DEVELOPER'S GUIDE TO TRANSIENT UI - BOPF INTEGRATION (TBI)

A STEP-BY-STEP CASE STUDY
BY DRAGOŞ M. FLORESCU

I.CONTENT

II.INTRODUCTION

Transient UI – BOPF Integration (TBI) is a TM-specific concept for implementation of more complex UI Building Blocks, which cannot be covered by the generic Floorplan Manager BOPF Integration (FBI).

Let's have a look at the following mock-up:

Hierarchy	Document Type	Source Location	Source Location Address	Departure Date	Departure Time	Departure Timezone	Dest. Location	Dest. Location Address	Arrival Date	Arrival Time	Arrival Timezone
Forwarding Order 100111	FWO	W-001	Warehouse Walldorf	01.07.2015	10:00:00	CET	C-666	Jane Doe / Boston	15.07.2015	16:00:00	EST
Freight Unit 501234	0001										
Truck FO 201234	1001	W-001	Warehouse Walldorf	02.07..2015	08:00:00	CET	DEHAM	Port Hamburg	02.07.2015	17:50:00	CET
Ocean Booking 15	BSEA	DEHAM	Port Hamburg	03.07.2015	15:00:00	CET	USNEK	Port Newark	10.07.2015	16:00:00	EST
US-Truck FO 301234	1002	USNEK	Port Newark	11.07.2015	08:00:00	EST	C-666	Jane Doe / Boston	11:07.2015	12:30:00	EST
Freight Unit 505678	0001										
Truck FO 205678	1001	W-001	Warehouse Walldorf	02.07..2015	16:00:00	CET	DEHAM	Port Hamburg	03.07.2015	08:00:00	CET
Ocean Booking 15	BSEA	DEHAM	Port Hamburg	03.07.2015	15:00:00	CET	USNEK	Port Newark	10.07.2015	16:00:00	EST
US-Truck FO 305678	1002	USNEK	Port Newark	11.07.2015	07:00:00	EST	C-666	Jane Doe / Boston	11:07.2015	11:30:00	EST

The UIBB displays data from various sources:

- the "forwarding order" hierarchical level -> from TRQ Root
- the freight unit level -> from TOR (FU) Root
- the freight document level -> from TOR (FO/BO) Root and TOR Stop

This is something that FBI concept cannot handle with its standard logic. The mechanism intended to handle such cases (the so-called "UI Helper classes") has some drawbacks, especially in the area of performance, mainly because there are no "good practices" available, but also due to the fact that some standard FBI processing still takes place, sometimes totally unnecessarily, or even confusingly.

Historically, the business object TOR used a transient node (OVERVIEW) to display such complex data. The advantage seemed obvious: since now everything was in one node, the standard FBI mechanism could be used. Unfortunately, this node increased with time, as it became used in more scenarios, and the performance was affected.

The purpose of TBI is to offer the developers an alternative to these concepts (FBI helper classes and BOPF transient nodes), while providing a more guided approach than previously.

In this document, I'll describe the basic steps required for implementation of the above mock-up.

III.BASIC CONCEPTS

Before going into the complete specification of the repository objects involved, let's define some important concepts that need to be understood before any implementation starts.

A. DATA INTERFACE

As by-product of TBI development, we introduced the concept of **abstract data interface** when retrieving BOPF data.

There are in TM quite a few "central" methods that are supposed to retrieve and modify, in an encapsulated manner, data from various BO nodes. Initially, these methods were written using the official BOPF interface – service manager. Then, these methods were called in a "BO-internal" context, like from a determination or action, where the usage of the service manager is an overhead (one should use the provided IO_READ and IO_MODIFY interfaces).

Things became more complicated when such methods were used in UI context, where FBI manages own buffers, which the UI-specific functionalities should actually use. Moreover, when writing UI data back to BO, the change notifications must be captured and sent to the UI controller.

In the best cases, some methods were rewritten to take into account all three cases, something like:

```
IF io_controller IS BOUND.
  "read data using provided FBI controller
  ...
ELSE IF io_read IS BOUND.
  "read data using provided IO_READ.
  ...
ELSE.
  "instantiate and use a service manager
  ...
ENDIF.
```

This leads to cumbersome maintenance of similar coding (not entirely equal, since each approach features different method signatures).

More often than not, though, the original methods were used, leading to performance problems, due to usage of service manager in inappropriate places.

To alleviate this, we introduced an abstract data interface, featuring the most common methods of the service manager. There are three implementations: one using service manager, one using a given IO_READ and one using the FBI controller singleton. There is a factory method that instantiate by default a service manager-based implementation, unless optional IO_READ / IO_MODIFY are provided.

Thus, such central "helper" methods can be written context-agnostic. If their caller provides an explicit data interface object, it is used, otherwise the factory methods is called to provide a service manager-based one. But from this point on, the central "helper" is freed from the burden of keeping track of the context.

