\_DATA* of class */SCMTMS/CL\_PRINTOUT\_FWO*. Here helper class method */SCMTMS/CL\_TRQ\_HELPER=>READ\_NODES* returns the required data from the corresponding BO instance, i.e. besides other information the complete Root node data, including the extension fields.

In other examples this might not be the case. Then you need to add coding in method *FILL\_PRINTSTRUCTURE* to read the additional data and map this data to the corresponding fields in the print structure.

In our example we just need to add a few lines of code to map the available content of the new fields coming from the BO onto the corresponding fields of the print structure. To do this, you can e.g. create a post exit for method *FILL\_PRINTSTRUCTURE*.

In any case, review the standard implementation of method *FILL\_PRINTSTRUCTURE* to get to know the way how data is read and mapped for the form. In general this is fairly easy ABAP coding that can be enhanced accordingly. When writing your own access methods for additional data to be provided to the print form, always make sure that this is done in a way that ensures a maximum of performance.

## 7.2 Adjusting the Layout

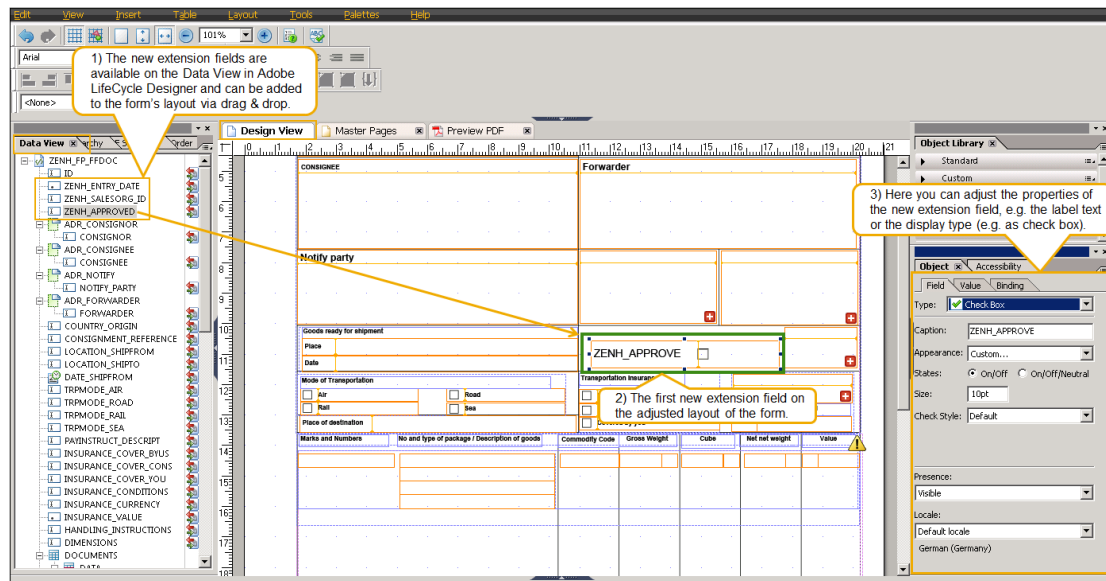
### 7.2.1 Adobe LiveCycle Designer Installation

The layout of the PDF-based forms is specified and adjusted using the Adobe LiveCycle Designer which is required to be installed on your local machine. Review SAP note 1522483 for instructions how to download this tool from the SAP Service Market Place and how to install it. As a customer or partner please make sure in advance that you have sufficient authorization to download software from the Service Market Place (SAP internally, you can get this authorization via the access enforcer as described in note 1037575).

### 7.2.2 Placing additional content on the form layout

As shown in section 7.1 we now have the new extension fields available in the context of the form. The fields of the context are now available within the Adobe LiveCycle Designer and can be integrated into the layout of the form with the following example steps:

- 1) In transaction *SFP* Click on tab strip *Layout* to display the form definition in the Adobe LiveCycle Designer. If installed correctly, you should now see a layout as shown below. Make sure that *SFP* is in edit mode when adjusting the layout.
- 2) In the Adobe LiveCycle Designer you can now drag & drop the new fields from the data view into the layout view and maintain corresponding field properties.



Picture: The form in the layout view of the Adobe LiveCycle Manager.

### 3) Finally save and activate the form again.

With the form adjusted you could now create e.g. a Forwarding Order document with a document type that has a corresponding printing configuration assigned. For example you could configure a manual Output Management / PPF action that allows displaying the form via the print preview in section *Output Management* of the Forwarding Order UI. Such a configuration is described in the example of section 7.4.



## 7.3 Creating a new form

In the following example we will see how to build a print form from scratch, using some of the steps and aspects already described in the previous sections. As an example a very simple form based on BO /SCMTMS/TRQ (Forwarding Order) is created. The form shall contain header and item information of a Forwarding order. In section 7.4 a PPF/Output Management example configuration is created that allows displaying the new form in the preview section of the output management section of the Forwarding Order UI.

### 7.3.1 Creating a print structure and table type

Before the actual form is created we first of all need to create a corresponding print structure, a related table type and a form interface. The print structure is the technical DDIC representation of the form content. The related table type allows the representation of a list of documents based on the corresponding print structure. The form interface contains and combines all the technical structures and parameters of the form.

The data source for the new form to be created is again BO /SCMTMS/TRQ (Forwarding Order). In this example data from the Root and Item node of the BO will be shown. This means that the print structure is allowed to be and will be a deep structure as there can be multiple items in a corresponding BO instance. Create the print structure as follows:

- 1) Start transaction *SE11*, select radio button *Data type* and enter the name of the print structure in the input field next to it. Example: *ZENH\_S\_PRINT\_TRQ*.
- 2) In the menu bar select *Dictionary Object* → *Create...* or press *F5*.
- 3) Specify fields and tables in the new structure, representing the content of the form (keep in mind that the data will come from BO /SCMTMS/TRQ). There are two options to do this:
  - You define your own list of fields and tables that make up the structure. This will later require some coding to map the data of BO node instances onto the corresponding parts of the print structure. Or:
  - You can reuse the node structures and table types that are used to specify the corresponding BO nodes and provide the data for the form. When reading the BO data in this format the mapping onto the print structure is very easy to realize (move-corresponding). In this example we will follow this second approach for simplification reasons.
- 4) Define the print structure by including the combined structure of the TRQ Root node and defining a component *ITEM* that is specified with the combined table type for the TRQ Item node (in transaction */BOBF/CONF\_UI* you can browse the node model and identify the used component types).

Structure		Description
ZENH_S_PRINT_TRQ		Enh. Demo: Print Structure for a new TRQ Form
Component	Typing Method	Component Type
.INCLUDE	Types	/SCMTMS/S_TRQ_ROOT_K
ITEM	Types	/SCMTMS/T_TRQ_ITEM_K

This will allow placing all data on the form that is coming from the Root node and the item node of a TRQ instance. As the combined structure also will contain extension fields via the related extension includes they will be also implicitly available for placing them on the form.

- 5) Save and activate the new structure.

- 6) Use transaction *SE11* to create table type *ZENH\_T\_PRINT\_TRQ* with structure *ZENH\_S\_PRINT\_TRQ* as the line type.
- 7) Save and activate the new table type.

### 7.3.2 Creating a form interface

Now we can create the required form interface. The most important information in this interface is the assignment of the print structure which represents the data that will be available to define the content of the new form.

- 1) Start transaction *SFP*, select radio button *Interface* and in the menu bar select *Form Object* → *Create...* or press *F5*.
- 2) On the next popup screen enter the following data to specify the form interface:

Interface	<i>ZENH_FP_IF_TRQ</i>
Description	Demo Enhancement Interface for TRQ
Interface Type	ABAP Dictionary-Based Interface

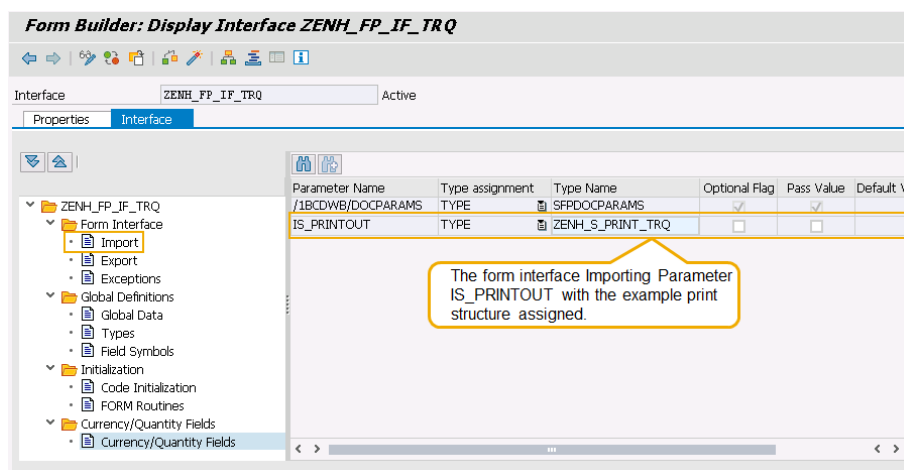
- 3) Click button *Save (Enter)* on the popup screen and specify the package where to place the new interface (you can e.g. create the interface as a local object in package *\$TMP*).
- 4) The form interface will be based on DDIC structure *ZENH\_S\_PRINT\_TRQ* created in the previous step. This is assigned to the form interface as follows:

- Navigate to tab strip *Interface*.
- In the tree on the right side follow the path *ZENH\_FP\_IF\_TRQ* → *Form Interface* → *Import* to define a new Importing Parameter for the form interface.
- Click on button *Append Row* in the tool bar on the left side and create the following entry:

Parameter Name	<i>IS_PRINTOUT</i>
Type assignment	<i>TYPE</i>
Type name	<i>ZENH_S_PRINT_TRQ</i>

- 5) Save and activate the new form interface.

The final print interface should look as shown in the following picture:



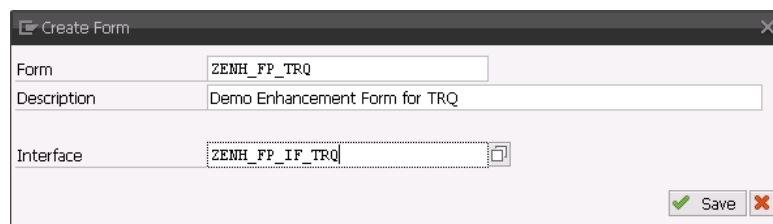
Picture: The form interface for the new form.

### 7.3.3 Creating the Adobe form

With the new print structure and form interface we finally can create the actual form. This is done by using Adobe LiveCycle Designer (see also section 7.2.1 and 7.2.2). Execute the following steps to create a simple Adobe Form:

- 1) Start transaction *SFP*, select radio button *Form* and in the menu bar select *Form Object* → *Create...* or press *F5*.
- 2) On the next popup screen enter the following data to specify the form:

Form	ZENH_FP_TRQ
Description	Demo Enhancement Form for TRQ
Interface	ZENH_FP_IF_TRQ



Picture: Creating the new form via transaction *SFP*.

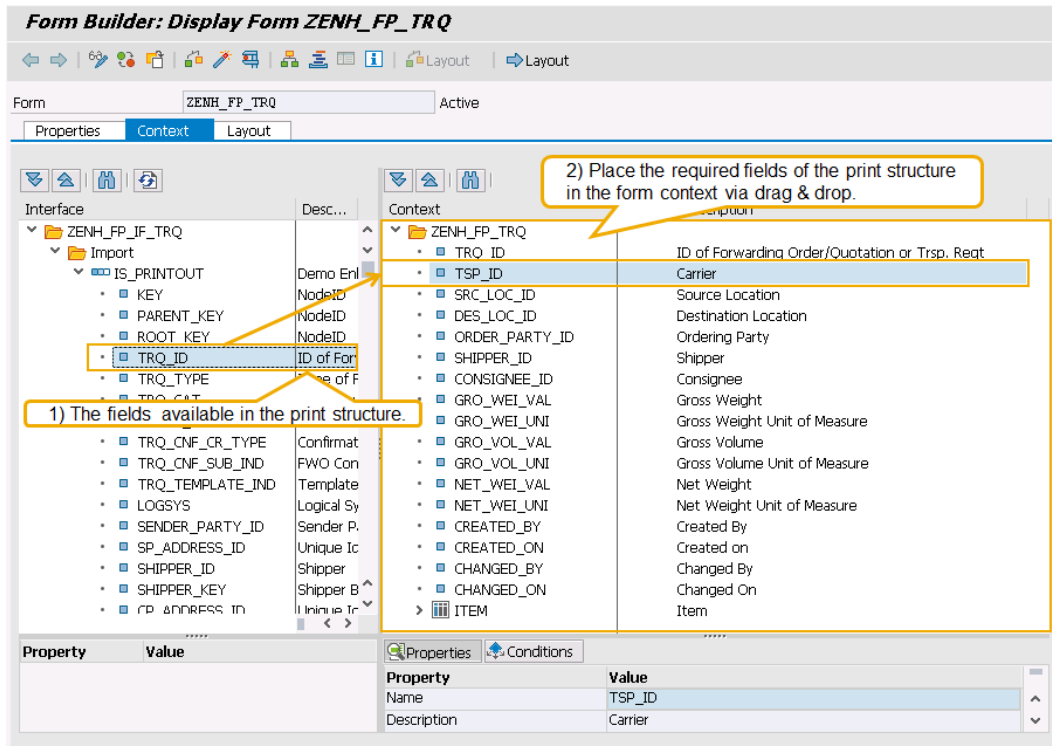
- 3) Click on button *Save (Enter)* on the popup screen and specify the package where to place the new form (create the form e.g. as a local object in package *\$TMP*).
- 4) On the next screen go to tab strip *Properties*. Make sure that the *Layout Type* is set to *Standard Layout*.
- 5) Now change to tab strip *Context*. In the tree on the left side follow the path *ZENH\_FP\_IF\_TRQ* → *Import* → *IS\_PRINTOUT*. Expand structure *IS\_PRINTOUT* via a double click). The context on the right side should be still empty. Only the context root *ZENH\_FP\_TRQ* should be available.

With this step the mapping between the form interface and the form is defined. Structure *IS\_PRINTOUT* represents the set of potential fields on the form while the context represents the list of actual fields that will be available when defining the layout of the form in the next step.

You can drag & drop attributes of the print structure *IS\_PRINTOUT* on the left side into the context of the form on the right side. Click on an attribute on the left side (keep mouse button pressed) and then drag it to the context on the left side (drop the very first one directly on the root *ZENH\_FP\_TRQ* of the context). For the example make the context contain the following:

Context attribute	Description
TRQ_ID	ID of Forwarding Order/Quotation or Trsp. Req
SRC_LOC_ID	Source Location
DES_LOC_ID	Destination Location
ORDER_PARTY_ID	Ordering Party
SHIPPER_ID Shipper	Shipper
CONSIGNEE_ID	Consignee
GRO_WEI_VAL	Gross Weight
GRO_WEI_UNI	Gross Weight Unit of Measure
GRO_VOL_VAL	Gross Volume
GRO_VOL_UNI	Gross Volume Unit of Measure
NET_WEI_VAL	Net Weight
NET_WEI_UNI	Net Weight Unit of Measure
CREATED_BY	Created By

CREATED_ON	Created on
CHANGED_BY	Changed By
CHANGED_ON	Changed On
ITEM	Item

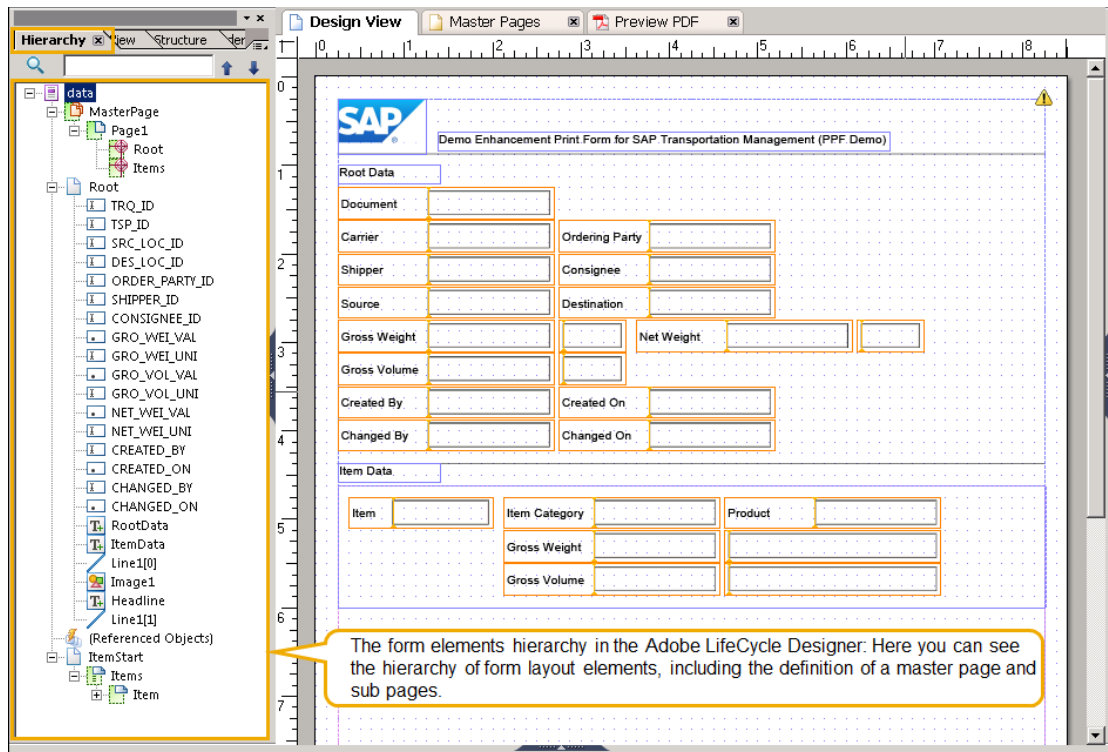


Picture: Specifying the context of a form.

6) Change to tab strip *Layout* (make sure that you have installed Adobe LiveCycle Designer as described in section 7.2.1). On the left side of the Adobe LiveCycle Designer navigate to tab strip *Hierarchy*.

- Define a master page with sub forms to represent different aspects of the form content. In the example form we have created a master page that has two content areas, one for the Root data and a second one for the Item data.
- Define a sub form for the Root data and assign it to the Root content area of the main page. Then define a sub form for the Item data and assign it to the Items content area of the main page

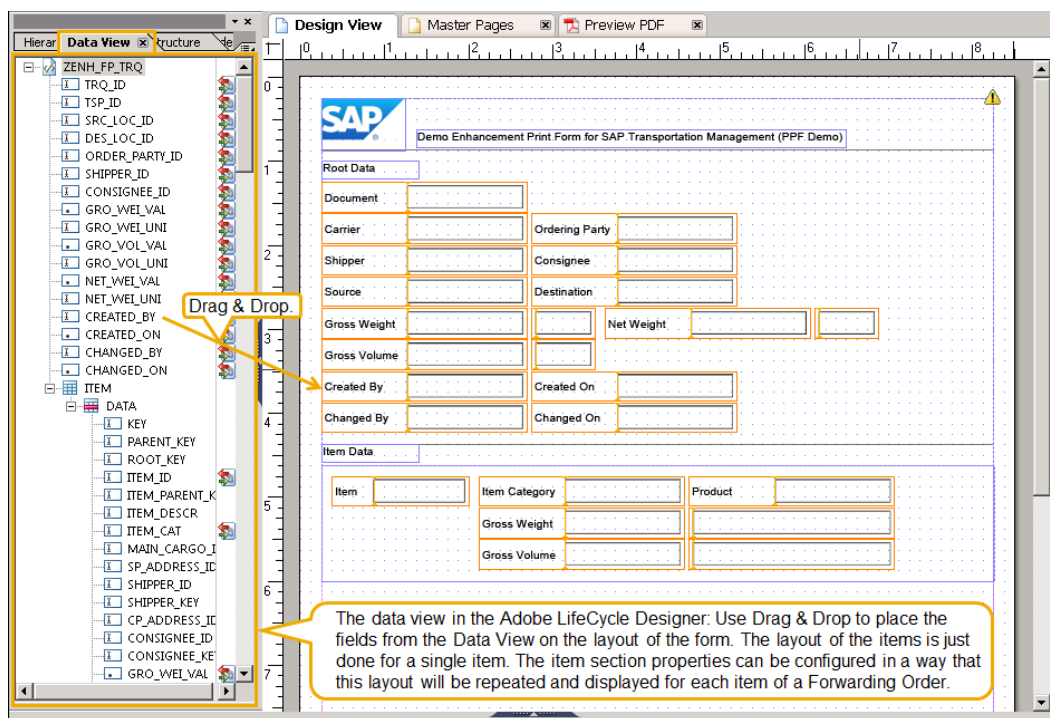
For more details on how to use the Adobe LiveCycle Designer review the corresponding help documentation in your system and take a look at the master page and sub form definitions of the standard forms provided by SAP to learn more. Moreover you can find further information about using Adobe LiveCycle Designer e.g. in the SCN (SAP Collaboration Network) or in the Internet in general. The following picture shows the final elements hierarchy of the example form.



Picture: The hierarchy of the example form.

- 7) On the left side of the Adobe LiveCycle Designer navigate to tab strip *Data View*.

Here the fields from the defined context are now available to be placed on the form layout. You can drag & drop any of the fields from the tree representation of the context on the left side *Data View* into the *Design View* in the middle section of the screen.



Picture: The data view of the example form.

- Drag & drop Root fields into the Root sub form area, arrange the fields as required and maintain field properties (e.g. define fields representing date/time information to be formatted accordingly as date/time, etc.).
- Drag & drop Item fields into the Item sub form area, arrange the fields as required and maintain field properties e.g. define fields representing date/time information to be formatted accordingly as date/time, etc.).
- Add additional required elements on the form like logos (in the picture above you can see that e.g. an SAP logo was added on the layout of the form), separators etc.
- The Items subpage displays the data of one or more items of the represented Forwarding Order document in this example. In the layout for the item data is defined for exactly one item. The item sub form is configured to be repeated for each data item of a Forwarding Document instance.

8) Save and activate the new form.

### 7.3.4 Creating required coding in the backend

With the previous step the definition of the form, its content and layout is finalized. In the next step ABAP coding is required to provide the data of a BO instance to the form and enable the communication with PPF and Output Management. For this we need to define a Printing Class and a BO Service Class as follows:

Define a Printing Class:

- 1) Start transaction *SE24* and create a new class that represents the Printing Class for the new form. This class will contain coding to fill the print structure of the form with data from a discrete BO instance. Use the following definitions for the example class:

Create class *ZCL\_ENH\_PRINTOUT\_FWO* which inherits from the super class */SCMTMS/CL\_PRINTOUT* (the TM Super Class for printing documents).