Example from class **/SCMTMS/CL_TOR_HELPER_ITEM**

```
METHOD get_dray_for_item.
  IF io_dataintf IS BOUND.
    DATA(lo_dataintf) = io_dataintf.
  ELSE.
```

```
    lo_dataintf = /scmtms/cl_data_intf_factory=>get_instance( io_read ).
  ENDIF.

  lt_item_key = it_item_key.
  lo_dataintf->retrieve_by_association(
    EXPORTING
      iv_bo_key      = /scmtms/if_tor_c=>sc_bo_key
      iv_node_key    = /scmtms/if_tor_c=>sc_node-item_tr
      iv_association = /scmtms/if_tor_c=>sc_association-item_tr-to_root
      it_key         = lt_item_key
      iv_fill_data   = abap_true
    IMPORTING
      et_target_key  = lt_root_key
      et_key_link    = lt_bookitem_bookroot
      et_data        = lt_book_root ).
```

This method is also called from UI, case in which the caller instantiates a FBI-controller-based data interface object, which then redirects the RBA call above to the methods GET_TARGET_KEYS and GET_NODE_DATA of the FBI controller, which benefit from the FBI buffers.

B. TBI SUPERCLASS

Technically, a TBI implementation is an ABAP class that implements two ABAP interfaces. Each implementation is free to choose own ways or writing the required logic, as long as data is provided via the declared interface methods and public interface attributes.

However, to simplify the implementation, we created an abstract Superclass, which offers some basic generic processing for some (but not all) steps.

We'll see in the next chapter how the superclass can help.

C. SEPARATION OF UI DATA FROM INTERNAL BUFFERING

FBI forces developers to design a UI structure containing not only the data displayed in UI, but also technical data required for further processing. Examples: foreign instance keys that were required for further navigation or various non-UI attributes (categorizations, classifications) that were needed in the view exit classes.

The bigger-than-necessary UI structures posed performance problems in the UI framework (Web Dynpro).

TBI allows (actually: encourages) the developers to design a minimal UI structure, while the rather technical attributes are kept internally in a member attribute of the TBI implementation class. We'll see detail on this in the next chapter.

D. OBJECT KEY

Similar to FBI, an UIBB is identified by an **object key (a combination of a BO and node)**:

```
interface /SCMTMS/IF_UI_TBI_CORE public .
  types:
    BEGIN OF ts_object,     "BO identification of accepted input keys
      bo   TYPE /bobf/obm_bo_key,
      node TYPE /bobf/obm_node_key,
    END OF ts_object .
```

There is a slight difference, however. In FBI, the object key applies to all records in the UIBB (if the base FBI view declares *"I am a TRQ Item",* then all records in the data table are TRQ items), whereas in TBI is merely describes the accepted input keys (the keys that are passed through connector from a source UIBB).

This information is used when adding a wire connector from the parent UIBB to the TBI-based UIBB.

E. ELEMENT CATEGORY

By looking at the mock-up, we can see that the records in the hierarchical lists need to be handled differently, based on the data source. For that purpose, TBI uses a so-called **Element Category**, which is used:

- to control where the data is to be read and where the modification are to be posted
- to control the generic functionality that determines field properties
- to control the generic functionality that determines action properties
- to execute generic BOPF actions

In the above example, we can see that we can define three element categories: one for the forwarding order hierarchy levels, one for freight unit levels and one for freight document levels.

Definition:

```
interface /SCMTMS/IF_UI_TBI_CORE public .
  types TY_ELEMENT_CAT type CHAR2 .
```

F. FIELD PROPERTY DEFINITION

FBI generates, for *each* attribute in the UI structure, for reference fields for properties (mandatory, read-only, enabled and visible). It does not even matter whether the attributes are marked as technical or not. As a result, the generated structure that is passed to FPM/WD is very large.

TBI allows the developer to specify, for each field in the UI structure, whether such reference fields are desired or not. In the mock-up, we can clearly see that document type, location descriptions and time-zones are displayed as non-editable text views, making by definition the "read-only" and "mandatory" properties obsolete. Since we always need to see this data, the "visible" property is not required either. Thus, we can completely skip the usage of "dynamic" (= instance-specific) properties for these columns, as the "static" (instance-independent) properties are sufficient.

The TBI-specific FPM feeders use this definition to decide whether its generic property handling should use the reference fields from the generated UI structure, or the FPM field usages.

Definition:

```
interface /SCMTMS/IF_UI_TBI_CORE public .
  types:
    BEGIN OF ts_fieldprop,  "Cardinality of attribute properties
      name    TYPE name_komp,
      dynamic TYPE boole_d,
    END OF ts_fieldprop .
  types:
    tt_fieldprop TYPE HASHED TABLE OF ts_fieldprop WITH UNIQUE KEY name .
```

Based on this definition, the TBI feeders generate less cluttered dynamic structures, improving the performance in UI framework. For the fields defined as static, the TBI core functionality assigns the

G. TBI ASSOCIATIONS

TBI-based UIBBs are usually complex lists or hierarchies, which require "detail" areas (mainly forms) displayed underneath, using the Master-Detail pattern.