- 2) Add the following attributes and constants to the class:

Attribute	Level	Visibility	Typing	Associated Type
MO_SRMGR	Instance Attribute	Protected	Type Ref To	/BOBF/IF_TRA_SERVICE_MANAGER
MT_TRQ_ROOT	Static Attribute	Protected	Type	/SCMTMS/T_TRQ_ROOT_K
MT_TRQ_ITEM	Static Attribute	Protected	Type	/SCMTMS/T_TRQ_ITEM_K
MT_STAGE	Static Attribute	Protected	Type	/SCMTMS/T_TRQ_STAGE_K
MT_DOCREFERENCE	Static Attribute	Protected	Type	/SCMTMS/T_TRQ_DOCREF_K
MT_PARTY	Static Attribute	Protected	Type	/SCMTMS/T_TRQ_PARTY_K
MT_PARTY_LINK	Static Attribute	Protected	Type	/BOBF/T_FRW_KEY_LINK
MT_ITEMCONTENTID	Static Attribute	Protected	Type	/SCMTMS/T_TRQ_CONTENT_IDENT_K
MT_DG_INFO_ITEM	Static Attribute	Protected	Type	/SCMTMS/T_TRQ_DGO_INFO_K
MT_ITEMPARTY	Static Attribute	Protected	Type	/SCMTMS/T_TRQ_PARTY_K
MT_ITEMPARTY_LINK	Static Attribute	Protected	Type	/BOBF/T_FRW_KEY_LINK

- 3) Redefine class method *FILL\_PRINTSTRUCTURE*. This method will read the data from the BO instance to be printed and does the mapping between the BO data structure and the print structure. The example coding looks as follows:

```
METHOD fill_printstructure.
```

```
FIELD-SYMBOLS: <lt_printout_fwo> TYPE zenh_t_print_trq,
               <ls_trq_root>     TYPE /scmtms/s_trq_root_k.
```

```
DATA: ls_printout_fwo      TYPE zenh_s_print_trq,
      ls_printout_fwo_item TYPE /scmtms/s_trq_item_k,
      lo_message           TYPE REF TO /bobf/if_frw_message,
      lr_trq_root          TYPE REF TO /scmtms/s_trq_root_k,
      lr_trq_item          TYPE REF TO /scmtms/s_trq_item_k.
```

```
CREATE DATA et_printout LIKE <lt_printout_fwo>.
```

```
ASSIGN et_printout->* TO <lt_printout_fwo>.
```

\* Use a TRQ helper class method to read the relevant TRQ data

```
CALL METHOD /scmtms/cl_trq_helper=>read_nodes
```

```
EXPORTING
```

```
    it_key          = it_keys
```

```
IMPORTING
```

```
    et_root          = mt_trq_root
    et_item           = mt_trq_item
    et_stage          = mt_stage
    et_docreferencce = mt_docreferencce
    et_party          = mt_party
    et_party_link     = mt_party_link
    et_itemcontentid  = mt_itemcontentid
    et_itemdanggoodsinfo = mt_dg_info_item
    et_itemparty      = mt_itemparty
    et_itemparty_link = mt_itemparty_link.
```

\* Fill the print structure with the data of the BO instance

```
LOOP AT mt_trq_root REFERENCE INTO lr_trq_root.
```

```
    "mapping the root node information
```

```
    ls_printout_fwo-trq_id = lr_trq_root>trq_id.
```

```
    "Document Number
```

```
    ls_printout_fwo-tsp_id = lr_trq_root->tsp_id.
```

```
    "Carrier Party
```

```
    ls_printout_fwo-src_loc_id = lr_trq_root->src_loc_id.
```

```
    "Source Location
```

```
    ls_printout_fwo-des_loc_id = lr_trq_root->des_loc_id.
```

```
    "Destination Location
```

```
    ls_printout_fwo-order_party_id = lr_trq_root->order_party_id.
```

```
    "Ordering Party
```

```
    ls_printout_fwo-shipper_id = lr_trq_root->shipper_id.
```

```
    "Shipper
```

```
    ls_printout_fwo-consignee_id = lr_trq_root->consignee_id.
```

```
    "Consignee
```

```
    ls_printout_fwo-gro_wei_val = lr_trq_root->gro_wei_val.
```

```
    "Gross Weight
```

```
    ls_printout_fwo-gro_wei_uni = lr_trq_root->gro_wei_uni.
```

```
    "Gross Weight Unit of Measure
```

```
    ls_printout_fwo-gro_vol_val = lr_trq_root->gro_vol_val.
```

```
    "Gross Volume
```

```
    ls_printout_fwo-gro_vol_uni = lr_trq_root->gro_vol_uni.
```

```
    "Gross Volume Unit of Measure
```

```
    ls_printout_fwo-net_wei_val = lr_trq_root->net_wei_val.
```

```
    "Net Weight
```

```
    ls_printout_fwo-net_wei_uni = lr_trq_root->net_wei_uni.
```

```
    "Net Weight Unit of Measure
```

```
    ls_printout_fwo-created_by = lr_trq_root->created_by.
```



```

"Created By
ls_printout_fwo-created_on = lr_trq_root->created_on.
"Created on
ls_printout_fwo-changed_by = lr_trq_root->changed_by.
"Changed By
ls_printout_fwo-changed_on = lr_trq_root->changed_on.
"Changed On

"mapping the item information
CLEAR ls_printout_fwo-item.
LOOP AT mt_trq_item REFERENCE INTO lr_trq_item
  WHERE parent_key = lr_trq_root->root_key.
  CLEAR ls_printout_fwo_item.
  ls_printout_fwo_item-item_id      = lr_trq_item->item_id.
  ls_printout_fwo_item-item_cat    = lr_trq_item->item_cat.
  ls_printout_fwo_item-product_id = lr_trq_item->
                                     product_id.

  INSERT ls_printout_fwo_item
  INTO TABLE ls_printout_fwo-item.
ENDLOOP.

INSERT ls_printout_fwo INTO TABLE <lt_printout_fwo>.
ENDLOOP.

ENDMETHOD.

```

Define a BO Service Class:

- 4) Start transaction *SE24* and create a new class that represents the BO Service class. This class will contain coding to communicate with the PPF and Output Management. For this it will also reference and use the Printing Class created in the previous steps (see the coding example for BO Service Class method *PERSONALIZE\_DOC\_BY\_ABAP* in the following step 7). Use the following definitions for the example class:

Create class *ZCL\_ENH\_TRQ\_PPF\_SERVICE* which inherits from the super class */BOFU/CL\_PPF\_SERV\_FOR\_BO* (PPF Services for BO). Review section 7.4.1 again where the role of this PPF class is mentioned.

- 5) Add the following attributes and constants to the class:

Attribute	Level	Visibility	Typing	Associated Type
MV_AD_TRQ_FWO_PRINT	Instance Attribute	Protected	Type	PPFDTT
MV_AD_TRQ_FWO_PRINT_MAN	Instance Attribute	Protected	Type	PPFDTT
GC_AD_TRQ_FWO_PRINT	Constant	Private	Type	PPFDTT Initial value: ZENH_TRQ_FWO_PRINT
GC_AD_TRQ_FWO_PRINT_MAN	Constant	Private	Type	PPFDTT Initial value: ZENH_TRQ_FWO_PRINT_MAN

- 6) Implement a constructor for the class (the method *CONSTRUCTOR* is a public instance method). The example coding for the constructor looks as follows:

```

METHOD CONSTRUCTOR.
* call super constructor
CALL METHOD super->constructor.

* configure class with default action definitions
mv_ad_trq_fwo_print      = gc_ad_trq_fwo_print.

```

```
mv_ad_trq_fwo_print_man = gc_ad_trq_fwo_print_man.
```

```
ENDMETHOD.
```

7) Redefine class method *PERSONALIZE\_DOC\_BY\_ABAP* with the following coding:

```
METHOD personalize_doc_by_abap.
```

```
DATA: lo_printout      TYPE REF TO /scmtms/cl_printout,
      ls_key           TYPE /bobf/s_frw_key,
      lt_keys          TYPE /bobf/t_frw_key,
      lv_document_number TYPE /scmtms/trq_id.
```

```
* PPF specific variables
```

```
DATA: lv_db_key      TYPE /bobf/conf_key.
```

```
DATA: lr_message TYPE REF TO /bobf/if_frw_message.
```

```
* Start. Via this container, determine the Root Node ID
```

```
CALL METHOD io_container->get_db_key
```

```
RECEIVING
```

```
result = lv_db_key.
```

```
* Determine BO Name and Node ID from PPF container
```

```
ls_key-key = lv_db_key.
```

```
APPEND ls_key TO lt_keys.
```

```
CASE is_ppf_act-ppf_action.
```

```
WHEN mv_ad_trq_fwo_print OR mv_ad_trq_fwo_print_man.
```

```
" Forwarding Order: create instance of class for document
```

```
" to be printed.
```

```
CREATE OBJECT lo_printout TYPE zcl_enh_printout_fwo.
```

```
WHEN OTHERS.
```

```
ENDCASE.
```

```
IF lo_printout IS BOUND.
```

```
* Fill data and print document.
```

```
CALL METHOD lo_printout->print_document
```

```
EXPORTING
```

```
it_keys = lt_keys
```

```
ip_function_name = ip_function_name
```

```
ip_form_name = ip_form_name
```

```
is_outputparams = is_outputparams
```

```
IMPORTING
```

```
es_formoutput = es_formoutput
```

```
es_joboutput = es_joboutput
```

```
eo_message = lr_message
```

```
ev_document_number = ev_document_number
```

```
CHANGING
```

```
cs_docparams = cs_docparams
```

```
cp_document_title = cp_document_title.
```

```
ENDIF.
```

```
IF ( lr_message IS NOT INITIAL ).
```

```
CALL METHOD io_message->add
```

```
EXPORTING
```

```
io_message = lr_message.
```

```
ENDIF.
```

```
ENDMETHOD.
```

- 8) Redefine class method *DETERMINE\_PRINTER\_BY\_ABAP* with the following coding:

```
METHOD DETERMINE_PRINTER_BY_ABAP.
```

```

* Get the default printer of the actual user.
* Uses function 'SUSR_GET_USER_DEFAULTS' in order to retrieve
* printer-related data from table with entries of type USDEF.
get_printer_for_logon_user(
  CHANGING
    cs_printer =      et_data ).

* A device is needed. Therefore set a default.
IF et_data-device IS INITIAL.
  et_data-device = 'A000'.
ENDIF.
```

```
ENDMETHOD.
```

- 9) Redefine class method */BOFU/IF\_PPF\_SERV\_FOR\_BO~GET\_PROFILES*. It will take care of determining the correct PPF Action Profile. Use the following example coding:

```
METHOD /BOFU/IF_PPF_SERV_FOR_BO~GET_PROFILES.
```

```

DATA: lo_trq_srv_mgr TYPE REF TO /bobf/if_tra_service_manager,

      lt_trq_root     TYPE /scmtms/t_trq_root_k,
      lt_trq_root_bi  TYPE /scmtms/t_trq_root_k,
      lr_s_trq_root   TYPE REF TO /scmtms/s_trq_root_k,

      lt_trq_types    TYPE /scmtms/t_trq_ty,
      ls_trq_type     LIKE LINE OF lt_trq_types,

      lt_key           LIKE it_key,
      lt_key_deleted  LIKE it_key_deleted,
      ls_key           LIKE LINE OF lt_key,

      lt_ppf_profile  TYPE /bofu/t_ppf_prof,

      lt_data          LIKE et_data,
      ls_data          LIKE LINE OF et_data.
```

```

* Clear return parameters
CLEAR et_data.
```

```

* Retrieve TRQ Root. The setting how to determine the relevant
* PPF Action Profiles is stored at the root node.
* For created/changed instances we need the current image, for
* deleted instances we need the before image
lo_trq_srv_mgr = /bobf/cl_tra_serv_mgr_factory=>
  get_service_manager( /scmtms/if_trq_c=>sc_bo_key ).
```

```

lo_trq_srv_mgr->retrieve(
  EXPORTING
    iv_node_key      = /scmtms/if_trq_c=>sc_node-root
    it_key            = it_key
  IMPORTING
```

```

        et_data                = lt_trq_root ).

lo_trq_srv_mgr->retrieve(
    EXPORTING
        iv_node_key            = /scmtms/if_trq_c=>sc_node-root
        it_key                  = it_key_deleted
        iv_before_image        = abap_true
    IMPORTING
        et_data                = lt_trq_root_bi ).

INSERT LINES OF lt_trq_root_bi INTO TABLE lt_trq_root.

* Retrieve all TRQ types.
CALL METHOD /scmtms/cl_trq_helper_cust=>get_trqtype_all
    IMPORTING
        et_trqtype = lt_trq_types.

* Retrieve PPF Action Profiles of current output agent
conf_get_valid_ppf_prof(
    EXPORTING
        is_ppf_conf          = is_ppf_conf
        iv_kind_of_profile   = iv_kind_of_profile
        io_message           = io_message
    IMPORTING
        et_data              = lt_ppf_profile ).

* Determine relevant PPF Action Profiles Separate instances were
* the Output Profile is specified explicitly and were the PPF
* Action Profiles shall be determined automatically according to
* the settings in the output management adapter configuration

LOOP AT lt_trq_root REFERENCE INTO lr_s_trq_root.

    READ TABLE lt_trq_types INTO ls_trq_type
        WITH TABLE KEY type = lr_s_trq_root->trq_type.
    IF sy-subrc NE 0.
        CLEAR ls_trq_type.
    ENDIF.

    IF ls_trq_type-ppf_profile_auto = abap_true.
        " PPF Action profiles are to be determined automatically
        " by output management adapter configuration
        " Collect key for automatic determination
        CLEAR ls_key.
        ls_key-key = lr_s_trq_root->key.

        READ TABLE it_key WITH TABLE KEY key_sort
            COMPONENTS key = lr_s_trq_root->key
            TRANSPORTING NO FIELDS.
        IF sy-subrc = 0.
            INSERT ls_key INTO TABLE lt_key.
        ELSE.
            INSERT ls_key INTO TABLE lt_key_deleted.
        ENDIF.
    ELSE.
        " Relevant PPF Action profile is stored in transportation
        " request root
        IF ls_trq_type-ppf_profile IS NOT INITIAL.
            READ TABLE lt_ppf_profile
                WITH TABLE KEY ppf_profile = ls_trq_type-ppf_profile

```

```

        TRANSPORTING NO FIELDS.
    IF sy-subrc = 0.
        CLEAR ls_data.
        ls_data-key          = lr_s_trq_root->key.
        ls_data-ppf_profile = ls_trq_type-ppf_profile.
        ls_data-appl_key     = lr_s_trq_root->key.
        INSERT ls_data INTO TABLE et_data.
    ENDIF.
ENDIF.
IF ls_trq_type-ppf_profile_add IS NOT INITIAL.
    READ TABLE lt_ppf_profile
        WITH TABLE KEY ppf_profile =
            ls_trq_type-ppf_profile_add
        TRANSPORTING NO FIELDS.
    IF sy-subrc = 0.
        CLEAR ls_data.
        ls_data-key          = lr_s_trq_root->key.
        ls_data-ppf_profile = ls_trq_type-ppf_profile_add.
        ls_data-appl_key     = lr_s_trq_root->key.
        INSERT ls_data INTO TABLE et_data.
    ENDIF.
ENDIF.
ENDIF.
ENDLOOP.

" For all instances in LT_KEY, LT_KEY_DELETED, determine the
" relevant PPF Action Profile automatically
IF lt_key IS NOT INITIAL OR lt_key_deleted IS NOT INITIAL.
    super->/bofu/if_ppf_serv_for_bo~get_profiles(
        EXPORTING
            is_ppf_conf          = is_ppf_conf
            iv_kind_of_profile = iv_kind_of_profile
            it_key               = lt_key
            it_key_deleted       = lt_key_deleted
            io_message           = io_message
        IMPORTING
            et_data              = lt_data ).

    " Merge determined PPF Action Profiles
    INSERT LINES OF lt_data INTO TABLE et_data.
ENDIF.

ENDMETHOD.

```

## 7.4 Output Management Adapter and PPF Configuration

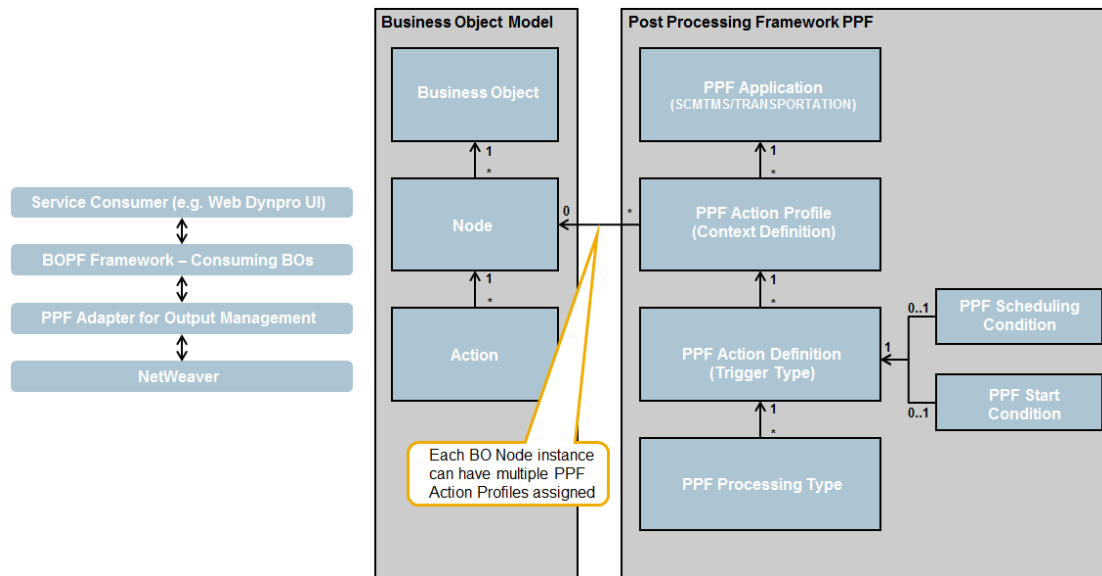
With the new form and the backend coding in place we can now create the required Post Processing Framework (PPF) and Output Management Adapter configuration to get the new example form displayed in the print preview and printed out on paper. But first we take a general look at some of the involved steps and concepts which will help understanding the configuration example.

### 7.4.1 Overview and general concepts

Output Management enables not only printing but also other output types like fax and email and it is also used in the context of alerting for business events, workflow items, A2A and B2B communication, BI Data Uploading or triggering events to SAP Event Management Systems.

The Output Management Adapter is a component which was developed in the Business Suite Foundation layer (SAP\_BS\_FND) and enables output functionality for a given BO node of a

BOPF-implemented Business Object (e.g. /SCMTMS/TRQ which can e.g. represent a Forwarding Order). It enables the Post Processing Framework (PPF) to connect with individual BO nodes and the BO node-specific assignment of PPF Action Profiles.

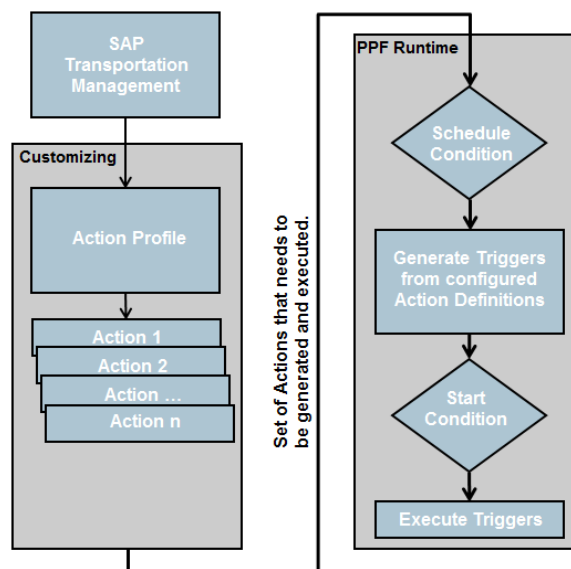


Picture: The relation between BOPF BO nodes and PPF.

PPF enables the generic execution of functions and processes and provides a unified interface to corresponding actions. Actions can be considered as business tasks. They can trigger e.g. conventional output like print, fax or email but generally speaking it can trigger any method call (e.g. sending A2A or B2B messages).

Available actions are configured and collected in so called Action Profiles. Such actions can then be executed manually, e.g. via the User Interface of a Business Object (in the later example we'll see an action that can be executed manually to print the example form of section 7.3 from the Forwarding Order UI).

The execution of an action can also be made dependent on the content of a business document and executed automatically in the backend. Moreover you can define conditions (schedule and start conditions) that determine why and when an action is executed. Such conditions can take into consideration e.g. the content of a business document to be printed.



Picture: PPF Actions and their condition based execution.

The application layer (e.g. SAP TM) uses the Output Management Adapter for communication with the PPF to execute outputs. For this execution PPF must be provided with the application-specific data that can be passed to PPF via a number of BADIs which have to be implemented by the consuming application.

All relevant PPF-Exits on Action-Level are permanently connected to the corresponding methods of the Agent Class (BO Service Class → see section 7.3.4) that can be specified in Output Management customizing (see section 7.4.4). The implementation of such an Agent Class always follows the same interface, i.e. each such class implementation inherits from standard super class `/BOFU/CL_PPF_SERV_FOR_BO` which defines the common structure for accessing PPF functionality from an application. This allows a unified and easy implementation of any Agent Class to be used by the application.

The application specific Agent Class implementation redefines the corresponding methods and maybe adds additional ones for its individual functionality (e.g. additional helper methods are added to further structure the implementation). The logic can be implemented of course in plain ABAP or by using BRF+-Conditions. When you take a look at the mentioned super class you will find a number of “method pairs”, i.e. one with the postfix `_BY_ABAP` (for ABAP based logic) and the same method with postfix `_BY_FDT` (for BRF+-based logic → *FDT* is the old name of BRF+ and stands for *Formula and Derivation Tool*). Some more detailed aspects of the Agent Class:

#### Action Profile Determination

You can assign multiple Action Profiles to a BO node in the Output Management Adapter customizing and also in the application layer. But at runtime PPF must be provided with the final Action Profile to be used. A Forwarding Order with Transportation Mode *Road* may require a different Action Profile than a Forwarding Order with Transportation Mode *Air* or *Ocean*.

The Agent Class method `GET_PROFILES` can be redefined and implemented with coding that determines the correct and relevant Action Profile depending e.g. on the information of the business document for which available actions shall be determined. The resulting Action Profile is then provided to PPF for further processing.

#### Printer Determination

Output Management provides alternative ways of how to determine a printer for external communication outputs. The printer determination follows a sequence:

1. The printer is determined via the implementation of the Printer Determination BAdI `PRINTER_DETERM_PPF`. This is simply done by redefining the Agent Class method `DETERMINE_PRINTER_BY_ABAP` (or `_BY_FDT`).
2. You can place a static printer configuration in the *Processing Details* of the conditions configuration for a PPF Action.
3. Define a printer in your User Profile settings (in the example implementation of Agent Class method `DETERMINE_PRINTER_BY_ABAP` you can actually see how the printer specified in the User Profile settings can be read (see step 8 of section 7.3.4).

#### Partner and Language Determination

Actions might be partner-dependent. In the Action Definition you can mark the action to be partner-dependent and specify a partner role. To compare this assigned role with a role specified in the business document to be processed you can redefine and implement Agent Class method `GET_DOCUMENT_PARTNERS`.



In case you need to execute an action for external communication in a language different to the default language you can redefine and implement the Agent Class methods `GET_LANGUAGE_FOR_EXT_COMM` or `GET_LANGU_4_EXT_COMM_BY_ABAP` (the latter one actually calls the last one by default).

### Generating and executing Actions

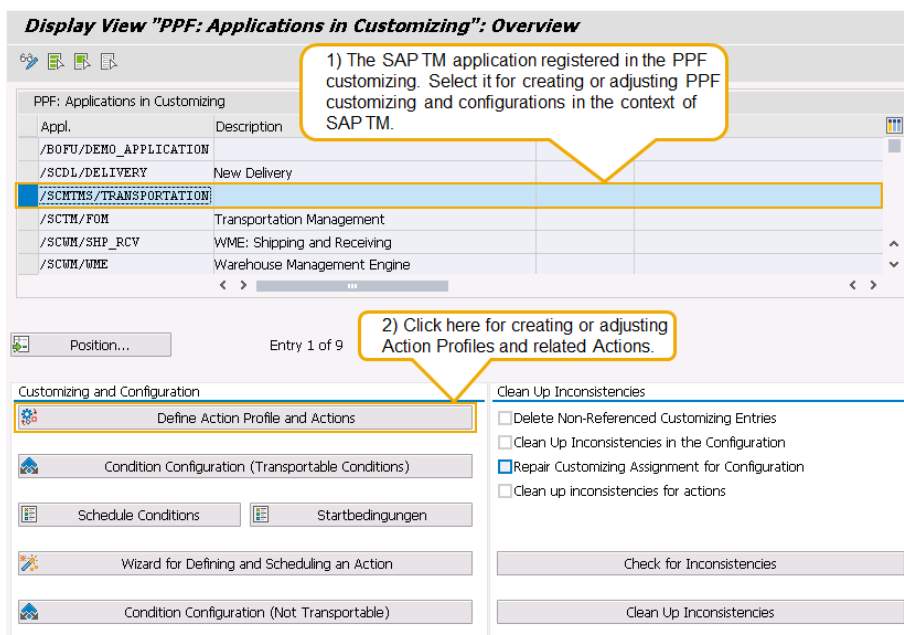
PPF outputs are executed in two steps. First the relevant actions are generated. The application can make use of the already mentioned scheduling conditions to determine which actions are allowed to be triggered in the context of a business scenario. If the schedule condition of an action is fulfilled, it is scheduled for (manual or automatic) execution. In the second step a generated action is executed in case the configured start condition is fulfilled (see section 7.4.3 for some more details).

## 7.4.2 Creating a PPF Action Profile and Action Definitions

In section 7.3 we defined and implemented a new print form for printing data of Forwarding Orders (BO /SCMTMS/TRQ). The following configuration example will create all required settings for PPF and the Output Management Adapter that will allow displaying the new print form in a preview within the Forwarding User Interface and print it from there. For this purpose we will configure a PPF action that can be triggered manually.

Each application that makes use of PPF is registered in the PPF customizing. SAP TM is registered there under the application name `/SCMTMS/TRANSPORTATION`. This application entry serves as a container for all PPF settings that will be done in the context of SAP TM.

- 1) Start transaction `SPPFCADM` or start customizing transaction `SPRO` and follow the path *Cross-Application Components → Processes and Tools for Enterprise Applications → Reusable Objects and Functions for BOPF Environment → PPF Adapter for Output Management → Maintain PPF Settings*.
- 2) On the first screen select application `/SCMTMS/TRANSPORTATION` and then click on button *Define Action Profile and Actions*.



Picture: Maintaining PPF Settings - Initial screen.

- 3) On the next screen switch into change mode (`Ctrl+F1`). Mark entry *Action Profile* in the *Dialog Structure* tree and click on button *New Entry* to create a new Action Profile with the following data:

Field	Value
Action Profile	ZENH_TRQ_FWO_PRINT
Description	Enhancement Action Profile for TRQ
Category of Object Type	Persistent Class
Object Type Name	/BOFU/CL_PPF_CONTAINER
Context Class	/BOFU/CL_PPF_CONTEXT

This new Action Profile will now be used to define actions for printing the example form.

- 4) Mark entry *Action Definition* in the *Dialog Structure* tree and click on button *New Entry* to create a new Action Definition *ZENH\_TRQ\_FWO\_PRINT\_MAN* with the following data:

Tab strip	Field	Value
	Action Definition	ZENH_TRQ_FWO_PRINT_MAN
	Description	Enhancement TRQ Manual Print Action
<b>Action Definition</b>	<b>Action Settings</b>	
	Processing Time	Processing using selection report
	Processing Times Not Permitted	No Restrictions
	Schedule Automatically	X
	Changeable in Dialog	X
	Delete After Processing	[blank]
	Executable in Dialog	X
	<b>Action Determination and Action Merging</b>	
	Determination Technology	Determination Using Conditions that Can Be Transported
	Rule Type	Conditions Using Business Add In (BAI)
	Action Merging	Set Highest Number of Processed Actions
<b>Action Description</b>		
	Description	Enhancement TRQ Manual Print Action
<b>Action Merging</b>	<b>Number of Unprocessed Actions</b>	
	One Unprocessed Action for each Action Definition	X
	<b>Number of Processed Actions</b>	
	Allow Any Number of Actions	X

An Action Definition represents the Meta Data of a business task (e.g. printing out a form or sending a service message). A few remarks on the semantics of the different properties defined for such an Action Definition. The properties allow quite some variations in defining the way how an action will behave at runtime.

- Processing time: An action can be configured for immediate processing, when the related business document is posted or later using a selection report (which will then process the action).
- Processing times that are not permitted: This property allows excluding processing times that make no sense for the action. You may want to print a Forwarding Order e.g. only after it has been saved in a consistent status and not immediately while e.g. editing it. So you could exclude processing time option "immediate processing" here.
- Schedule automatically: With the flag set, the action is scheduled automatically if the schedule condition is fulfilled. Otherwise the action appears in the work list and can be scheduled manually by the user.
- Sort order for display: Allows defining a displaying sequence of actions in an action profile (e.g. when actions will be manually executed via the User Interface this can define the display sequence of actions).
- Delete after processing: The action is executed and then deleted.

- Changeable in dialog: Determines whether after automatic determination you are allowed to make changes and repeat the action definition manually.
- Executable in dialog: The action can be executed during document editing, even if a posting took not yet place at that point in time.
- Partner-dependent: If set, a partner determination is performed. Actions can be assigned to a specific group of partners or persons.
- Partner Function: Specifies the default function for a partner determination.
- Action Determination and Action Merging: During Action Determination PPF checks the Schedule Condition of all configured Action Definitions. Action Merging is also done during Action Determination. You can e.g. define with option “1 successful action per action definition” that each action will not be executed again after the first successful execution.

To create Action Definition ZENH\_TRQ\_FWO\_PRINT again click on button *New Entry* and enter the following data:

Tab strip	Field	Value
	Action Definition	ZENH_TRQ_FWO_PRINT
	Description	Enhancement TRQ Print Action
<b>Action Definition</b>	<b>Action Settings</b>	
	Processing Time	Processing when saving document
	Processing Times Not Permitted	No Restrictions
	Schedule Automatically	X
	Changeable in Dialog	X
	Delete After Processing	[blank]
	Executable in Dialog	X
	<b>Action Determination and Action Merging</b>	
	Determination Technology	Determination Using Conditions that Can Be Transported
	Rule Type	Conditions Using Business Add In (BAI)
	Action Merging	Set Highest Number of Processed Actions
<b>Action Description</b>		
	Description	Enhancement TRQ Print Action.
<b>Action Merging</b>	<b>Number of Unprocessed Actions</b>	
	One Unprocessed Action for each Action Definition	X
	<b>Number of Processed Actions</b>	
	Allow Any Number of Actions	X

- 5) Double click on entry *Action Definition* in the Dialog Structure tree. You should now see the two Action Definitions displayed in a list. For both Action Definitions execute the following steps.
- 6) Mark an Action Definition in the list and double click on entry *Processing Type* in the Dialog Structure tree. Then click on button *New Entry* to define the Processing Type for the selected Action Definition with the following data:

In the list *Permitted Processing Types of Action* use the F4-Help in column *Assignment / Change Using Value Help in List* and select value *External Communication*.

- 7) You can now see three tab strips which allow specifying further details for the Processing Type. On tab strip *Document* enter the following data:

Field	Value
Form Name	ZENH_FP_TRQ
Form Type	PDF-Based Forms
Format	/BOFU/PPF_STANDARD
Personalization Type	Recipient-Specific Variable Replacement

The Processing Type defines the technical realization of an Action Definition. With the example settings above you define an *External Communication* that enables the printing of the specified PDF-Based form (see following picture).

Picture: Settings for *External Communication*.

Picture: Detail settings for *Method Call*.

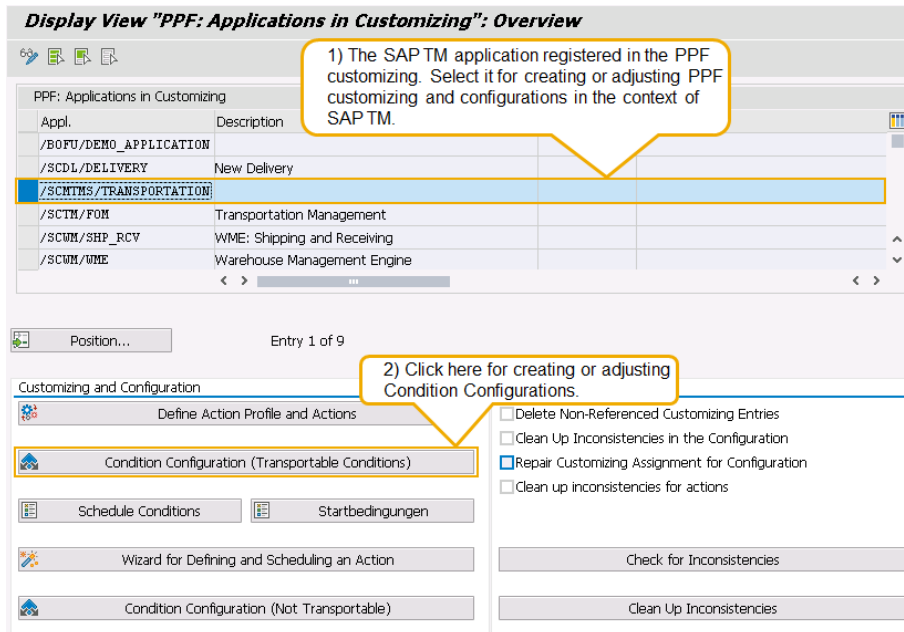
Besides the option *External Communication* there are further options available like e.g. *Method Call* for invoking a method containing the action's functionality. As you can see in the example screen above */SCMTMS/TEND\_TPNORD\_QUO\_CRTR (Creating Tendering Request B2B)* is chosen. This actually represents an available filter value for *BAdI EXEC\_METHODCALL\_PPF* which provides a method Execute that you can implement. The corresponding coding is called when executing the related action.

- 8) Double click on entry *Action Definition* in the Dialog Structure tree again to return to the list of Action Definitions and repeat steps 6 and 7 for the second Action Definition.
- 9) In the list of Action Definitions mark Action Definition *ZENH\_TRQ\_FWO\_PRINT* as inactive (will not be used for the time being but maybe in a future example).
- 10) Save your settings. The Action Profile and assigned actions are now configured to be used in the next configuration steps.

### 7.4.3 Creating PPF Conditions

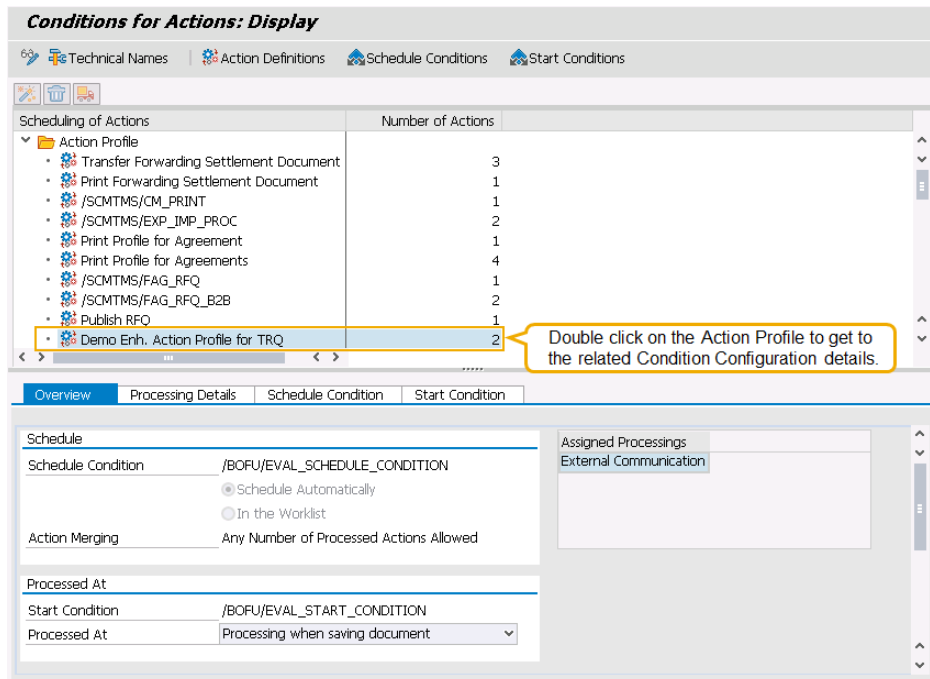
In the next steps we create the required condition configuration. The example condition configuration is done with the following steps:

- 1) On the first screen select application /SCMTMS/TRANSPORTATION and then click on button *Condition Configuration (Transportable Conditions)*.



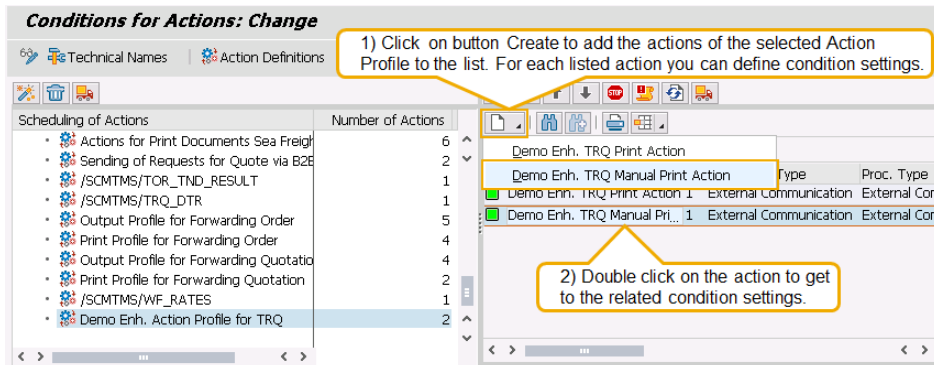
Picture: Maintaining PPF Settings - Initial screen.

- 2) On the next screen double click on entry *Enhancement Action Profile for TRQ* (which represents the action profile that was created before) in the list on the left side. Then switch to change mode (*Ctrl+F1*).



Picture: Navigating to the Condition Configuration of an Action Profile.

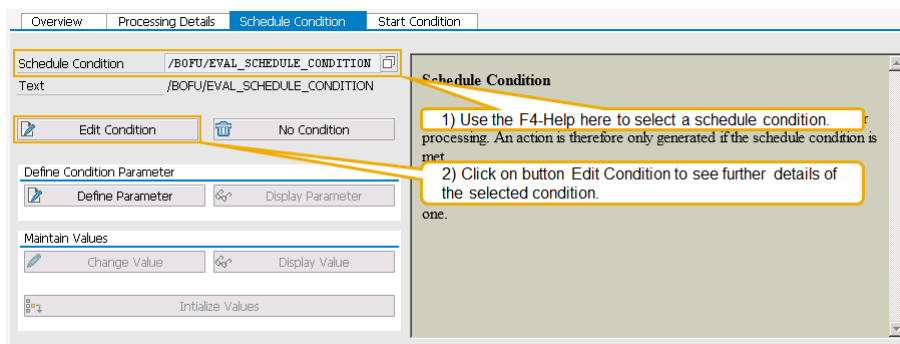
- 3) On the right side of the screen you can first of all see an empty action list. Click on button *Create* in the toolbar of this list. In the popup menu you can find all the Action Definitions that belong to the selected Action Profile. Select the listed action definitions from the popup menu one after another to add all of them to the list.



Picture: Taking over Action Definitions for Condition Configuration.

For both action definitions execute the following steps to configure the related Condition settings.

- 4) Double click on an Action Definition in the list on the right side. Navigate to tab strip *Schedule Condition* below the list. Make sure that you are in edit mode when creating the following condition settings. Use the F4-Help of field *Schedule Condition* to select the value */BOFU/EVAL\_SCHEDULE\_CONDITION*.



Picture: Defining the Schedule Condition for an action.

The Schedule Conditions decide whether actions should be scheduled for processing. An action is therefore only generated if the schedule condition is met. This is checked during Action Determination. In our example a standard default Schedule Condition is used. You could also create your own schedule condition here with your own logic.

The value entered in field *Schedule Condition* on the mentioned screen actually represents a filter value for BADL *EVAL\_SCHEDULECOND\_PPF*. This BADL provides a method *EVALUATE\_SCHEDULE\_CONDITION* that can then be implemented in a filter-specific class. The method interface provides all required data (e.g. the content of a business document) that allows realizing the corresponding condition logic.

At runtime the method implementation will check whether the schedule condition is fulfilled. The data that this decision is based on is provided with importing parameter *IO\_CONTEXT* (which can e.g. contain the document data in the print form example). Via exporting parameter *EP\_RC* the method returns the check result (where value 0 indicates "condition fulfilled" and a value <> 0 indicates "condition not fulfilled").

- 5) Now navigate to tab strip *Start Condition* (the screen looks almost the same as shown for the Schedule Condition above). Use the F4-Help of field *Start Condition* to select the value */BOFU/EVAL\_START\_CONDITION*.

The start condition is checked before the action is executed. The action is only executed when the start condition has been fulfilled. As for the schedule condition, we use a standard default start condition for the example. You could also create your own start condition here with your own logic.

The value entered in field *Start Condition* on the mentioned screen actually represents a filter value for BAdI *EVAL\_STARTCOND\_PPF*. This BAdI provides a method *EVALUATE\_START\_CONDITION* that can then be implemented in a filter-specific class. The method interface provides all required data (e.g. the content of a business document) that allows realizing the corresponding condition logic.

Just like mentioned for the schedule condition, the start condition check method has an importing parameter *IO\_CONTEXT* and a corresponding exporting parameter *EP\_RC* that returns the check result (where value 0 indicates "condition fulfilled" and a value  $\neq 0$  indicates "condition not fulfilled").

- 6) Save your settings. The condition configuration is now complete and ready to be used in the next steps.

Remark: As you can see in this example the PPF Conditions are primarily based on BAdIs that you can implement to realize the check logic of a condition. This logic can take into account e.g. document data like in the print form example. Depending on the value of a document attribute the condition is fulfilled and the related action is scheduled or started. So the condition technology provided with the BRF+-Framework is not directly involved. Of course you can make use of BRF+-Conditions within the BAdI method implementations. An example how to integrate BRF+ into your own implementations is described in section 4.3.7.

## 7.4.4 Maintaining Output Management Adapter Settings

In this step the output management adapter settings are done. These settings determine the output for a given Business Object (BO) node. Remember the example Printing Class and the BO Service Class that was created in section 7.3.4. The latter includes the usage of the first one and will now be used in the Output Management Adapter Settings to connect the BO (which provides the relevant document data) with the corresponding PPF configuration and settings that were created in the previous sections.

To finalize the example configuration execute the following steps.

- 1) Start customizing transaction *SPRO* and follow the path *Cross-Application Components* → *Processes and Tools for Enterprise Applications* → *Reusable Objects and Functions for BOPF Environment* → *PPF Adapter for Output Management* → *Maintain Output Management Adapter Settings*.
- 2) On the first screen double click on entry *PPF Output Agents for BO Nodes* in the Dialog Structure tree and then click on button *New Entries*. Enter the following data:

Field	Value
Business Object	/SCMTMS/TRQ
Node	ROOT
Output Agent	ZENH_TRQ_STANDARD
Agent Class for Node	ZCL_ENH_TRQ_PPF_SERVICE
Enable	X

With this entry you specify a PPF Output Agent that identifies the required combination or connection of a BO node and a PPF Action Profile (whose actions are provided with the BO data and will make of it during execution).



**Change View "PPF Output Agents for BO Nodes": Details**

New Entries

Dialog Structure

- PPF Output Agents for BO Nodes
  - Assign PPF Profiles
    - Action Settings
      - Processing Type Set
      - Assign Transient Action
  - Direct Output Agents (w/o PPF)
  - Nodes for Before Image

1) Double click to display available Output Agents. Then double click on button *New Entries* to create a set up a new Output Agent.

Business Object: /SCMTMS/TRQ  
Node: ROOT  
Output Agent: ZENH\_TRQ\_STANDARD

PPF Output Agents for BO Nodes

Agnt Class for Node: ZCL\_ENH\_TRQ\_PPF\_SERVICE  
☒ Enable  
☐ Ind. Child Chgs  
Appl. ID for Node:   
Func. for Act. Prof.:   
Func. for Partners:   
Catig ID for Node:   
Created By: SCMSUPPORT  
Created On: 19.10.2012 17:13:32  
Changed By: SCMSUPPORT  
Changed On: 19.10.2012 17:13:32

The settings for the new Output Agent.

Picture: Setting up a new Output Agent.

- 3) With the next step assign the Action Profile (created in section 7.4.2) to the PPF Output Agent created in the previous step. In the Dialog Structure tree double click on entry *Assign PPF Profiles* and enter the following data:

Field	Value
Action Profile	ZENH_TRQ_FWO_PRINT
Enable	X
Output Type	Has Uncritical o/p: Process after Commit (background)
Preprocess Actions	[space]
Application for Action Profile	

**Change View "Assign PPF Profiles": Details**

New Entries

Dialog Structure

- PPF Output Agents for BO Nodes
  - Assign PPF Profiles
    - Action Settings
      - Processing Type Set
      - Assign Transient Action
  - Direct Output Agents (w/o PPF)
  - Nodes for Before Image

1) Double click to display assigned Action Profiles. Then double click on button *New Entries* to assign an Action Profile to the given Output Agent.

Business Object: /SCMTMS/TRQ  
Node: ROOT  
Output Agent: ZENH\_TRQ\_STANDARD  
Action Profile: ZENH\_TRQ\_FWO\_PRINT

Assign PPF Profiles

Action Profile Des.: Demo Enh. Action Profile for TRQ  
☒ Enable  
Output Type: Has Uncritical o/p: Process after Commit (background)  
Create DB Image: Current Node  
☐ Preprocess Actions  
Agnt Class for Node: ZCL\_ENH\_TRQ\_PPF\_SERVICE  
Appl. ID for Node:   
Appl. for Act. Prof.:   
Created By: SCMSUPPORT  
Created On: 19.10.2012 17:13:32  
Changed By: SCMSUPPORT  
Changed On: 19.10.2012 17:13:32

The settings for assigning the Action Profile

Picture: Assign an Action Profile to the Output Agent.

- 4) Save your settings.

### 7.4.5 Preparing an example print document

With the following steps you finally prepare an example document (in this example a Forwarding Order) for which you can display the developed PDF-based form in the print preview of the Forwarding Order User Interface. Moreover you can print the example document from there by executing the now available action.

If you want to print the form you should make sure that you have maintained a corresponding output device for your user. When reviewing method *DETERMINE\_PRINTER\_BY\_ABAP* of the BO Service Class (Agent Class) in section 7.3.4 you can see that the implementation calls a method to determine a printer that is maintained in your user settings and if not just returns with a default output device.

Open the menu path *System → User Profile → Own Data* and navigate to the tab strip *Defaults*. In section *Spool Control*, enter the output device name (i.e. a printer name) in field *Output Device* by selecting an existing one via the F4-Help. Also set the flag *Output Immediately*.

Remember that the example print form is based on the BO */SCMTMS/TRQ* that provides its Root node and Item node data as the content for the form (the header and item data of a Forwarding Order). For testing the example form proceed as follows:

- 1) Start customizing transaction *SPRO* and follow the path *SAP Transportation Management → Forwarding Order Management → Forwarding Order → Define Forwarding Order Types*.

Switch into change mode (*Ctrl+F4*) Create a new Forwarding Order Type (click on button *New Entries*) or choose an existing one from the list (which you are allowed to use and change).

- 2) In section *Process Control / Business Object Mode* on the main screen enter the output (action) profile *ZENH\_TRQ\_FWO\_PRINT* that was created in section 7.4.1, step 3. Here the term output profile is used to nevertheless refer to a PPF Action Profile.

Any Forwarding Order that you create now with this type will now make use of this output profile with its settings and the example form.

Picture: Specifying the Output Profile in the Forwarding Order Type.

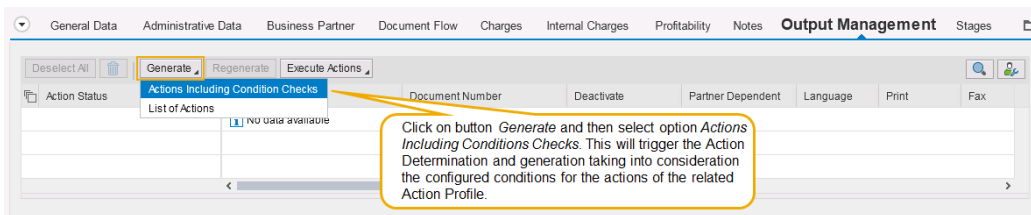
- 3) Maintain all other mandatory or required settings and save the Forwarding Order Type settings.

- 4) Create a new Forwarding Order with the corresponding Forwarding Order Type.

You can do this via the TM UI in the NetWeaver Business Client (NWBC) or the standalone UI triggered via the user menu path *User Menu for [user] → Forwarding Order Management → Forwarding Order → Create Forwarding Order*.