However, as we've seen above, the "object identification" of such UIBBs refers merely to the semantic of the incoming keys, but tells nothing of individual records from the UIBB. Due to the presence of more object categories, depending on the selected line, the source object is different, making the usage of "normal" FBI Lead Selection outport impossible.

To overcome this, each TBI implementation class can declare a list of "associations":

```
interface /SCMTMS/IF_UI_TBI_CORE public .
  types:
    BEGIN OF ts_assoc,      "Association definition
      name   TYPE /bobf/obm_name,
      bo     TYPE /bobf/obm_bo_key,
      node   TYPE /bobf/obm_node_key,
      desc   TYPE char40,
    END OF ts_assoc .
  types:
    tt_assoc TYPE HASHED TABLE OF ts_assoc WITH UNIQUE KEY name .
```

At runtime, the TBI implementation class must also "resolve" the association for a set of given keys, there is a specialized method for that. Usually, this involves the evaluation of element category.

H. UI GROUPS

One of the most important features of TBI is the preference towards building the UI structure (and the internal buffer structure) out of **sub-structures included as groups** (with or without suffixes).

When looking again at the mock-up, we can see that the data from a line can be semantically split into: (1) header information, (2) departure information and (3) arrival information:

Hierarchy	Document Type	Source Location	Source Location Address	Departure Date	Departure Time	Departure Timezone	Dest. Location	Dest. Location Address	Arrival Date	Arrival Time	Arrival Timezone
Forwarding Order 100111	FWO	W-001	Warehouse Walldorf	01.07.2015	10:00:00	CET	C-666	Jane Doe / Boston	15.07.2015	16:00:00	EST
Freight Unit 501234	0001										
Truck FO 201234	1001	W-001	Warehouse Walldorf	02.07..2015	08:00:00	CET	DEHAM	Port Hamburg	02.07.2015	17:50:00	CET
Ocean Booking 15	BSEA	DEHAM	Port Hamburg	03.07.2015	15:00:00	CET	USNEK	Port Newark	10.07.2015	16:00:00	EST
US-Truck FO 301234	1002	USNEK	Port Newark	11.07.2015	08:00:00	EST	C-666	Jane Doe / Boston	11:07.2015	12:30:00	EST
Freight Unit 505678	0001										
Truck FO 205678	1001	W-001	Warehouse Walldorf	02.07..2015	16:00:00	CET	DEHAM	Port Hamburg	03.07.2015	08:00:00	CET
Ocean Booking 15	BSEA	DEHAM	Port Hamburg	03.07.2015	15:00:00	CET	USNEK	Port Newark	10.07.2015	16:00:00	EST
US-Truck FO 305678	1002	USNEK	Port Newark	11.07.2015	07:00:00	EST	C-666	Jane Doe / Boston	11:07.2015	11:30:00	EST

Similar to FBI "related views", the groups must be designed to take data from one node only. The benefits of this approach will be described later in more detail; it is enough to say now that this allows the TBI superclass to implement some very good generic logic.

In this context, the central TBI functionality maintains a so-called **field group mapping**, which describes the connection between the final fieldname (as seen by UI, including potential suffixes) and the field name of the included substructure. In this example, supposing that the "header"

substructure is included as a group named ROOT with suffix RT, the UI field named DOCUMENT_TYPERT must be linked to the field named DOCUMENT_TYPE of the group ROOT.

Definition of field group mapping:

```
types:
  BEGIN OF ts_fieldmap,
    uiname TYPE name_komp,      "name of the component in UI structure
    elem   TYPE ty_element_cat, "element category
    group  TYPE ddgroup,        "group name of the included sub-structure
    ddname TYPE name_komp,      "name of the component in the included sub-structure
    attr   TYPE string,         "name of the attribute in the backend node
    suffix TYPE char3,
  END OF ts_fieldmap .
types:
  tt_fieldmap TYPE SORTED TABLE OF ts_fieldmap WITH NON-UNIQUE KEY uiname elem
              WITH NON-UNIQUE SORTED KEY byddname COMPONENTS ddname group elem
              WITH NON-UNIQUE SORTED KEY byattr   COMPONENTS attr group elem
              WITH NON-UNIQUE SORTED KEY bygroup  COMPONENTS group elem .
```

This mapping is used when retrieving the properties and also when dispatching the UI changes back to BO.