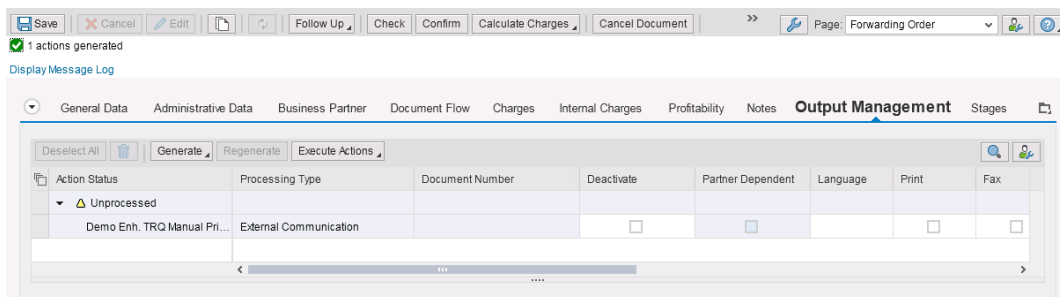
Provide all mandatory fields on tab strip *General Data*. Define the required business partners on tab strip *Business Partner* and maintain the required location information on tab strip *Location and Dates / Times*. Moreover, maintain item data in the item list of your example document (e.g. an item hierarchy of Container, Package and Product).

- 5) Save the new document and switch into edit mode again.
- 6) On the Forwarding Order UI go to tab strip *Output Management*. First of all you can now see an empty Action List. Open the dropdown menu of button *Generate* and choose option *Actions Including Condition Checks*.



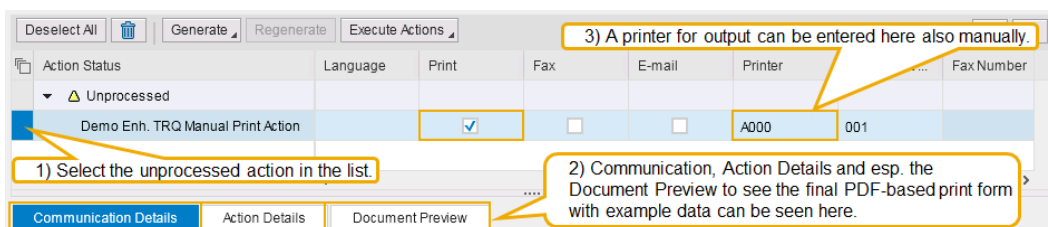
Picture: Generating available actions.

This triggers the Action Determination and generation, taking into consideration the configured conditions for the actions of the related Action Profile. All actions fulfilling the schedule condition are now displayed in the Action List. In the example you should now see only action *Demo Enh. TRQ Manual Print* that we had configured to be active. It is shown as an unprocessed action.



Picture: The generated action.

- 7) Now select the generated action in the list. Below the list you can now see further tab strips with the details of the generated action.

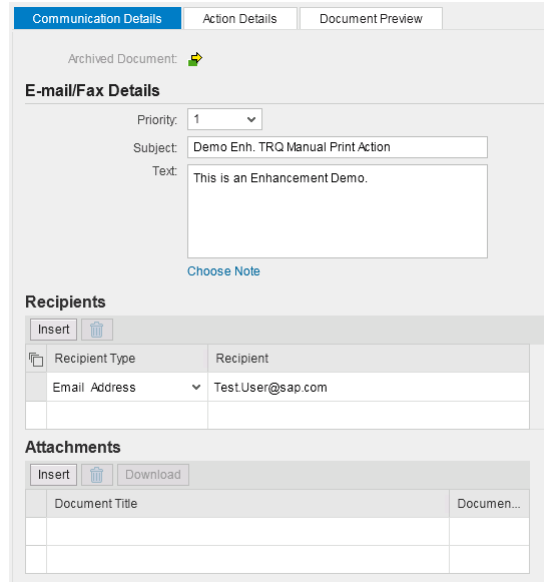


Picture: Selecting an Action, specifying a printer manually and displaying details.

In the selected line with the action you can enter properties for its execution manually, e.g. the output device – in this case a printer – in column *Printer* (when the action is executed, the Printer Determination implemented in the Output Agent Class might

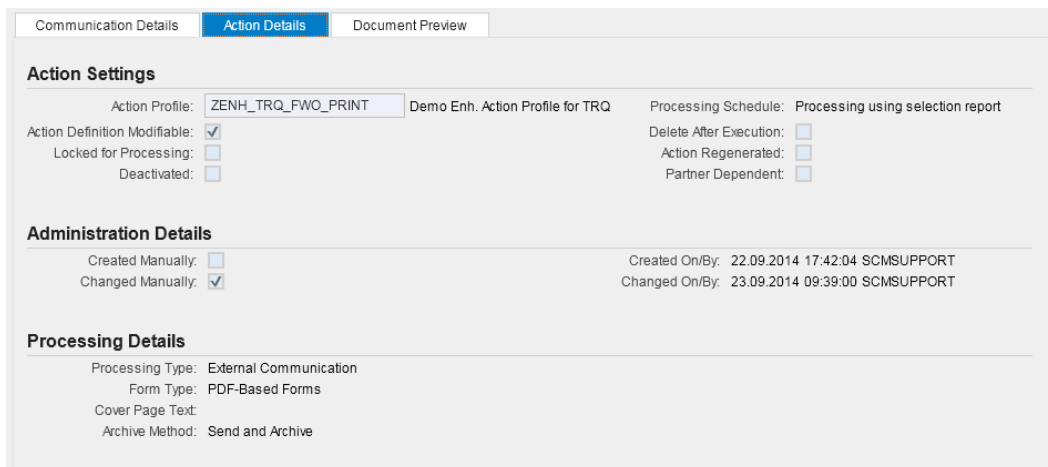
override this input again). In the F4-Help of field *Printer* you can find available printers that can be used.

On tab strip *Communication Details* you can specify e.g. details for email and fax communication. You can specify here e.g. a subject for the email as well as a list of recipients and attachments that shall be sent along with the business document.



Picture: Communication Details.

On tab strip *Action Details* you can see details of the action settings, its administration (e.g. when it was changed the last time and by whom) as well as details about the processing settings. The details shown here represent the settings that were described in sections 7.4.2 and 7.4.3.



Picture: Action Details.

- 8) Click on tab strip Document Preview to finally show the example PDF-based document with the data of the Forwarding Order as a preview.

The screenshot shows the SAP Output Management interface. The 'Document Preview' tab is active, displaying a demo enhancement print form for SAP Transportation Management (PPF Demo). The form contains two main sections: Root Data and Item Data. Root Data includes fields for Document (2100000143), Carrier, Shipper (SP3100), Source (SP3100), Gross Weight (2.275 KG), Gross Volume (30 M3), Created By (SCMSUPPORT), and Changed By (SCMSUPPORT). Item Data includes fields for Item (30), Item Category (PRD), Product (PKE-BOX), Gross Weight (25 KG), and Gross Volume (0.15 M3). The interface also shows a table with columns for Action Status, Processing Type, Document Number, Deactivate, Partner Dependent, and Language.

Picture: The example PDF-based print form in the document preview.

- 9) In the tool bar of tab strip *Outputs* click on selection button *Execute* and then choose option *Execute*. With this, the form will finally be send to the specified printing device.

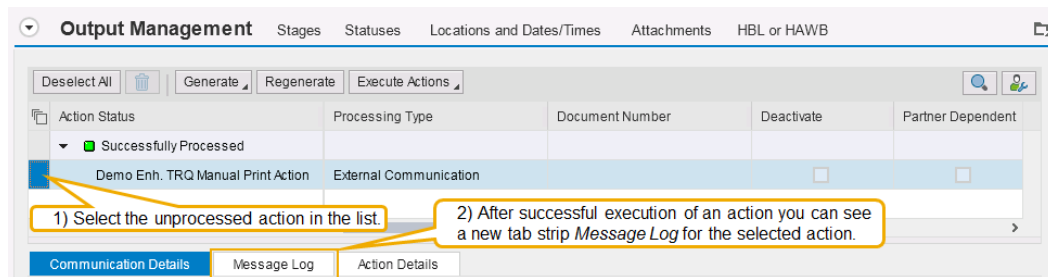
The screenshot shows the SAP Output Management interface with the 'Execute Actions' dropdown menu open. The menu options are 'Include Condition Checks' and 'Skip Condition Checks'. Annotations indicate the steps to execute the action: 1) Select the unprocessed action in the list, and 3) Open the dropdown menu of button Execute Actions to finally execute the selected action, i.e. printing it in this example. The interface also shows a table with columns for Action Status, Processing Type, Document Number, Deactivate, Partner Dependent, and Language.

Picture: Manually executing the action.

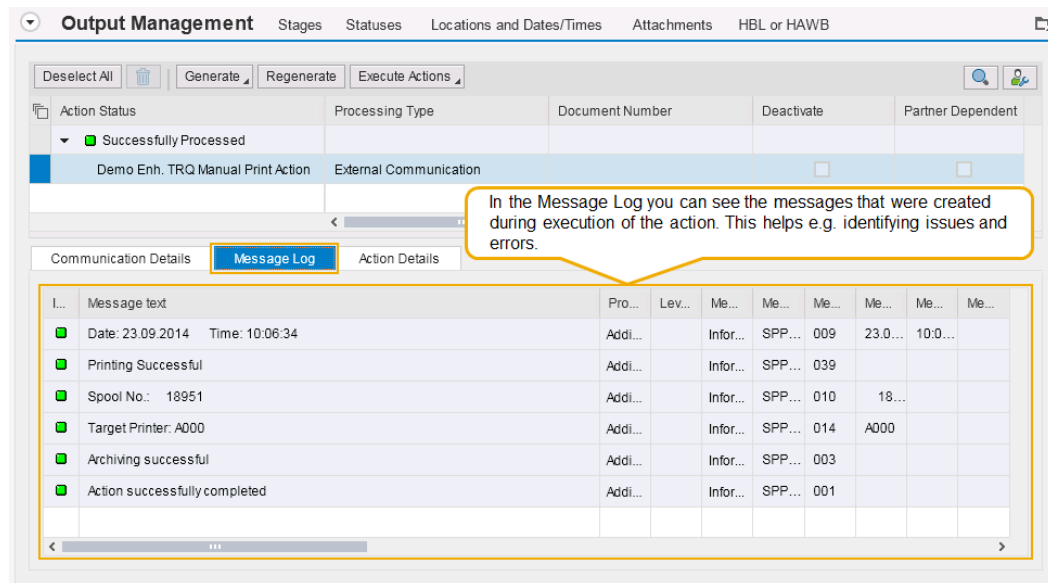
The screenshot shows the SAP Display Forwarding Order 2100000143 message bar. The message bar displays three messages: 'Selected uncritical actions will be processed in the background', 'Uncritical actions found; actions will be processed in background - Display Help', and 'Data saved successfully'. The Output Management interface is visible in the background, showing the 'Execute Actions' dropdown menu.

Picture: Messages in the message bar after successful execution.

- 10) In the list of actions you should now see your action in status *Successfully Processed* (also indicated by a green light). Moreover, there is a new tab strip *Message Log* displayed below the action list. Here you can see the list of messages generated during the execution of the action.



Picture: The successfully executed action in the action list



Picture: The message log for an executed action.

## 8 Enhancing Services

SAP TM uses Enterprise Services for business process integration and the integration with the outside world. This comprises A2A Services for the integration with other components and applications within an enterprise as well as B2B Services for the communication across enterprise boundaries. The standard A2A and B2B Services provided with TM are designed and implemented for the most common standard use cases, i.e. they represent the core content of related business objects.

Enhancements of Enterprise Services are required in case customers or partners have enhanced the standard TM business functionality or applications integrated with TM and want to provide these enhancements in existing services as well.

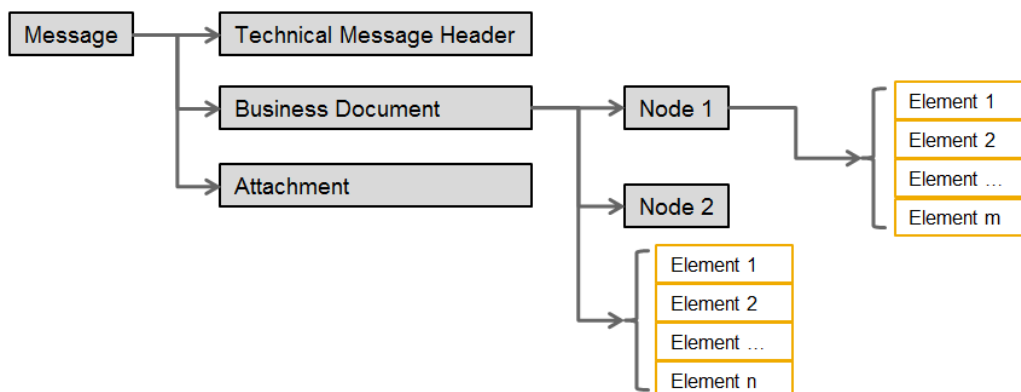
The following sections describe the steps required for creating such service enhancements, focusing on the procedure recommended by SAP for customers and partners, i.e. enhancing the standard service in a new enhancement name space.

In this approach, customers and partners define their enhancement elements in their own name spaces. These enhancements nevertheless refer to the SAP standard service.

- Every SAP Enterprise Service can be enhanced.
- Customers can communicate additional fields to the Business Partner according to their own business logic.
- The Enhancement Concept is modification free.
- The Enhancement structure and the backend implementation do not change by an upgrade, if the Enterprise Service does not change or changes in a compatible way.
- For a new version of the Enterprise Service a new enhancement should be developed.

### 8.1 General remarks on Service Enhancements

The first step is choosing the right part of the structure of the Enterprise Service to be enhanced. A service message typically contains a message header, a business document and (optionally) an attachment. The business document contains one or more nodes that again contain elements, i.e. further nodes and/or attributes (message node hierarchy). It represents the actual business related content of the message.



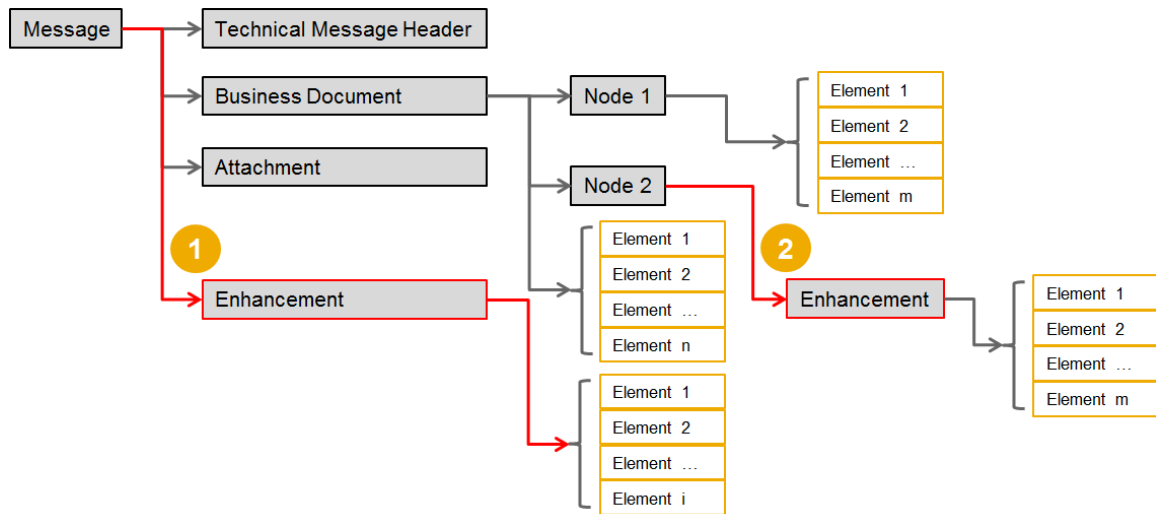
Picture: General structure of a message.

There are two options to enhance a message as shown in the next picture. The first one is to enhance the message data type. You can add an enhancement data type to the message data type that contains all your enhancement elements.

The second one is to enhance the Business Document structure of the message. It is the recommended approach to be used. This allows you adding your enhancement elements to a section of the Business Document data type that it semantically belongs to. For example



adding an additional party to a message should be done on an already existing party node in the business document structure, i.e. you add your enhancement data type to the corresponding node data type. In case your enhancement is semantically not represented in the existing nodes, you can of course create your own enhancement node within the Business Document structure.



Picture: Two options for a service enhancement.

Similar to modeling a BOPF BO, there should be as few nodes as possible and as much as necessary and it should be checked whether the enhancement information can be an attribute of an existing node rather than introducing a new node.

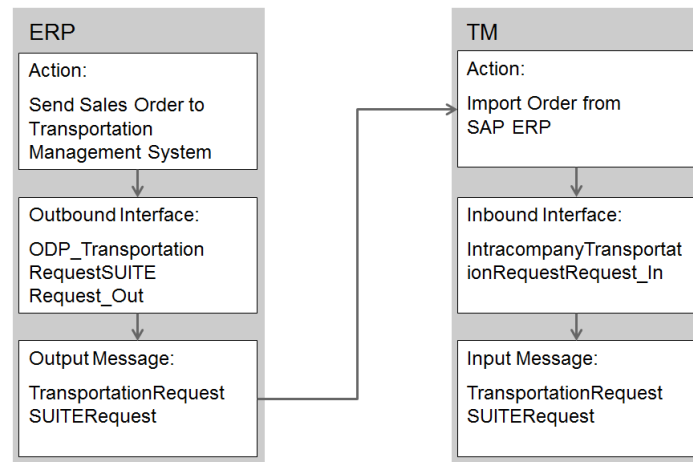
Each message is represented by a message data type that consists of subcomponents (a hierarchy of nodes and attributes) that are again represented by corresponding data types. These data types are modeled in the Enterprise Service Repository (ESR). In general any of these data types can be enhanced as long as they have an explicit ESR data type representation.

It is recommended to only enhance data types that are uniquely used in a single message as otherwise your enhancement will also be included in other messages that reuse the same data type. In case this is not possible the BADIs for all affected messages should be implemented to make sure that all services handle your enhancement in a consistent and common way. In ESR you can create a “where-used” list of the data type to be enhanced to identify all affected places where the data type to be enhanced is reused.

### 8.1.1 Example Service Enhancement

As an example for a service enhancement we take a look at the Sales Order Integration scenario between ERP and TM. In this scenario, a sales order is send from an ERP system to a connected TM system where it will be transferred into a corresponding transportation request. Let’s assume there are already customer/partner specific fields added to the sales order on ERP side that now shall also be taken into consideration in the processing of the related transportation request on TM side.

In the example, we will add a new attribute to the corresponding message that will allow transferring a Route ID from an ERP Sales Order to TM that shall be used for further processing of the resulting transportation request.



Picture: Schema of the example integration scenario.

On the sending side, the service operation (or action) *Send Sales Order to Transportation Management System* of Outbound Interface *ODP\_TransportationRequestSUITERequest\_Out* takes the data of a Sales Order created in ERP and puts this data into the related Output Message of type *TransportationRequestSUITERequest*.

On the receiving side, the service operation (or action) *Import Order from SAP ERP* of Inbound Interface *IntracompanyTransportationRequestRequest\_In* receives the message and takes its data into the related Input Message of type *TransportationRequestSUITERequest*.

Name	Category	Type	Occurrence
p1:TransportationRequestSUITERequest	Element	TransportationRequestSUITERequestMessage	1
MessageHeader	Element	BusinessDocumentMessageHeader	1
TransportationRequest	Element	TranspReqSUITEReqTranspReq	1
actionCode	Attribute	ActionCode	required
changeOrdinalNumberValue	Attribute	OrdinalNumberValue_V1	optional
conciliationPeriodCounterValue	Attribute	CounterValue	optional
LogisticsSchedulingTypeCode	Attribute	LogisticsSchedulingTypeCode	optional
transmissionIndicator	Attribute	Indicator	required
ReceivingCompanyID	Element	NOSC_Tr	0..1
	Element	NOSC_CompanyID	0..1

Picture: The message type representation in ESR.

In the picture above you can see the message type relevant for the example displayed in the Enterprise Service Repository. The example enhancement attribute *Route* will be added as an additional direct sub element of the business document node *TransportationRequest*. The data type that specifies this node is *TranspReqSUITEReqTranspReq* which represents the header level of the business document. Placing the new attribute as a direct sub component of this level would fit from a business perspective, i.e. this fits semantically.

### 8.1.2 Basic steps to enhance an Enterprise Service

In general it takes the following steps in three different areas to create an enhancement for an Enterprise Service. They will be described in more detail in the next sections:

#### Development in System Landscape Directory (SLD):

- Create a Product and Software Component.
- Define Dependencies between an EnSWCV and an underlying SWCV.

#### Development in Enterprise Service Repository (ESR):

- Import an EnSWCV into ESR.
- Create a Namespace in EnSWCV.
- Create an Enhancement Data Type in the SWC.
- Create an Enhancement Data Type for TM in the SWC.
- Create an Enhancement Data Type for ERP (ECC) in the SWC.

#### Development in the Backend Systems:

- Generate the Enhancement Proxy Structure on the ERP (ECC) side and enhance the Outbound Program with the help of a BAdI implementation.
- Generate the Enhancement Proxy Structure on TM side and enhance the Inbound Program with the help of a BAdI implementation.

In the PI System that is connected to your ERP- and TM system to integrate them both, use transaction *SXMB\_IFR* to access the System Landscape Dictionary (SLD) and the Enterprise Service Repository (ESR). For the steps to be done in the backend systems we will use transaction *SPROXY* and the various transactions for implementing BAdIs.

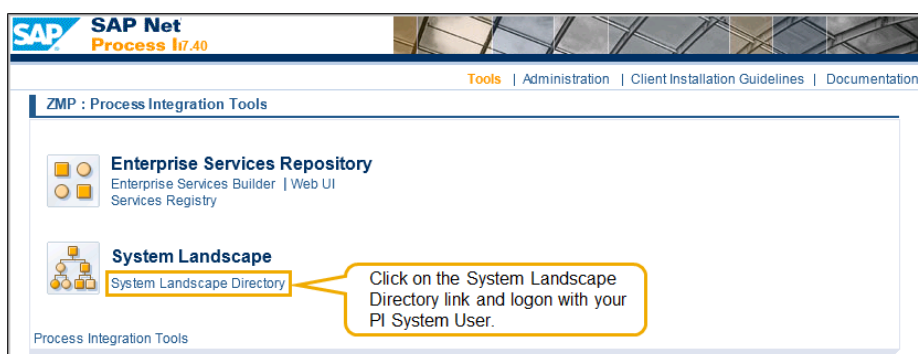
## 8.2 Development in System Landscape Directory (SLD)

A service enhancement done by a customer or partner is always assigned to and defined in a non-SAP Product Version along with a corresponding Enhancement Software Component Version (EnSWCV). This will not only allow keeping the standard service definition separated from any enhancement definition but also separating different enhancements done by e.g. different implementation partners.

### 8.2.1 Creating a Product Version and Software Component

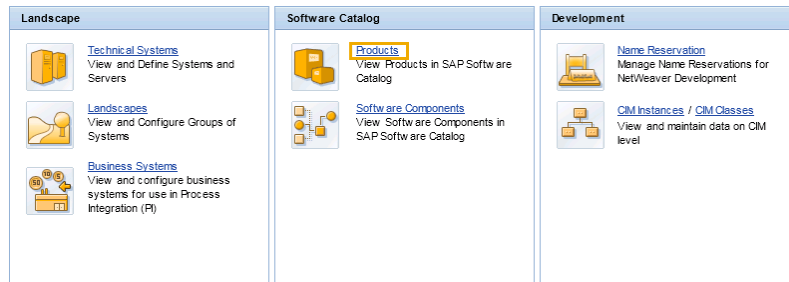
In this first step we create a non-SAP Product Version and an Enhancement Software Component Version that will contain the target service enhancements of the example.

- 1) In your PI System start transaction *SXMB\_IFR*. On the initial screen in the browser click on the System Landscape Directory link and then logon with your PI System User (which should have all required authorizations assigned to execute all following steps).



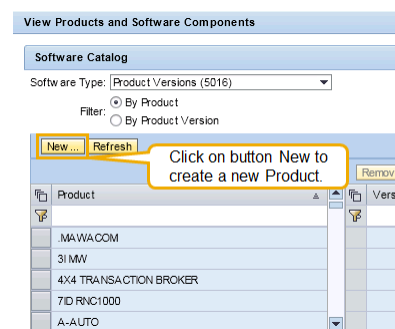
Picture: The start screen of *SXMB\_IFR* in the browser

- 2) On the next screen you can see in section *Software Catalog* a link to the products available in this system. Click on the link *Products*.



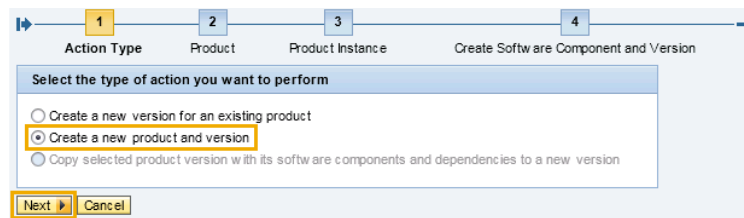
Picture: Software Catalog with link to Products.

Click on button *New* to start creating a new Product / Product Version.



Picture: Create a new product version.

On the next screen you get to the first step of the guided procedure that will help you entering and defining all required information for the new Product Version and the Enhancement Software Component Version (EnSWCV). In step 1 (*Action Type*) choose option *Create a new product and version* and click on button *Next* to get to step 2.



Picture: Create a new Product Version (step 1).

In step 2 (*Product*) enter the following data to define your new product and click on button *Next* to get to step 3.

Field	Value	Comment
Product Name	ZENH_HP_TM_Product	The name of the non-SAP product.
Product Vendor	www.zenhhtpm.com	An identifier of the product vendor (usually a URL/Link like this).
Product Version	9.3	The Product Version (here 9.3 was chosen as also the example SAP TM system was on version 9.3)

Picture: Create a new Product Version (step 2).

In step 3 (*Product Instance*) enter *PIZENHHPTMCOMPONENT* as (example) Instance Name and click on button Next to get to step 4.

Picture: Create a new product version (step 3).

In step 4 (*Create Software Component and Version*) make sure that the following data is entered in the mandatory fields. In this step the required EnSWCV is defined.

Field	Value	Comment
Vendor	www.zenhptm.com	Vendor Name (see step 2)
Name	PIZENHHPTMCOMPONENT	Name for our EnSWCV.
Version	9.3	The version number.
Production State	released	Set to value released to allow immediate import into ESR.

Picture: Create a new product version (step 4).

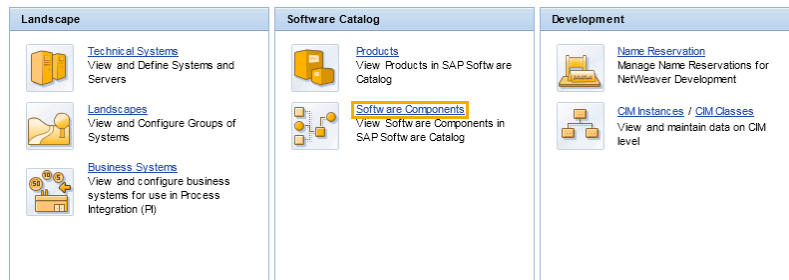
Click on button *Finish* to finally create the (released) Product Version and EnSWCV. The production state was set to value *released*. This allows an immediate import of this new EnSWCV into the ESR after it has been created here in the System Landscape Directory. Finally you should see the following success message.

Picture: Message on successful creation of the Product Version.

## 8.2.2 Defining dependencies between EnSWCV and SWCVs

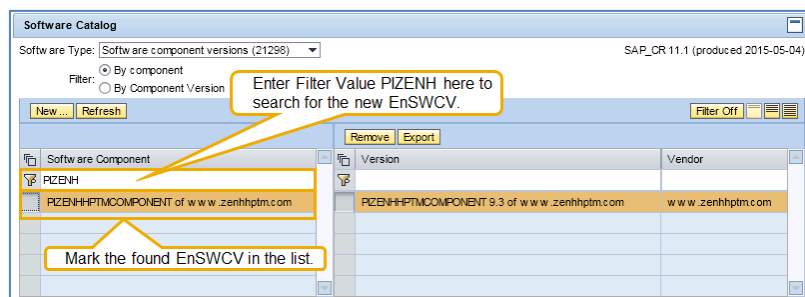
The enhancements that are defined in the new example EnSWCV created in the previous section are always done based on a certain version of the underlying standard application and the Software Component Versions (SWCVs) that it is built on. So for the EnSWCV we also need to define the prerequisite SWCVs that it relies or is based on. Defining these dependencies is done with the following steps.

- 1) On the initial screen of the System Landscape Directory (SLD) you can see in section *Software Catalog* a link to the software components available in this system. Click on the link *Software Components*.



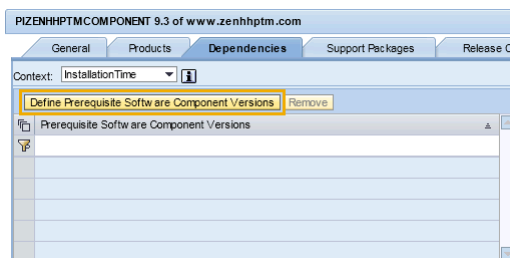
Picture: Software Catalog with link to Software Components.

- 2) On the first screen search for the new EnSWCV *PIZENHHPTMCOMPONENT* by entering a corresponding filter value in the filter field of the Software Component list (see picture below).



Picture: Searching for the new EnSWCV.

In the lower section of the same screen you can then see several tab strips that show the detailed settings and properties of the new EnSWCV. On tab strip *Dependencies* you can now define the prerequisite Software Component Versions that our EnSWCV shall rely or be based on. Click on button *Define Prerequisite Software Component Versions*.



Picture: Define prerequisite Software Component Versions.

On the following screen enter the filter value SAPTM in the filter field of column *Software Component* as shown in the picture below and press *Enter*. Then select the SAP TM Version that your EnSWCV shall be based on. In the example this is SAPTM 1.40 (SAP TM Release 9.3). Select the corresponding entry in the result list and finally click on button *Define as Prerequisite Software Components*.

Software Component	Version	Name
SAPTM	SAPTM 1.01	SAPTM
SAPTM	SAPTM 1.10	SAPTM
SAPTM	SAPTM 1.20	SAPTM
SAPTM	SAPTM 1.30	SAPTM
SAPTM	SAPTM 1.40	SAPTM

Picture: Defining the prerequisite SAP TM Software Component.

With this step, you have defined that the example enhancements are based on Software Component Version SAPTM 1.40 (SAP TM Release 9.3). As we enhance a service to integrate a SAP TM system with an SAP ERP (ECC) system, we also need to define the corresponding ECC Software Component Version which is prerequisite for the example enhancements. This is done with the same steps as for the SAP TM Software Component.

On tab strip *Dependencies* press again on button *Define Prerequisite Software Component Versions*. On the following screen enter the filter value ECC in the filter field of column *Software Component* as shown in the picture below and press *Enter*. Then select the ECC Version that your EnSWCV shall be based on. In the example this is ESA ECC-SE 605. Select the corresponding entry in the result list and finally click on button *Define as Prerequisite Software Components*.

Software Component	Version	Name
ESA ECC-SE	ESA ECC-SE 602	ECC-SE
ESA ECC-SE	ESA ECC-SE 603	ECC-SE
ESA ECC-SE	ESA ECC-SE 604	ECC-SE
ESA ECC-SE	ESA ECC-SE 605	ECC-SE
ESA ECC-SE	ESA ECC-SE 800	ECC-SE

Picture: Defining the prerequisite ECC Software Component.

The final list of prerequisite Software Component Versions should now look as follows:

Prerequisite Software Component Versions
SAPTM 1.40
ESA ECC-SE 605

Picture: Prerequisite Software Component Versions.



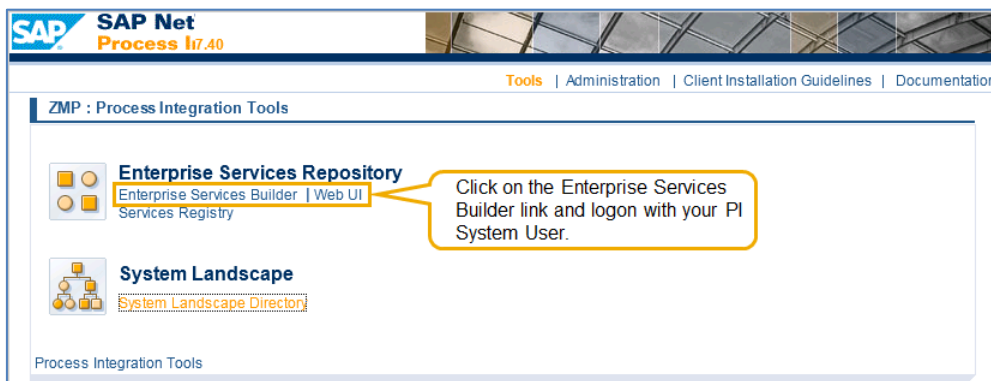
## 8.3 Development in Enterprise Service Repository (ESR)

The additional service message content and the enhancement of the service message type by this new content is done in the ESR. The corresponding objects and definitions are assigned to the Product Version and EnSWCV created in the previous steps.

### 8.3.1 Importing the EnSWCV into ESR

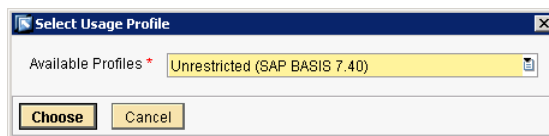
So as a first step, the Enhancement Software Component Version needs to be imported from the System Landscape Directory into the Enterprise Service Repository.

- 1) In your PI System start transaction *SXMB\_IFR*. On the initial screen in the browser click on the Enterprise Services Builder link and then logon with your PI System User (which should have all required authorizations assigned to execute all following steps).



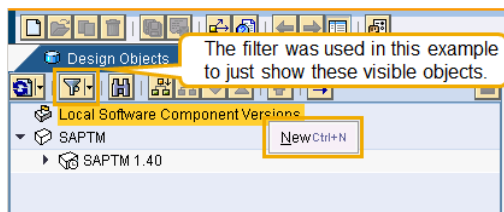
Picture: The start screen of *SXMB\_IFR* in the browser

During the logon process you will be asked to select an available Usage Profile that gives your user the authorizations for accessing all necessary functions and creating all the ESR objects required for this example.



Picture: Selecting a Usage Profile.

- 2) On the left side of the initial screen of the Enterprise Services Builder navigate to tab strip *Design Objects*. There you can find an entry *Local Software Component Versions*. Mark this entry and click the right mouse button to open the related popup menu which contains *New* as the only option. Choose *New*.

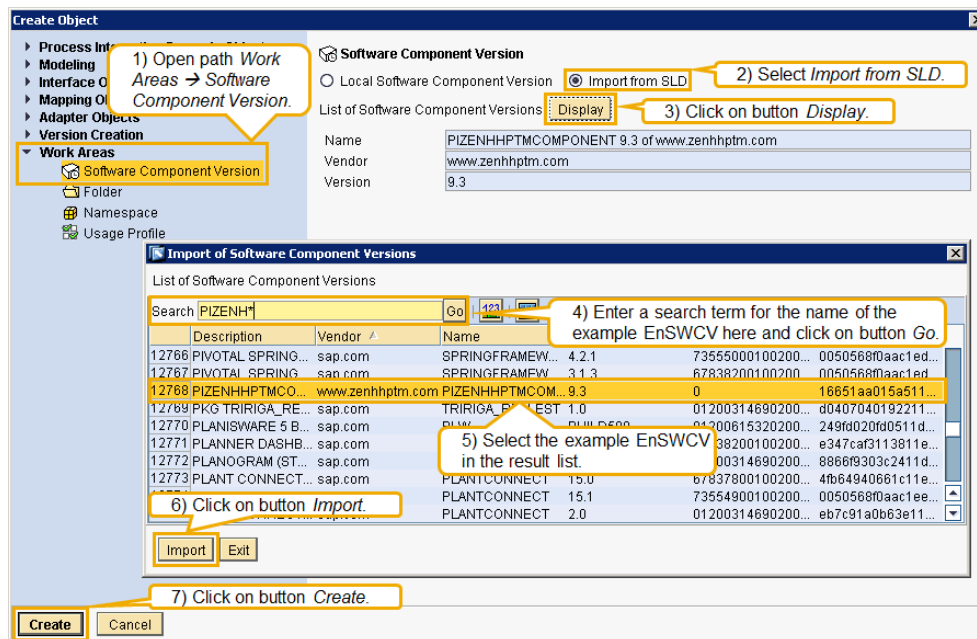


Picture: Creating a new Local Software Component Version.

On the following popup screen follow the path *Work Areas* → *Software Component Version*. On the right side of this screen chose radio button *Import from SLD* (we want to import our EnSWCV from the SLD) and click on button *Display*.

Another popup will come up where you can enter our example EnSWCV as a search term in field *Search* (enter e.g. *PIZENH\**) and then click on button *Go*. Select your EnSWCV *PIZENHCOMPONENT* in the result list and then click on button *Import*. Afterwards you

are back on the initial popup. Finally click on button *Create* to import the selected EnSWCV from the System Landscape Directory into the ESR.

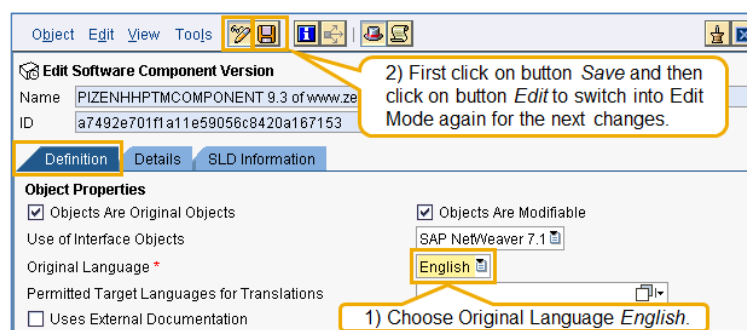


Picture: Importing the EnSWCV from SLD into ESR.

### 8.3.2 Creating a Namespace in the EnSWCV

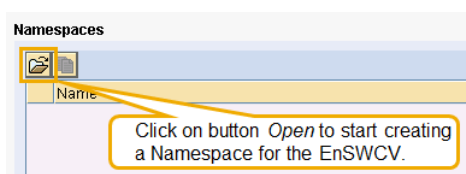
After the EnSWCV has been imported you get back to the Enterprise Services Builder screen. On the right side of this screen you can now see the details of the imported EnSWCV where you need to do the following adjustments.

On tab strip *Definition* select Original Language *English* from the dropdown list and click on the *Save* button. Then click again on the button *Edit* next to the *Save* button to enable the next adjustments for the EnSWCV.



Picture: Defining the Original Language.

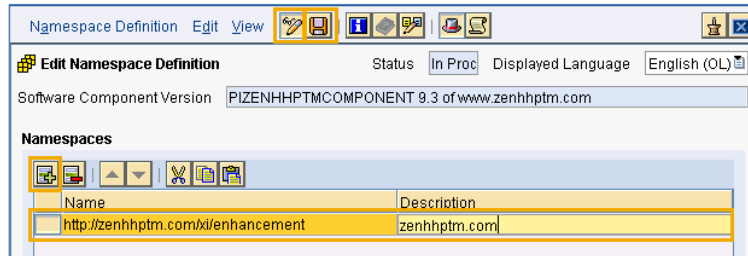
On the bottom of the same tab strip you can find the list of Namespaces assigned to your EnSWCV. Click on button *Open* to create a Namespace for the EnSWCV.



Picture: Create a new Namespace.

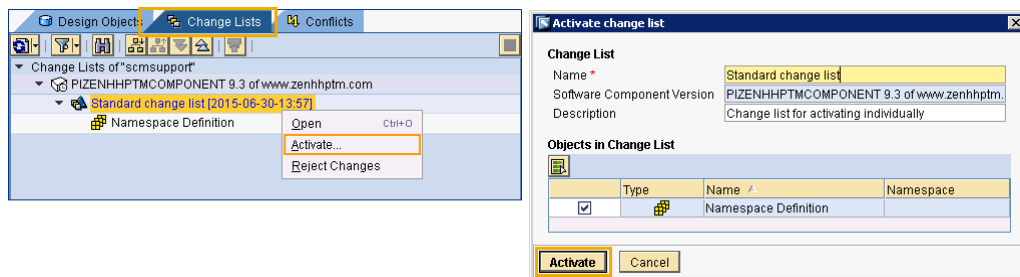
A new tab strip is opened on the right side of the Enterprise Services Builder where you can now define the required Namespace. First click on button *Edit* then on button *Insert Line*. Enter the following data in the new table line to define the Namespace and then click on button *Save*.

Field	Value	Comment
Name	http://zenhhptm.com/xi/enhancement	The Namespace.
Description	zenhhptm.com	The Namespace description.



Picture: Defining the new Namespace.

Finally navigate to tab strip *Change Lists* and activate the created Namespace as follows (this activation step will be repeated also for other objects created in the next steps). In the Change List tree expand the entry with the EnSWCV where you can find a Standard Change List on the next tree level. Right mouse click on this entry opens a popup menu. Select option *Activate*. The following popup contains a list of all objects to be activated. Make sure that all objects are marked for activation and click on button *Activate*.



Picture: Activating a Standard Change List in the Enterprise Services Builder.

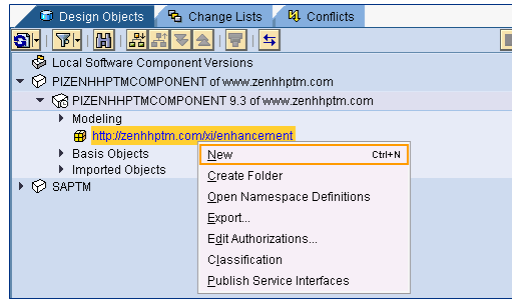
### 8.3.3 Create a Data Type in the EnSWCV.

In the next step, a Data Type for the Route ID is created. This data type is used to represent the additional content of the service message to be enhanced in this example.

Navigate to the tab strip *Design Objects* and in the object tree expand the path down to the Namespace created in the previous step located in the *Modeling* folder of your EnSWCV. The path should look as follows.

*PIZENHHPTMCOMPONENT 9.3 of www.zenhhptm.com* → *PIZENHHPTMCOMPONENT 9.3 of www.zenhhptm.com* → *Modeling* → *http://zenhhptm.com/xi/enhancement*.

Right mouse click on the Namespace opens a popup menu. Choose the option *New*.

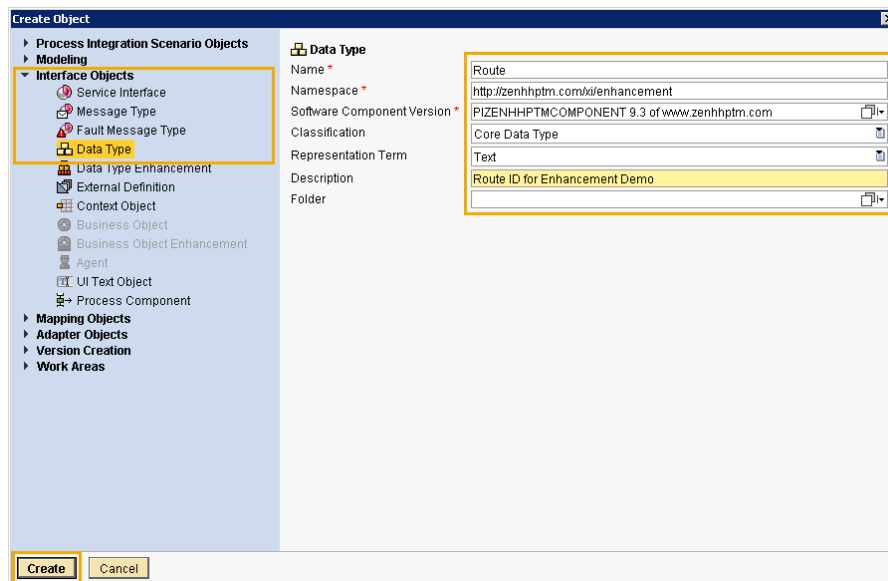


Picture: Creating a new object in the Namespace.

On the following popup follow the path *Interface Objects* → *Data Type*. On the right side enter the following data to specify the Data Type for the example Route ID:

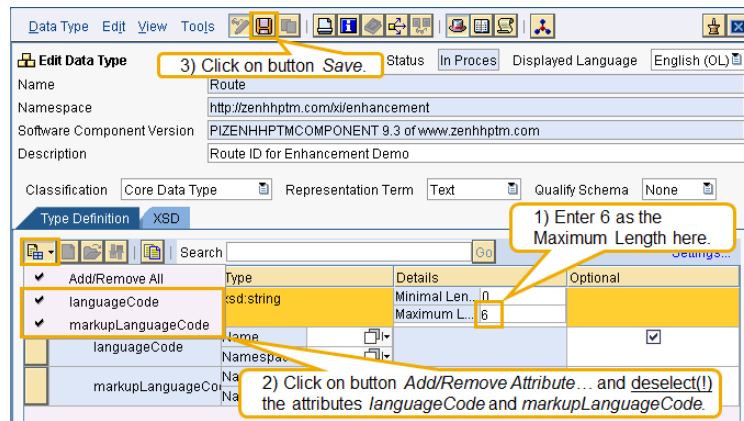
Field	Value	Comment
Name	Route	Name of the data type.
Namespace	http://zenhhptm.com/xi/enhancement	The Namespace that the data type will be located in (predefined).
Software Component Version	PIZENHHPTMCOMPONENT 9.3 of www.zenhhptm.com	The name of the assigned EnSWCV (predefined).
Classification	Core Data Type	Specifies the kind of Data Type to be created.
Description	Route ID for Enhancement Demo	Any text that describes the semantics of the Data Type Enhancement.

Click on button *Create* to create the entered Data Type Enhancement.



Picture: Creating the Data Type for the SAP TM side.

When returning back to the Enterprise Services Builder you can see on the right side that a tab strip was added displaying the details of the new Data Type. Here you should now do the following changes.



Picture: Defining the details of the new Data Type.

Finally, the new Data Type should look as follows:

Name	Type	Details	Optional
Route	XSD Type	xsd:string	
	Data Type	Minimal Len... 0	
		Maximum L... 6	
	Name		
	Namespace		

Picture: The final data type after Save.

Activate the Standard Change List on tab strip *Change Lists* as already described in section 8.3.2 to activate the new data type.

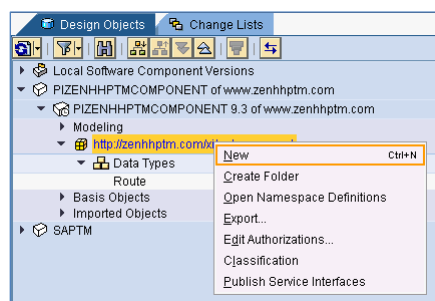
### 8.3.4 Create the Data Type Enhancement for the TM side.

In the next step we create a Data Type Enhancement for the involved message type *TranspReqSUITEReqTranspReq* on the SAP TM side. It will be used to enhance the message type on the receiving side in our example.

Navigate to the tab strip *Design Objects* and in the object tree expand the path down to the Namespace created in the previous steps located in the *Modeling* folder of your EnSWCV. The path should look as follows.

*PIZENHHPTMCOMPONENT 9.3 of www.zenhhptm.com* → *PIZENHHPTMCOMPONENT 9.3 of www.zenhhptm.com* → *Modeling* → *http://zenhhptm.com/xi/enhancement*.

Right mouse click on the Namespace opens a popup menu. Choose the option *New*.

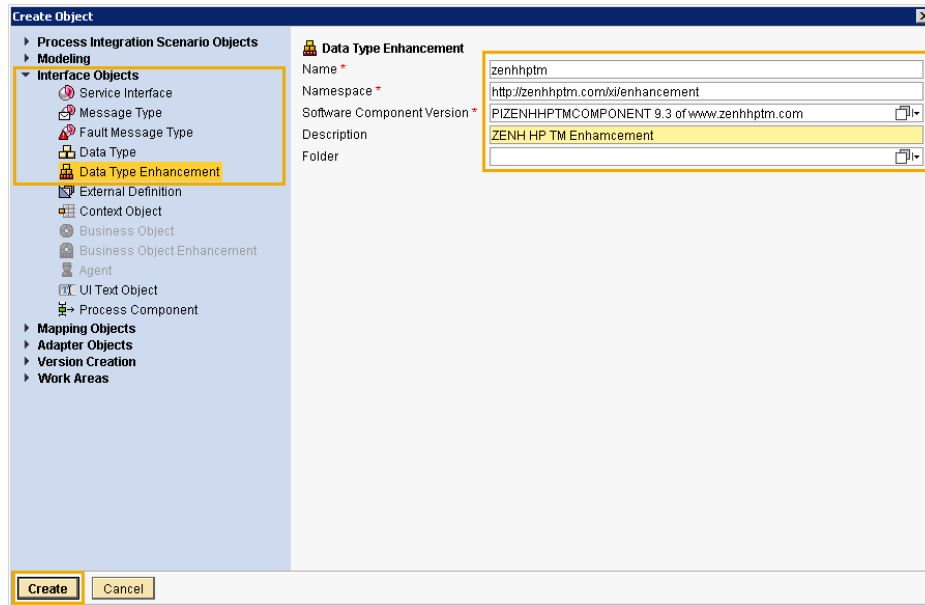


Picture: Creating a new object in the Namespace.

On the following popup follow the path *Interface Objects* → *Data Type Enhancement*. On the right side enter the following data to specify the Data Type Enhancement for the example message type *TranspReqSUITEReqTranspReq* in the corresponding SAP TM namespace.

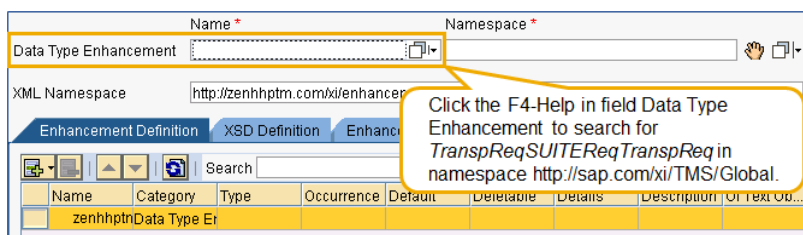
Field	Value	Comment
Name	zenhhptm	Name of the Data Type Enhancement.
Namespace	http://zenhhptm.com/xi/enhancement	The Namespace that the Data Type Enhancement will be located in (predefined).
Software Component Version	PIZENHHPTMCOMPONENT 9.3 of www.zenhhptm.com	The name of the assigned EnSWCV (predefined).
Description	ZENH HP TM Enhancement	Any text that describes the semantics of the Data Type Enhancement.

Click on button *Create* to create the entered data type.



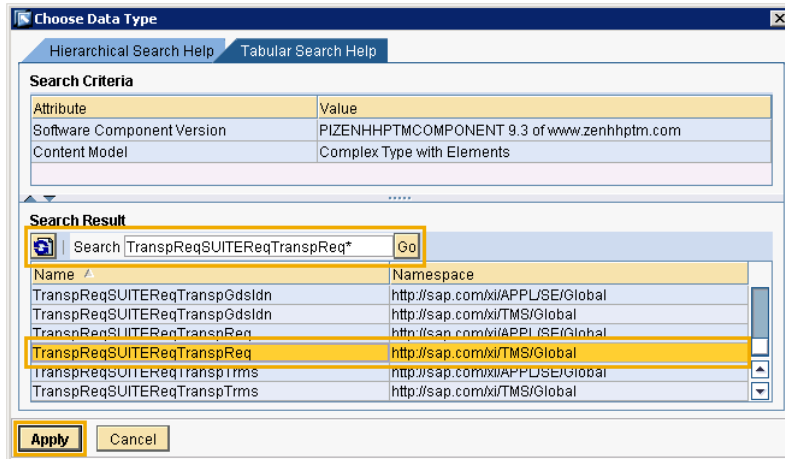
Picture: Creating the Data Type Enhancement for the SAP TM side.

When returning back to the Enterprise Services Builder you can see on the right side that a tab strip was added displaying the details of the new Data Type Enhancement. Here you should now do the following changes. In field Data Type Enhancement use the F4-Help and enter *TranspReqSUITEReqTranspReq* as search term.



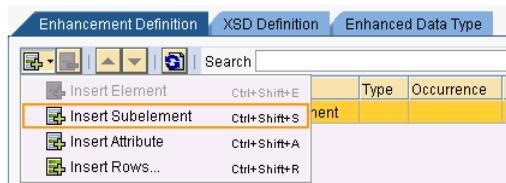
Picture: Adjusting details of the Data Type Enhancement.

In the search result list choose the entry for message type *TranspReqSUITEReqTranspReq* with the assigned namespace *http://sap.com/xi/TMS/Global*. Then click on button *Apply*.



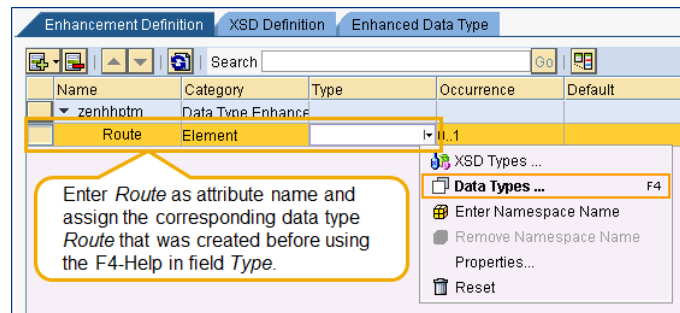
Picture: Searching for the correct Message Data Type.

When returning back to the Enterprise Services Builder screen you should see again the details your Data Type Enhancement. On tab strip *Enhancement Definition* select the listed Data Type Enhancement *ZENHPTM* and click on button *Insert New Lines*. On the popup menu choose option *Insert Subelement*.



Picture: Insert a sub element to the Data Type Enhancement.

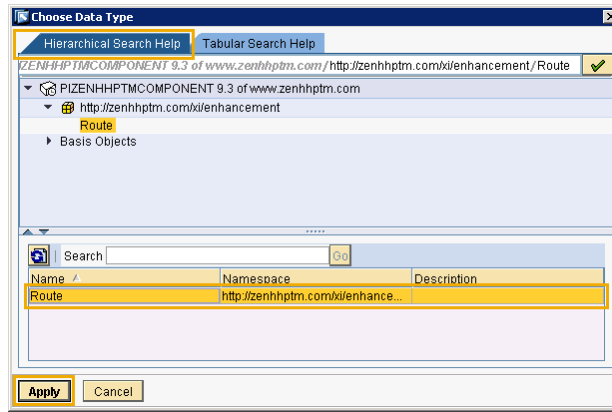
In the new input line enter the attribute name *Route* in column *Name*. Then use the F4-Help in field *Type* to assign the data type *Route* that was created in the previous section.



Picture: Adding an attribute to the Data Type Enhancement.

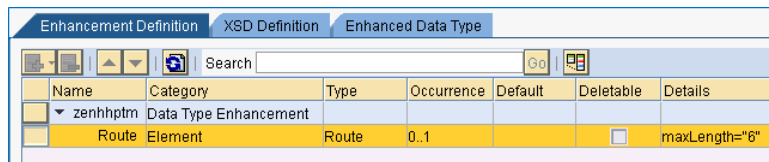
On the F4-Help screen navigate to tab strip *Hierarchical Search Help* and then drill down to the elements of your EnSWCV. There you can find again the data type *Route* created in section 8.3.3. Double click on data type *Route* in the hierarchy to take it over directly or click on it, mark in the search result list and click on button *Apply*.





Picture: Search for the Data Type in the EnSWCV.

Save the Data Type Enhancement and then activate the Standard Change List on tab strip *Change Lists* as already described in section 8.3.2 to activate the new Data Type Enhancement. The Data Type Enhancement for the TM side should finally look as follows.



Picture: The final Data Type Enhancement for the TM side.

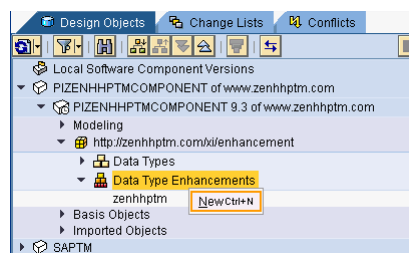
### 8.3.5 Create an Enhancement Data Type for the ECC side.

Also for the ECC side of the example integration scenario we create a Data Type Enhancement for the involved message type *TranspReqSUITEReqTransReq*. It will be used to enhance the message type on the sending side in our example.

Navigate to the tab strip *Design Objects* and in the object tree expand the path down to the Namespace created in the previous steps located in the *Modeling* folder of your EnSWCV. The path should look as follows.

*PIZENHHPTMCOMPONENT 9.3 of www.zenhhtpm.com* → *PIZENHHPTMCOMPONENT 9.3 of www.zenhhtpm.com* → *Modeling* → *http://zenhhtpm.com/xi/enhancement*.

Right mouse click on the Namespace opens a popup menu. Choose the option *New*.

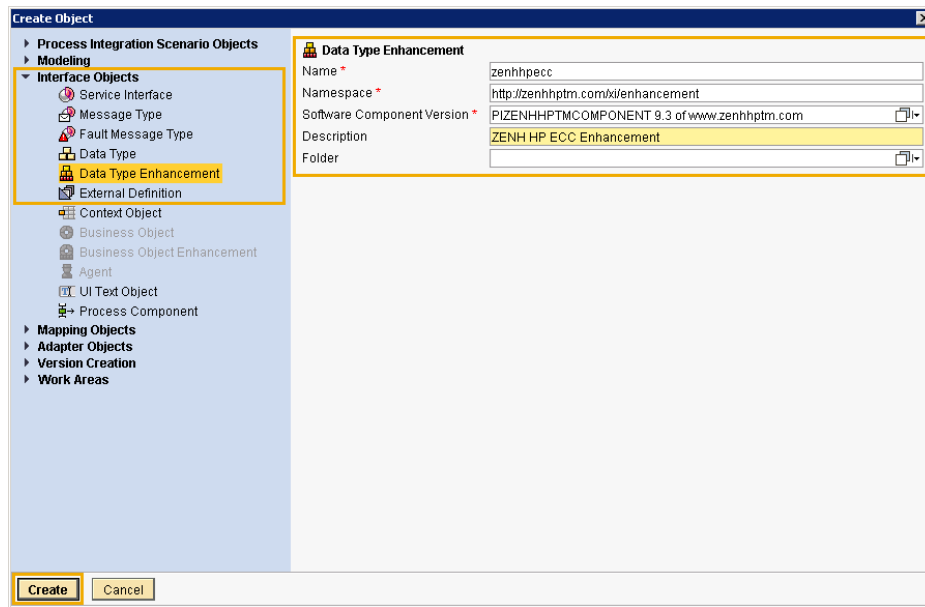


Picture: Creating a new object in the Namespace.

On the following popup follow the path *Interface Objects* → *Data Type Enhancement*. On the right side enter the following data to specify the Data Type Enhancement for the example message type *TranspReqSUITEReqTransReq* in the corresponding ECC namespace.

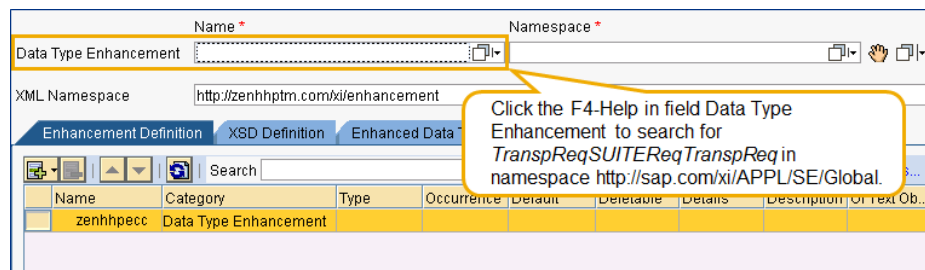
Field	Value	Comment
Name	zenhhpecc	Name of the Data Type Enhancement.
Namespace	http://zenhhptm.com/xi/enhancement	The Namespace that the Data Type Enhancement will be located in (predefined).
Software Component Version	PIZENHHPTMCOMPONENT 9.3 of www.zenhhptm.com	The name of the assigned EnSWCV (predefined).
Description	ZENH HP ECC Enhancement	Any text that describes the semantics of the Data Type Enhancement.

Click on button *Create* to create the entered data type.



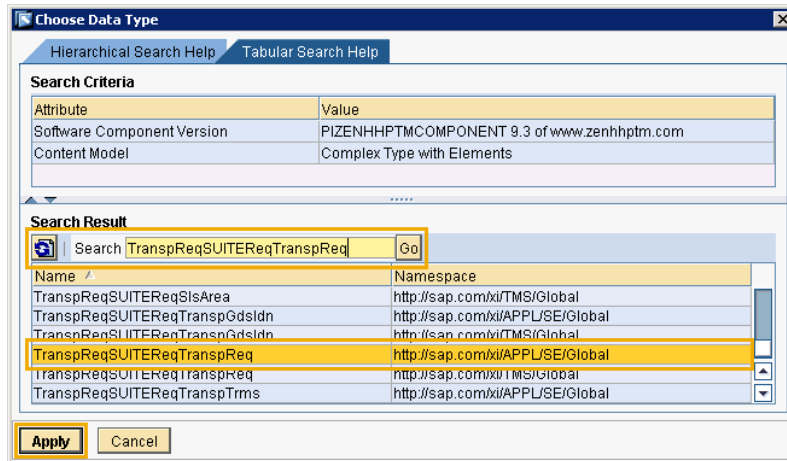
Picture: Creating the Data Type Enhancement for the ECC side.

When returning back to the Enterprise Services Builder you can see on the right side that a tab strip was added displaying the details of the new Data Type Enhancement. Here you should now do the following changes. In field *Data Type Enhancement* use the F4-Help and enter *TranspReqSUITEReqTranspReq* as search term.



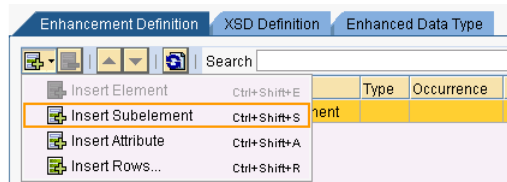
Picture: Adjusting details of the Data Type Enhancement.

In the search result list choose the entry for message type *TranspReqSUITEReqTranspReq* with the assigned namespace *http://sap.com/xi/APPL/SE/Global*. Then click on button *Apply*.



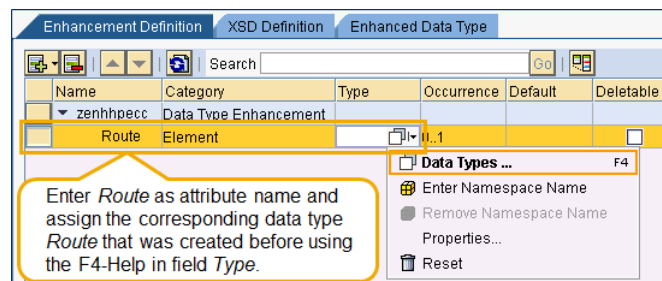
Picture: Searching for the correct Message Data Type.

When returning back to the Enterprise Services Builder screen you should see again the details your Data Type Enhancement. On tab strip *Enhancement Definition* select the listed Data Type Enhancement *ZENHPECC* and click on button *Insert New Lines*. On the popup menu choose option *Insert Subelement*.



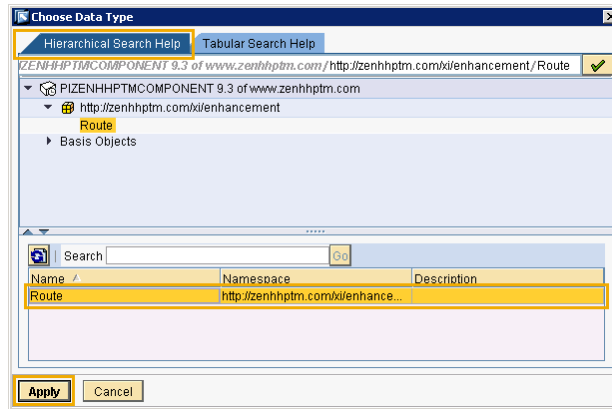
Picture: Insert a sub element to the Data Type Enhancement.

In the new input line enter the attribute name *Route* in column *Name*. Then use the F4-Help in field *Type* to assign the data type *Route* that was created in the previous section.



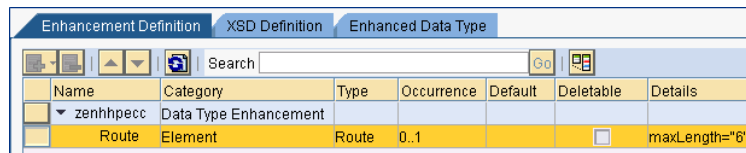
Picture: Adding an attribute to the Data Type Enhancement.

On the F4-Help screen navigate to tab strip *Hierarchical Search Help* and then drill down to the elements of your EnSWCV. There you can find again the data type *Route* created in section 8.3.3. Double click on data type *Route* in the hierarchy to take it over directly or click on it, mark in the search result list and click on button *Apply*.



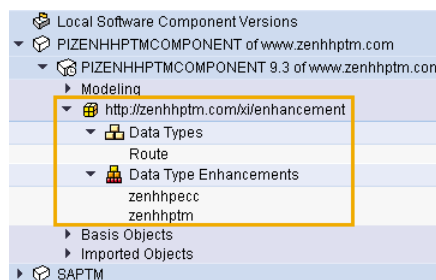
Picture: Search for the Data Type in the EnSWCV.

Save the Data Type Enhancement and then activate the Standard Change List on tab strip *Change Lists* as already described in section 8.3.2 to activate the new Data Type Enhancement. The Data Type Enhancement for the TM side should finally look as follows.



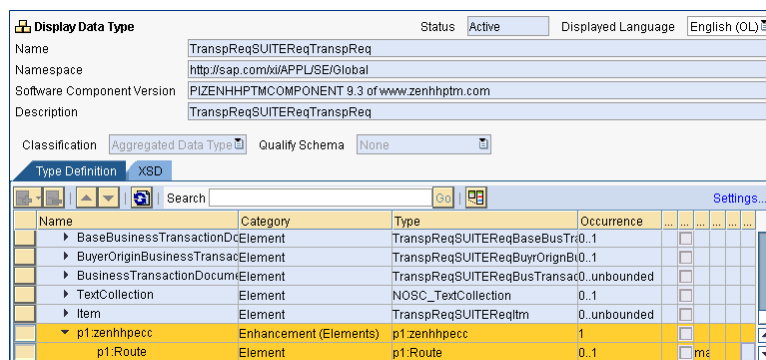
Picture: The final Data Type Enhancement for the TM side.

With this, the development steps in the ESR are complete. On tab strip *Design Objects* of the Enterprise Services Builder, you should now see the following activated objects:



Picture: The created objects in the EnSWCV.

The steps in the last two sections describe how to enhance the message data type of the involved service on the receiver (section 8.3.4) and on the sender side (section 8.3.5). When you take a look at the corresponding message data types in the related namespaces <http://sap.com/xi/TMS/Global> (for TM / receiver) and <http://sap.com/xi/APPL/SE/Global> (for ECC / sender) you can see the created Data Type Enhancements attached to them.



Picture: Example - The message data type enhancement for the ECC side.

## 8.4 Development in the Backend Systems

Before the enhanced service message can be used to exchange also the additional content between the ECC- and the TM-Backend System, a proxy structure for the enhancement must be created in both systems. With this, the Data Dictionary (DDIC) representation of the involved service message, i.e. the Service Message Data Type is enhanced with the additional content and also the WSDL representation of the service message is (automatically) enhanced.

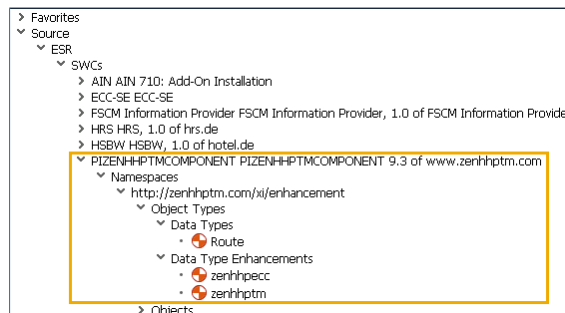
The following sections describe the required steps for generating the corresponding proxies in both backend systems (ERP and TM) and finally how to implement the BADIs of the involved service implementation which will allow filling the new service message content on the sender side (ERP) and extracting the new service message content on the receiving side (TM).

Note: The new ESR content created in the previous sections will only be available and visible in the ECC- and TM-Backend System if the RFC-Connection `SAP_PROXY_ESR` of connection type G (i.e. a HTTP Connection to External Server) is set up correctly in the systems. This connection is required to allow accessing the new ESR content from the ECC- and TM-System which is prerequisite for generating corresponding backend objects.

### 8.4.1 Generating the Enhancement Proxy Structure in ECC

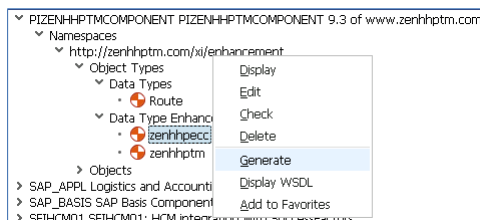
Logon to the ECC System (ERP) and start transaction `SPROXY`. With the above mentioned RFC `SAP_PROXY_ESR` the ESR content created in the previous section is also transferred to the ECC System. In the Enterprise Services Browser navigate to your EnSWCV and the objects that were created there in the ESR.

Besides other navigation options you can navigate along the following path as shown in the picture below: *Source* → *ESR* → *PIZENHHPTMCOMPONENT* (the example EnSWCV) → *Namespaces* → ...



Picture: The example EnSWCV in `SPROXY`.

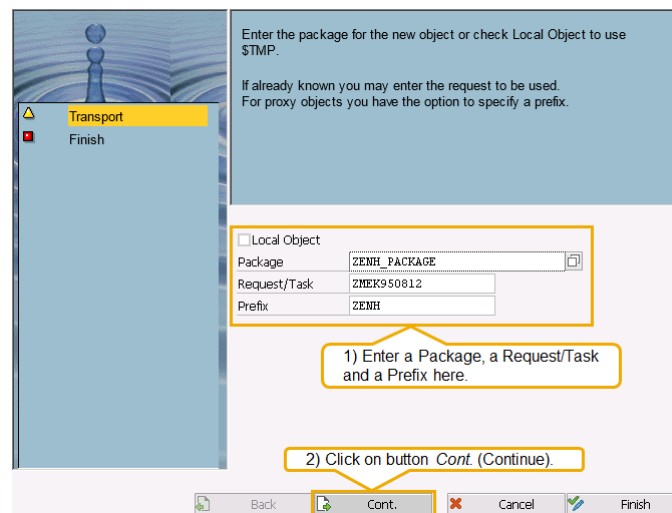
In this section, the enhancement proxy for the ECC side (i.e. the sender) is generated as follows. Navigate to the Data Type Enhancements assigned to the example namespace. Mark Data Type Enhancement `ZENHPECC`, right mouse click to open the popup menu and choose option *Generate*.



Picture: Start generating a Data Type Enhancement Proxy.

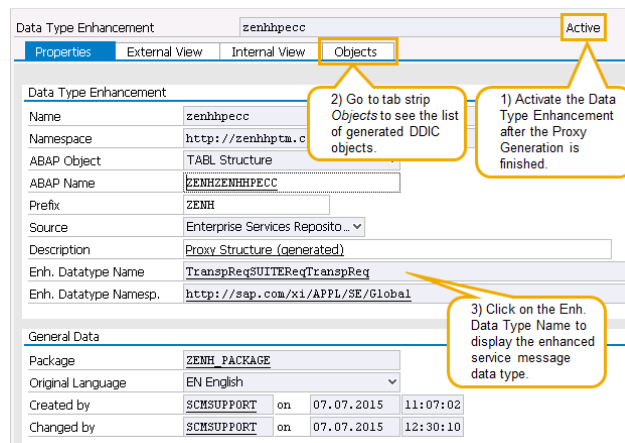
A wizard will come up which helps you specifying all further information required for the generation. On the initial wizard screen, enter the following data:

Field	Value	Comment
Local Object	[false, not set]	Set this flag if you want to make the proxy a local object.
Package	Example: ZENH_PACKAGE	An existing package where that the proxy objects shall be assigned to.
Request/Task	Example: ZMEK123456	The request/task that will allow transporting the generated proxy objects through the system landscape.
Prefix	Example: ZENH	A unique prefix is required to prevent naming conflicts with other, already existing objects → see also F1-Help of this field.



Picture: The wizard for the Proxy Generation (ECC side example).

On the initial wizard screen click on button *Cont.* (Continue) and on the next screen click on button *Complete* to start generating the proxy for Data Type Enhancement *ZENHHPECC*. When the generation is finished, save and activate the displayed Data Type Enhancement.



Picture: The activated Data Type Enhancement after Proxy Generation (ECC).

As shown in the picture above, you can e.g. click on tab strip *Objects* to see the DDIC objects that have been created during Proxy Generation.

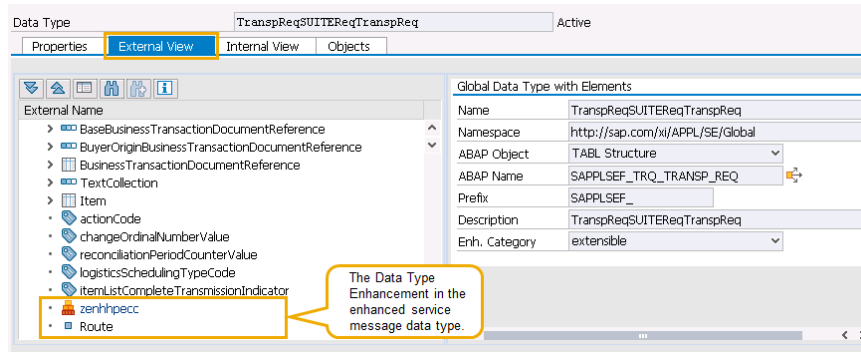
Type	ABAP Name	Prefix	Name	Namespace	Status	Info	Package
	ZENH-ZENHHPECC	ZENH	zenhhpecc	http://zenhhptm.com/xi/enhancement			ZENH_PACKAGE
	ZENH-ROUTE	ZENH	Route	http://zenhhptm.com/xi/enhancement			ZENH_PACKAGE

Picture: The generated DDIC objects.

As you can see there, the Data Type Enhancement *ZENHHPECC* was generated as DDIC structure *ZENHZENHHPECC* (remember that we used the prefix *ZENH*). Moreover, also the Data Type Route was generated as DDIC object (Data Element) *ZENHROUTE*. When you go

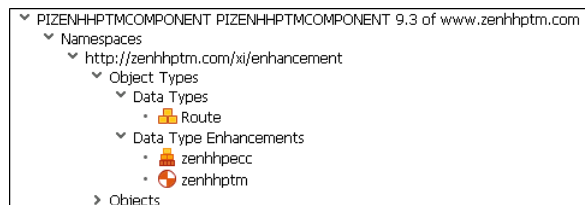
back to the tab strip *Properties*, you can click on the enhanced Service Message Data Type *TranspReqSUITEReqTranspReq* (in field *Enhancement Datatype Name*) to verify that it has been enhanced by the Data Type Enhancement *ZENHHPECC*.

When the Service Message Data Type is displayed, navigate to tab strip *External View* and scroll down to the end of the hierarchy with the data type elements. In this example you should now see the created Data Type Enhancement included in the Service Message Data Type.



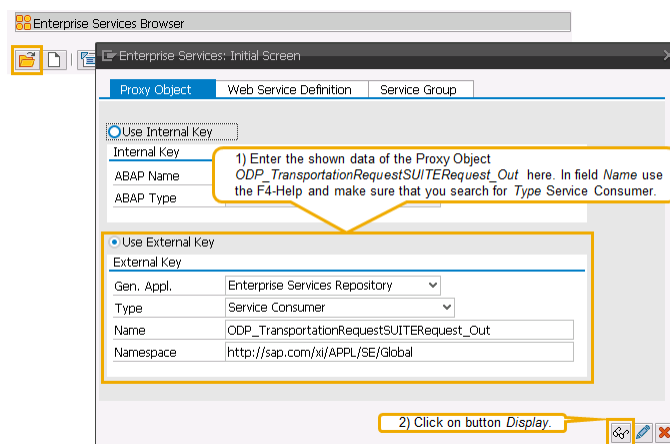
Picture: The enhanced Service Message Data Type (ECC).

In transaction *SPROXY* you can now see that the ECC-related Data Type Enhancement *ZENHHPECC* as well as the data type *Route* have been generated and activated. They can now be used in the context of the BAdI implementation on the (ECC) sender side fill the new service message content for transfer to the related TM-System.



Picture: The generated and active Data Type and Data Type Enhancement (ECC).

You can now also verify via transaction *SPROXY* that the corresponding Outbound Interface *ODP\_TransportationRequestSUITERequest\_Out* contains the enhancement as it uses the Service Message Data Type *TranspReqSUITEReqTranspReq* to define the data that can be send by it.

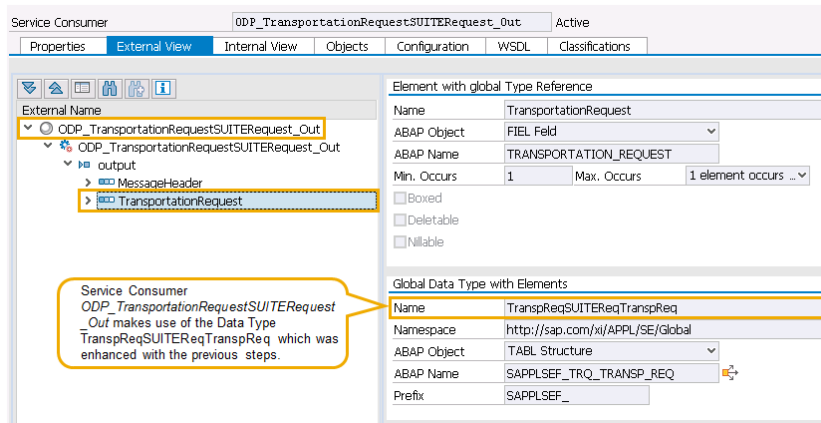


Picture: Searching for the Outbound Interface.



As shown in the picture above, start transaction *SPROXY* and in the Enterprise Services Browser click on button *Open Object*. On the following popup activate radio button *Use External Key*, enter the following data and click on button *Display*.

Field	Value	Comment
Gen. Appl.	Enterprise Services Repository	The Generation Source of the object.
Type	Service Consumer	
Name	ODP_TransportationRequestSUITERequest_Out	Name of the Outbound Interface.
Namespace	http://sap.com/xi/APPL/SE/Global	Automatically filled when you use the F4-Help to search for the name in the field above.

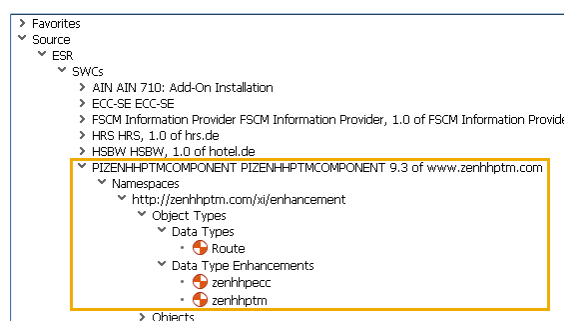


Picture: Outbound Interface using the enhanced Service Message Data Type.

## 8.4.2 Generating the Enhancement Proxy Structure in TM

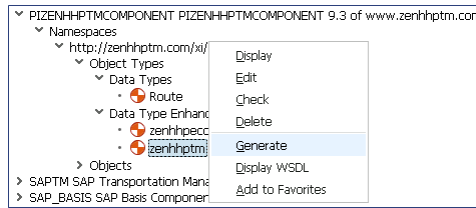
The same steps described in the previous section 8.4.1 on the ECC side now have to be done as well on the TM side to allow receiving and processing the new service message content. Logon to the TM System and start transaction *SPROXY*. In the Enterprise Services Browser navigate to your EnSWCV and the objects that were created there in the ESR.

Besides other navigation options you can navigate along the following path as shown in the picture below: *Source* → *ESR* → *PIZENHHPTMCOMPONENT* (the example EnSWCV) → *Namespaces* → ...



Picture: The example EnSWCV in *SPROXY*.

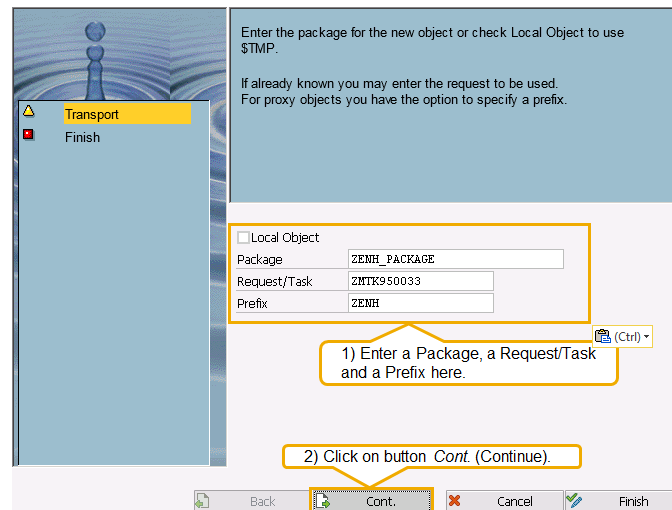
In this section, the enhancement proxy for the TM side (i.e. the receiver) is generated as follows. Navigate to the Data Type Enhancements assigned to the example namespace. Mark Data Type Enhancement *ZENHPTM*, right mouse click to open the popup menu and choose option *Generate*.



Picture: Start generating a Data Type Enhancement Proxy.

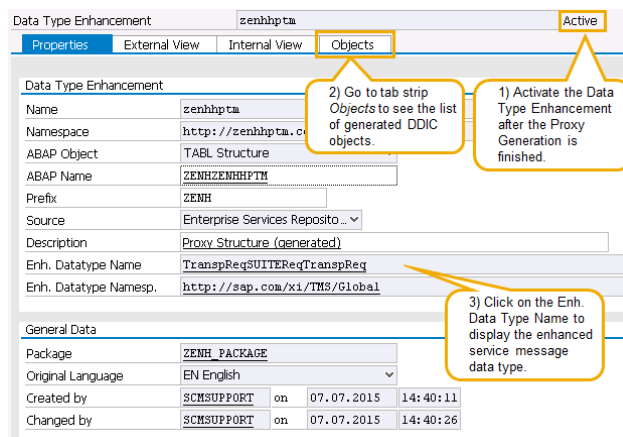
Again the wizard will come up to help you specifying all further information required for the proxy generation. On the initial wizard screen, enter the following data:

Field	Value	Comment
Local Object	[false, not set]	Set this flag if you want to make the proxy a local object.
Package	Example: ZENH_PACKAGE	An existing package where that the proxy objects shall be assigned to.
Request/Task	Example: ZMTK987654	The request/task that will allow transporting the generated proxy objects through the system landscape.
Prefix	Example: ZENH	A unique prefix is required to prevent naming conflicts with other, already existing objects → see also F1-Help of this field.



Picture: The wizard for the Proxy Generation (ECC side example).

On the initial wizard screen click on button *Cont.* (Continue) and on the next screen click on button *Complete* to start generating the proxy for Data Type Enhancement *ZENHHPTM*. When the generation is finished, save and activate the displayed Data Type Enhancement.



Picture: The activated Data Type Enhancement after Proxy Generation (TM).

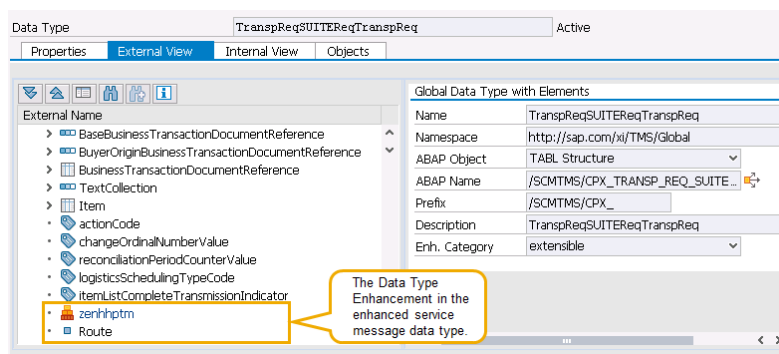
As shown in the picture above, you can e.g. click on tab strip *Objects* to see the DDIC objects that have been created during Proxy Generation.

Type	ABAP Name	Prefix	Name	Namespace	Status	Info	Package
	ZENHZENHHPTM	ZENH	zenhhptm	http://zenhhptm.com/xi/enhancement	✓		ZENH_PACKAGE
	ZENHROUTE	ZENH	Route	http://zenhhptm.com/xi/enhancement	✓		ZENH_PACKAGE

Picture: The generated DDIC objects.

As you can see there, the Data Type Enhancement *ZENHHPTM* was generated as DDIC structure *ZENHZENHHPTM* (remember that we used the prefix *ZENH*). Moreover, also the Data Type *Route* was generated as DDIC object (Data Element) *ZENHROUTE*. When you go back to the tab strip *Properties*, you can click on the enhanced Service Message Data Type *TranspReqSUITEReqTranspReq* (in field *Enhancement Datatype Name*) to verify that it has been enhanced by the Data Type Enhancement *ZENHHPTM*.

When the Service Message Data Type is displayed, navigate to tab strip *External View* and scroll down to the end of the hierarchy with the data type elements. In this example you should now see the created Data Type Enhancement included in the Service Message Data Type.



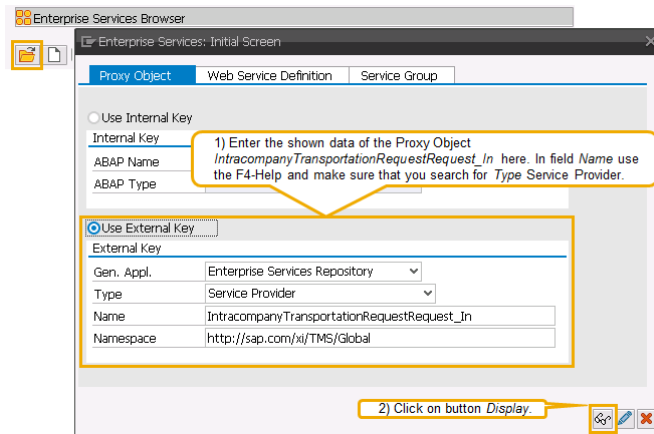
Picture: The enhanced Service Message Data Type (TM).

In transaction *SPROXY* you can now see that the TM-related Data Type Enhancement *ZENHHPTM* as well as the data type *Route* have been generated and activated. They can now be used in the context of the BAdI implementation on the (TM) receiver side to extract the new service message content to be processed by the TM-System.



Picture: The generated and active Data Type and Data Type Enhancement (TM).

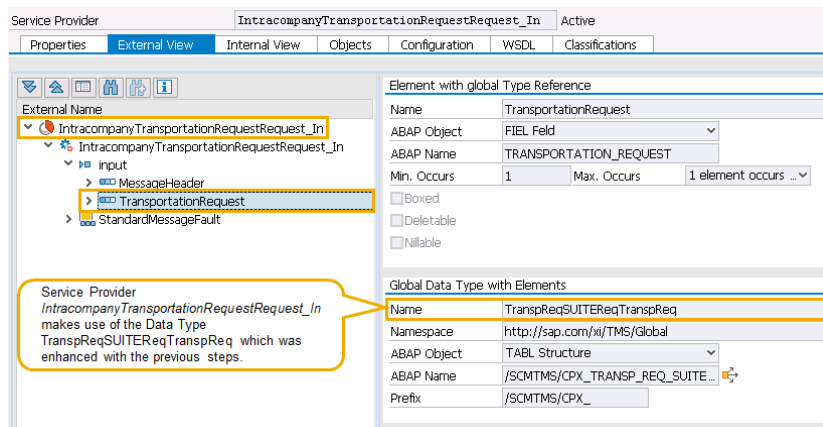
You can now also verify via transaction *SPROXY* that the corresponding Inbound Interface *IntracompanyTransportationRequestRequest\_In* contains the enhancement as it uses the service message data type *TranspReqSUITEReqTranspReq* to define the data that can be received by it.



Picture: Searching for the Inbound Interface.

As shown in the picture above, start transaction *SPROXY* and in the Enterprise Services Browser click on button *Open Object*. On the following popup activate radio button *Use External Key*, enter the following data and click on button *Display*.

Field	Value	Comment
Gen. Appl.	Enterprise Services Repository	The Generation Source of the object.
Type	Service Provider	
Name	<i>IntracompanyTransportationRequestRequest_In</i>	Name of the Outbound Interface.
Namespace	http://sap.com/xi/TMS/Global	Automatically filled when you use the F4-Help to search for the name in the field above.

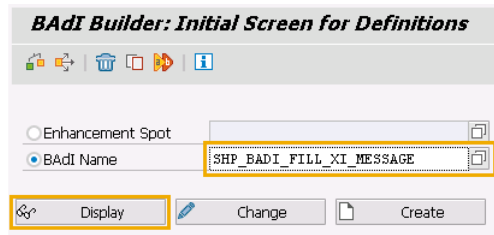


Picture: Inbound Interface using the enhanced Service Message Data Type.

### 8.4.3 BAdI Implementation in ECC

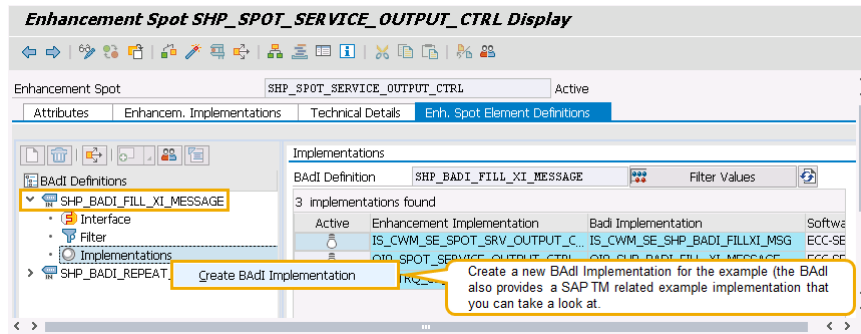
With the steps described so far, the service message has been enabled to transfer additional content. In the example this is the additional information represented with a Route ID. In the next step we need to ensure that the additional service message content is also filled from the related backend data source.

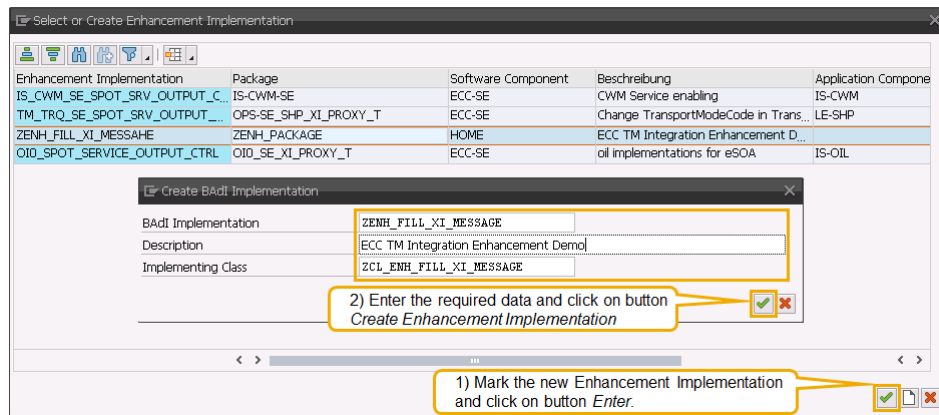
To fill the additional enhancement content of the service message on the ECC side before it is send over to the connected TM-System, you can implement a BAdI. The BAdI relevant in this example is *SHP\_BADI\_FILL\_XI\_MESSAGE*. Start transaction *SE18*, enter this BAdI name in field *BAdI Name* and click on button *Display*.



Picture: Transaction SE18.

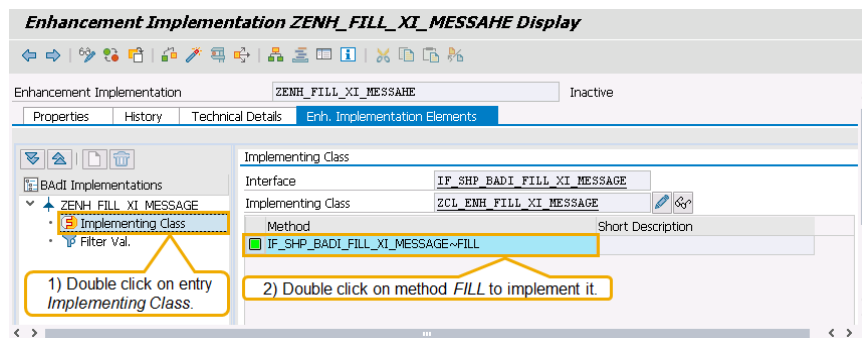
On the next screen navigate to the *Implementations* of the BAdI as shown in the picture below and start creating a new BAdI implementation. Note that the BAdI also provides an example implementation for SAP TM that you can take a look at to see how the BAdI can be used.





Picture: Defining the Implementing Class for the BAdI Implementation

On the next screen you can see your BAdI Implementation *ZENH\_FILL\_XI\_MESSAGE*. Open the hierarchy and double click on section *Implementing Class*. On the right side of the screen you can now see the details of your implementing class, i.e. the Interface, the class and the available methods.



Picture: Starting the implementation of method *FILL*.

You can now start implementing method *IF\_SHP\_BADI\_FILL\_XI\_MESSAGE~FILL* by double clicking on it. The following lines of code show an implementation example for this method:

```
METHOD if_shp_badi_fill_xi_message~fill.

    FIELD-SYMBOLS: <fs_transpreq> TYPE sapplsef_trq_transp_req.

    DATA: lo_trd0 TYPE REF TO cl_shp_xi_message_trd0,
           lv_vbeln TYPE vbeln_vl.

    TRY.
        "Cast XI message provided from outside
        lo_trd0 ?= io_ximsg.

        "Assign Message content to local field symbol to adjust content
        ASSIGN lo_trd0->ms_trd0-transportation_request_suitere-
            transportation_request TO <fs_transpreq>.

        "Determine Sales Order Number from message content
        CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
            EXPORTING
                input  = <fs_transpreq>-id
            IMPORTING
                output = lv_vbeln.

        "Determine Route - the enhanced Message content -
        "based on Sales Order Number. The Route can be read
        "from table LIKP - SD Document: Delivery Header Data
```

```

SELECT SINGLE route INTO <fs_transpreq>-zenhroute
  FROM likp
  WHERE vbeln = lv_vbeln.

CATCH cx_sy_move_cast_error.
  "Error Handling
ENDTRY.

"In addition to the already provided standard content,
"the messagenow contains also the data for the enhanced
"attribute in the message type

ENDMETHOD.

```

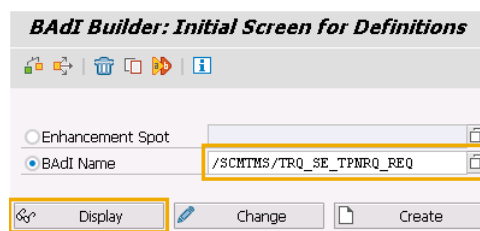
In general, this method implementation takes the Sales Order Number from the already existing content of the service message to be sent and uses it for reading the attribute *Route* from the Data Base Table *LIKP* (the Delivery Header Data). The result of the corresponding *SELECT SINGLE* statement is directly moved into the message enhancement content (attribute *ZENHROUTE*).

#### 8.4.4 BAdI Implementation and BO Enhancement in TM

To extract the additional enhancement content from the service message on the TM side before it is processed in the receiving TM-System, you can implement the corresponding BAdI. The BAdI relevant in this example is */SCMTMS/TRQ\_SE\_TPNRQ\_REQ*. On TM side you can find this BAdI in the IMG (transaction *SPRO*) under the following path:

*SAP Transportation Management → Transportation Management → Business Add-Ins (BAdIs) for Transportation Management → Integration → Enterprise Services → Forwarding Order Management → Forwarding Order → BAdI for TransportationRequestRequest\_In.*

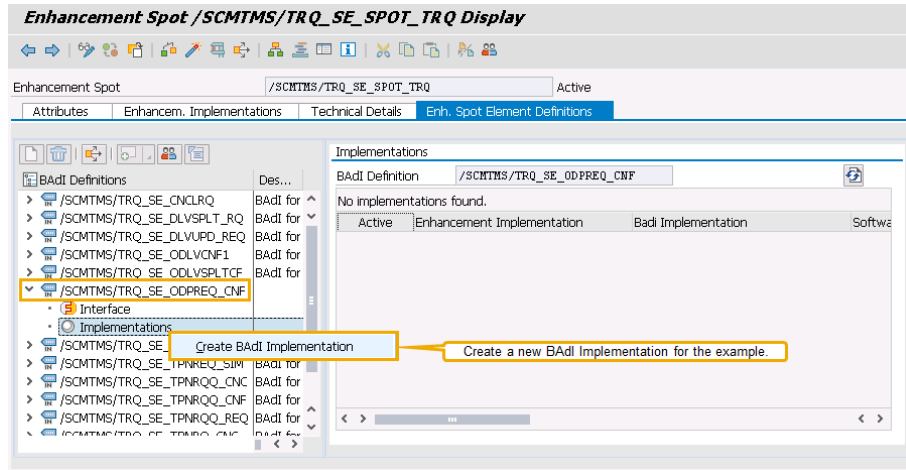
You can start the implementation here or start transaction *SE18*, enter this BAdI name in field *BAdI Name* and click on button *Display*.



Picture: Transaction *SE18*.

On the next screen navigate to the *Implementations* of the BAdI as shown in the picture below and start creating a new BAdI implementation.

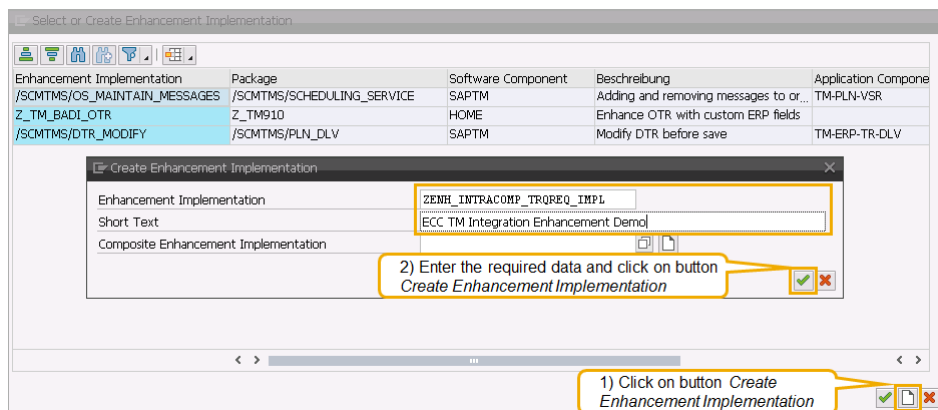




Picture: Creating a BAdI Implementation for the example.

On the first popup click on button *Create Enhancement Implementation*, provide the following data on the second popup and there click on button *Enter*.

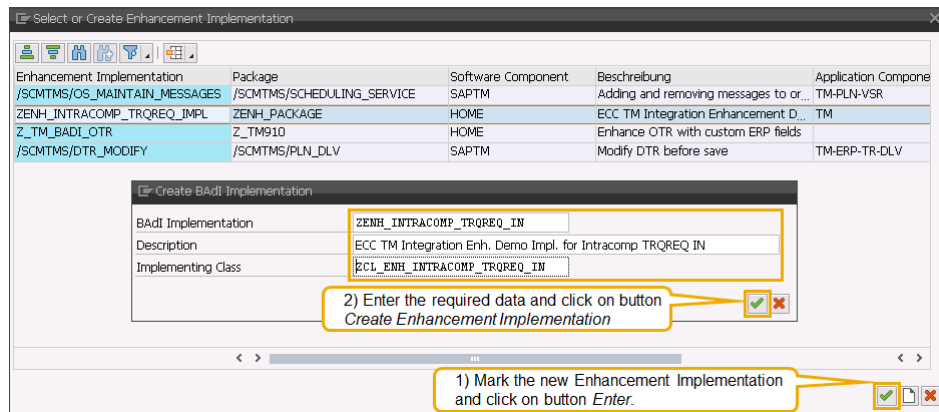
Field	Value	Comment
Enhancement Implementation	ZENH_INTRACOMP_TRQREQ_IMPL	Name of the Enhancement Implementation
Short Text	ECC TM Integration Enhancement Demo	Description



Picture: Creating the Enhancement Implementation.

You get back to the first popup and can see your Enhancement Implementation in the list. Double click on your Enhancement Implementation, on the next popup enter the following data and finally click on button *Create Enhancement Implementation*.

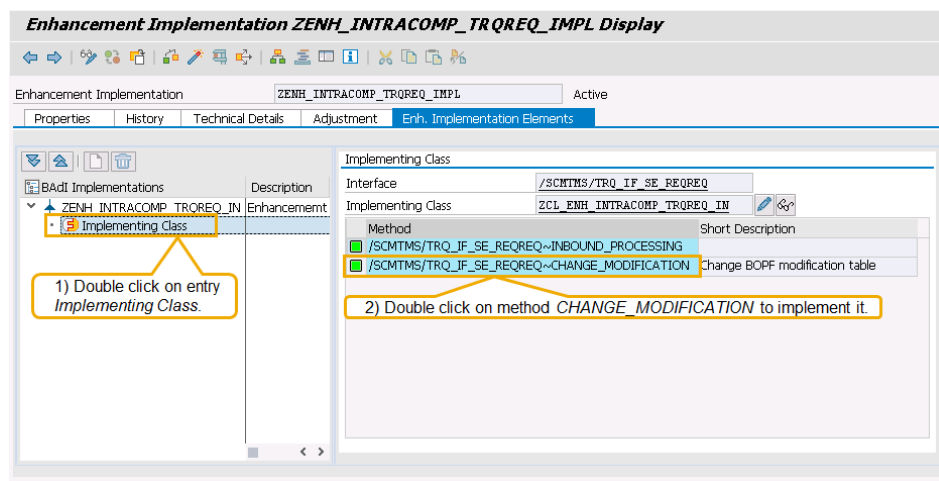
Field	Value	Comment
BAdI Implementation	ZENH_INTRACOMP_TRQREQ_IN	Name of the BAdI Implementation.
Description	ECC TM Integration Enh. Demo Impl. for Intracomp TRQREQ IN	Description
Implementing Class	ZCL_ENH_INTRACOMP_TRQREQ_IN	The name of the Implementing Class.



Picture: Defining the Implementing Class for the BAdI Implementation

Note: In case such a BAdI provides one or more example implementations, another popup comes up which allows create a new, empty BAdI Implementation, copy existing or inherit from existing example implementations.

On the next screen the BAdI Implementation *ZENH\_INTRACOMP\_TRQREQ\_IN* can be seen. Open the hierarchy and double click on section *Implementing Class*. On the right side of the screen you can now see the details of your implementing class, i.e. the Interface, the class and the available methods.



Picture: Starting the implementation of method *CHANGE\_MODIFICATION*.

Start implementing method */SCMTMS/TRQ\_IF\_SE\_REQREQ~CHANGE\_MODIFICATION* by double clicking on it. The following lines of code show an implementation example:

```
METHOD /scmtms/trq_if_se_reqreq~change_modification.

    FIELD-SYMBOLS: <ls_root>          TYPE /scmtms/s_trq_root_k,
                  <ls_modification> TYPE /bobf/s_frw_modification.

    " Process all TRQ Roots - ok, usually it's just one at a time...
    LOOP AT ct_modification ASSIGNING <ls_modification>
        WHERE node = /scmtms/if_trq_c=>sc_node-root.

        "Take the data of the Root Modification and enrich it
        "with the content for the new enhancement field
        "NOTE(!): The BO TRQ Root Node must be enhanced first(!)
        "by the corresponding extension field ZENHROUTE(!)
        ASSIGN <ls_modification>-data->* TO <ls_root>.
        <ls_root>-zenhroute = is_input-transportation_request-zenhroute.
```

ENDLOOP.

ENDMETHOD.

The method receives the message data already via its importing parameter *IS\_INPUT*, including the enhanced content. Remember that on this TM side, the data is extracted from the service message, is transferred into this DDIC representation and then finally used to create or update a corresponding BOPF Business Object Instance (in this example an Order Based Transportation Request which is technically an instance of the Business Object /SCMTMS/TRQ).

To create a new or update an existing BO instance, modifications are created (see e.g. section 3.2.11 in this document). The above mentioned method allows adjusting or adding modifications before they are finally executed and persisted. In this example implementation, the modification for the Root Node of the TRQ (new or to be updated) is identified and the data of the related attributes is enhanced with taking also over the new attribute *ZENHROUTE*.

But note: When you just simply implement the method as shown, you will first of all run into an activation error. The reason is that the TRQ Root Node at this point in time does not have an attribute *ZENHROUTE* that could be filled with the content coming from the service message.

Therefore, you need to add this attribute as an enhancement field to the Root Node of the TRQ BO or you implement the method in a way that it maps the new service message attribute onto an attribute that already exists on the TRQ Root. So you need to ensure that a service message enhancement (in this example a simple additional attribute) can be really processed by the SAP TM Backend and the involved BOPF Business Object (how to create field extensions on a BO node is described in section 3.3.4).

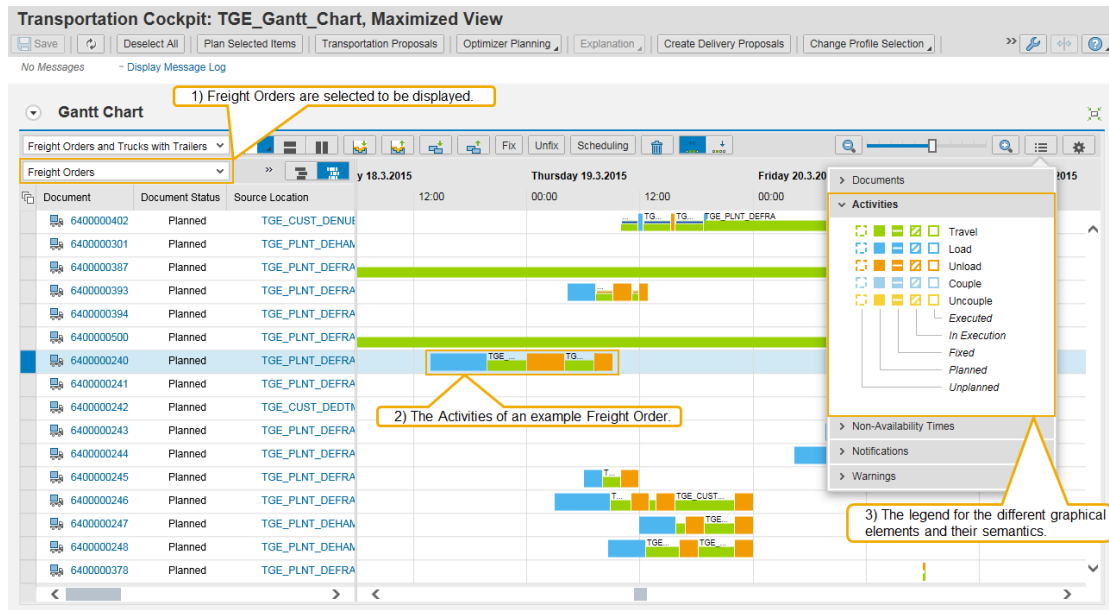
With the described example you can now send a sales order with additional information from ERP to SAP TM. Go to <http://help.sap.com/transportationmanagement> where you can find general and detailed information on how to setup this ERP-TM Integration via Enterprise Services. Navigate e.g. along the following path: SAP Transportation Management → SAP Transportation Management 9.2 → Configuration and Deployment Information. Click on the link to the SAP Service Marketplace provided there. On the following page you can open the list of Integration Guides.

## 9 Enhancing further Objects, Features & Functions

### 9.1 Gantt Chart for Planning Functionality

As per SAP TM 9.2 a Gantt Chart was introduced that allows a visual and interactive transportation planning. In general, the Gantt chart UI is built with HTML5 means.

The Gantt chart is integrated into the standard Web Dynpro (FPM/FBI) based UI via a so called HTML Island that is integrated then into a Web Dynpro Freestyle Component (UIBB). The interfaces *IF\_FPM\_UI\_BUILDING\_BLOCK* and *IF\_FPM\_UIBB\_MODEL* are implemented by this Freestyle Component.



Picture: Example Gantt Chart for a set of planned Freight Orders.

Java Script code is used in a generic reuse part for drawing the Gantt chart. An application (SAP TM, proxy) part contains Java Script code that provides application and customizing data to the Gantt chart and supports the mapping between Gantt Chart and application data.

Note: Currently, with SAP TM 9.2 and SAP TM 9.3 development just finished it is not possible to do enhancements in the mentioned Java Script code parts.

In the picture above you can see the maximized Gantt Chart view for a set of planned Freight Orders. The Freight Orders in this setup are shown with their different activities that are planned for their execution. The Freight Order selected in the picture above shows the following activity sequence: Loading (blue), Travel to the first stop (green), Unloading (at the first stop, orange), Travel to final stop (green) and Unloading (at the final stop, orange).

While enhancing Java Script code of the Gantt Chart is currently not supported, it is nevertheless possible to define e.g. the layout of the Transportation Cockpit and define e.g. a color schema for the Gantt Chart. The following example shows how to make use of the available options for defining such layouts in general, including the Gantt Chart.

- 1) Start SAP TM in the NWBC environment and follow the path *Application Administration* → *Planning* → *General Settings* → *Page Layouts* → *Page Layouts for Transportation Cockpit*.

Click on button *New* to create a Transportation Cockpit Layout. Provide the following details in section *Layout*:

Field	Value	Comment
Page Layout	ZENH_GANTT_CHART	The name of the new Transportation Cockpit Layout.
Validity	User	The new Layout shall be valid for the specified user. Other options are validity for all users or a user role to be specified.
User	[e.g. your User Name]	Enter e.g. your User Name here if you want the new Layout to be only valid for yourself.
Activate Command Line	[space]	If set, a command line is displayed that allows you entering planning instructions, similar to a Shell program in an OS where you can type in commands to get things done.
Transportation Proposal Layout	Standard Layout	The Layout for the Transportation Proposal (TP) Screen; if you create TPs along with using your new Layout, the specified Layout will be used for the TP Screen.

Picture: Creating a new Transportation Cockpit Layout.

- 2) Open section *Visibility Pushbutton* to define which actions shall be visible on the toolbar of the Transportation Cockpit. Activate the actions that are required in the context of the new Transportation Cockpit Layout.

Picture: Defining the visibility of actions on the Transp. Cockpit toolbar.

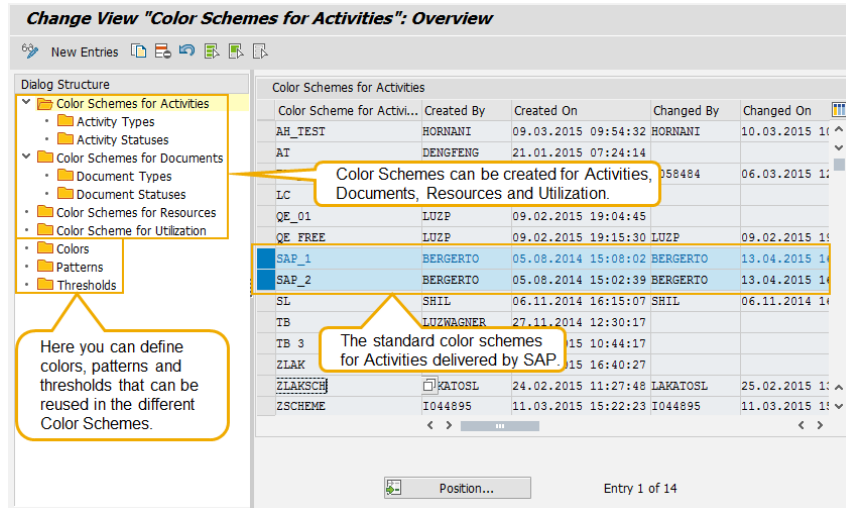
- 3) Below the section *Visibility Pushbutton* you can find the details for all areas that you can include into a Transportation Cockpit Layout. For each area you can define its position and width on the Layout as well as its visibility. For the example, use the following details for each area:

Area / Field	Field/Value	Value	Comment
<b>Gantt Chart</b>			
Position of Screen Area	Bottom Left		The Gantt Chart shall be displayed in the bottom left section of the Transportation Cockpit.
Width	50%		50% of the Screen Width shall be used.
<b>Content of Gantt Chart</b>			
Gantt Chart	Display	Yes	The Gantt Chart content shall be displayed in the Layout
	Layout	ZENH_TRK_AND_TRL	The Gantt Chart Layout used here is actually customizing that will be created in the next step of this example.
<b>Requirements</b>			
Position of Screen Area	Top Left		
Width	50%		
<b>Content of Requirements Area</b>			
Freight Unit Stages	Display	Yes	
	View	Standard View	
Freight Unit Hierarchy	Display	No	
	View	Standard View	
<b>Requirement Groups</b>			
Position of Screen Area	Not Visible		
<b>Transportation Unit</b>			
Position of Screen Area	Not Visible		
<b>Orders</b>			
Position of Screen Area	Top Right		
Width	50%		
<b>Content of Orders Area</b>			
Freight Orders/Freight Bookings	Display	Yes	
	View	Standard View	
[all other content]	Display	No	
	View	Standard View	
<b>Order Details</b>			
Position of Screen Area	Bottom Right		
Width	50%		
<b>Content for Order Details Area</b>			
Overview, Stages, Carrier Ranking, Allocation, Charges	Display	Yes	
	View	Standard View	
[all other content]	Display	No	
	View	Standard View	
<b>Resources</b>			
Position of Screen Area	Not Visible		
<b>Hierarchies</b>			
Position of Screen Area	Not Visible		
<b>Map</b>			
Position of Screen Area	Not Visible		

- 4) As for the Gantt Chart area settings mentioned in the table above, the layout for the content of the Gantt Chart used in this Transportation Cockpit Layout is defined in customizing. The Gantt Chart customizing can be found in the IMG under the following path: *SAP Transportation Management* → *Transportation Management* → *Basic Functions* → *Gantt Chart*.

- 5) Navigate along the following IMG path: *SAP Transportation Management → Transportation Management → Basic Functions → Gantt Chart → Define Color Schemes and Patterns for Gant Chart.*

Here you can define color schemes and patterns for displaying activities, resources. It allows also defining threshold values e.g. for resources when the color for the corresponding capacity shall change. Own colors can be defined too by providing RGB color codes. Moreover, patterns can be defined that represent e.g. the status of an activity.



Picture: Color Schemes and Patterns for Gantt Chart.

Some standard colors, patterns and color schemes for activities, documents, resources and utilization are delivered with the standard SAP TM. Of course you can create and define your very own color schemes with all the available details.

- 6) Navigate along the following IMG path: *SAP Transportation Management → Transportation Management → Basic Functions → Gantt Chart → Define Field Lists and Label Schemes for Gantt Chart.*

Here you can define field lists that represent the data of documents and resources displayed as columns in the selection panel of the Gantt Chart. Before you can define such a field list, the fields and their source (i.e. where the data actually comes from) needs to be defined here too.

Depending on the object context, an available field can be provided with data from different field sources. The following table shows the available objects and the DDIC structures that define the content for the corresponding fields:

Object	DDIC Structure	Comment
Activity	/SCMTMS/S_GNT_ACTIVITY	DDIC structure with fields for Activities.
Freight Unit	/SCMTMS/S_GNT_ORDER	DDIC structure with fields for TOR Root.
Handling Resource	/SCMTMS/S_GNT_RESOURCE	DDIC structure with fields for Resources.
Road Freight Order	/SCMTMS/S_GNT_ORDER	DDIC structure with fields for TOR Root.
Trailer	/SCMTMS/S_GNT_RESOURCE	DDIC structure with fields for Resources.
Trailer Unit	/SCMTMS/S_GNT_RESOURCE	DDIC structure with fields for Resources.
Truck	/SCMTMS/S_GNT_RESOURCE	DDIC structure with fields for Resources.

At runtime, these structures are then filled via Move-Corresponding. Only the fields defined in the field lists used for a Layout are read (to keep performance high). If you add e.g. an Extension Field to the Root Node of BO /SCMTMS/TOR, it will automatically be available in the DDIC structure /SCMTMS/S\_GNT\_ORDER and filled at runtime via Move-Corresponding. In case you want to add a field which is not already read in standard, besides adding such a field to the DDIC structure, you would have to place





## 9.2 Transportation Charge Management Enhancements

### 9.2.1 Adding a new scale base

(Short summary – detailed description with examples will follow soon).

- 1) Extend the Scale Item structure with the required extension fields in extension include /SCMTMS/INCL\_EEW\_TC\_SCALE\_ITEM. Do the enhancements as described earlier within a new append structure and add the required extension fields there.
- 2) Add a new scale base in customizing.
  - a. In the IMG (SPRO) follow the path Transportation management -> Basic Functions -> Charge Calculation -> Data Source Binding for Charge Calculation -> Define Scale Bases.
  - b. Add a new entry and use the new field for the field assignment.
  - c. Set the other properties accordingly.
- 3) Enhance the Scale UI: Open Web Dynpro Component configuration /SCMTMS/WDCC\_TCM\_SCALE\_ITM (package /SCMTMS/UI) Use the Web Dynpro enhancement concept to add a new column - the new field from the scale item structure will be available here The column visibility is controlled by TCM view exit class /SCMTMS/CL\_UI\_VIEWEXIT\_TCM, method SCALE\_ITEM\_PROP. No changes should be necessary.

### 9.2.2 Adding a new calculation base

(Short summary – detailed description with examples will follow soon).

- 1) Extend communication structure: Use the Extension Include for the calculation base structure /SCMTMS/INCL\_EEW\_TCC\_CB Append a new field with the corresponding data element. The component name must be the same as on the underlying document. The field will be copied over to the internal communication structure and available in the engine automatically
- 2) Add calculation base in customizing In IMG go to Transportation Management - Basic Functions - Charge Calculation - Data Source Binding for Charge Calculation - Define Calculation Bases Add new entry and use the new field for the field assignment. Set scale base and other properties
- 3) Optionally implement BAdI or helper class  
If the calculation base needs additional logic, you can use the BAdI method GET\_CALC\_BASE\_VALUES of BAdI /SCMTMS/TCC\_BO\_DATA\_ACCESS . Please refer to the BAdI documentation for details. Set BAdI flag in calculation base customizing. You can also use the helper class approach, which allows you to extend existing helper classes. The interface /SCMTMS/IF\_TCC\_CALC\_BASE is very similar to the BAdI interface. You can refer to existing helper classes /SCMTMS/CL\_TCC\_CB\_\* for sample implementations. The BAdI or helper class is called after the value has been retrieved from the communication structure if a field assignment is provided as well.

### 9.2.3 Adding a new resolution base

(Short summary – detailed description with examples will follow soon).

- 1) Adapt resolution base configuration. Add new resolution base to table /SCMTMS/C\_RES\_BS. This used to be customizing but is now delivered as control table. Only the resolution base name has to be maintained, all other fields are currently not used.
- 2) Implement logic in BAdI: The Data Access Object provides a BAdI /SCMTMS/TCC\_BO\_DATA\_ACCESS for customer implementations which will be called

in case no standard implementation for the resolution base is found. The method GET\_RES\_BASE\_KEYS needs to be implemented. Please refer to the BAdI documentation or the standard implementation for details.

(This is a draft Version! To be completed in the next version).

### 9.3 Master Data Objects

(This is a draft Version! To be completed in the next version).