I. KEY MAPPING

Probably the most import feature of TBI is the key mapping, based on the "UI Groups" concept mentioned above. While building the data table for UI, the TBI implementation class is supposed to fill a mapping between the UI key (the unique key of the line) and the used BO node and instance keys, *for each group*.

This mapping is defined like this:

```
interface /SCMTMS/IF_UI_TBI_CORE public .
  types:
    BEGIN OF ts_keymap,
      ui_inst_key TYPE /bobf/conf_key, "row key in UI structure
      group       TYPE ddgroup,        "group name for the included sub-structure
      bo          TYPE /bobf/obm_bo_key,
      node        TYPE /bobf/obm_node_key,
      inst_key    TYPE /bobf/conf_key,
      src_key     TYPE /bobf/conf_key, "instance for which data was req. (in IT_KEY)
      root_key    TYPE /bobf/conf_key, "root key instance
    END OF ts_keymap .
  types:
    tt_keymap TYPE SORTED TABLE OF ts_keymap
              WITH UNIQUE KEY ui_inst_key group node inst_key
              WITH NON-UNIQUE SORTED KEY node     COMPONENTS node
              WITH NON-UNIQUE SORTED KEY group    COMPONENTS group
              WITH NON-UNIQUE SORTED KEY inst_key COMPONENTS inst_key
              WITH NON-UNIQUE SORTED KEY root_key COMPONENTS root_key
              WITH NON-UNIQUE SORTED KEY src_key  COMPONENTS src_key .
```

In the example given in this guide, for each row we need to maintain up to three mapping records: one for the "header" group, one for the "departure" group and one for the "arrival" group.

J. BO ACTION DEFINITION

Another important feature is the possibility to declare BO actions that are executed generically, provided that the key mapping was defined previously. An action can be defined for one element category and one group.

The action definition (called also action mapping) is used not only during event processing, but also when mapping generically the action properties from backend to UI.

The definition is:

```
interface /SCMTMS/IF_UI_TBI_CORE public .
  types:
    BEGIN OF ts_action,
      ui_action    TYPE /scmtms/string,  "action name
      element_cat  TYPE ty_element_cat,  "element category
      group        TYPE ddgroup,         "group name for the included sub-structure
      bo_action    TYPE /bobf/act_key,
      bo_key       TYPE /bobf/obm_bo_key,
      action_param TYPE REF TO data,
      navigation   TYPE boolean,          "action is relevant for navigation
      static       TYPE boolean,          "action does not require selected rows
      lock_indep   TYPE boolean,          "action is decoupled from lock status
    END OF ts_action .
  types:
    tt_action TYPE SORTED TABLE OF ts_action
              WITH NON-UNIQUE KEY ui_action element_cat
              WITH NON-UNIQUE SORTED KEY bo_action COMPONENTS bo_action
              WITH NON-UNIQUE SORTED KEY group COMPONENTS group element_cat .
```

It should be noted that this definition allows an exceptional non-BO use case. When the Boolean property **NAVIGATION** is set, no BO action is executed. The FPM part of the TM-specific TBI superclass uses such action definition records to execute generically navigation actions (either navigate to a different page of the same document UI, or to navigate externally to the UI of a different TM document.

K. FIELD SOURCE MAPPING

At a closer look, the presented mock-up reveals a further problem. Not only that data is coming from different BO nodes, depending on element category, but also the source field names are different. For instance, "departure location" is comes from the field named LOG_LOC_ID of the TOR Stop node (for category "freight document") and from the field named SRC_LOC_ID of TRQ Root node (for category "forwarding order"). But the UI structure can have only one name, meaning that for some element categories we can't use "move corresponding", but explicit "move" statements.

When retrieving the data this is not the biggest issue, but also field properties and changes must be treated differently.

The TBI Superclass offers however a possibility to handle this generically, with the help of a **category-specific field source mapping**, as defined below:

```
class /SCMTMS/CL_UI_TBI_UTIL definition public create public .
  types:
    BEGIN OF ts_fieldsourcemap,
      group  TYPE ddgroup,    "group name of the included sub-structure
      ddname TYPE name_komp, "name of the component in the included sub-structure
      elem   TYPE /scmtms/if_ui_tbi_core=>ty_element_cat,
      attr   TYPE string,     "name of the component in the source BO
    END OF ts_fieldsourcemap .
  types:
    tt_fieldsourcemap TYPE SORTED TABLE OF ts_fieldsourcemap
                      WITH UNIQUE KEY group ddname elem .
```

When this mapping is defined, then the both property retrieval and flush handling use it – thus freeing the actual implementation of the task of handling separately these fields. The data retrieval must still be done by the implementation itself (there is no generic data retrieval in the TBI superclass).

L. SEPARATION OF UI-AGNOSTIC FROM UI-SPECIFIC HANDLING

As mentioned above, a TBI implementation is an ABAP class that implements two ABAP interfaces. Why two?

At the closer look, the basic interface is UI-technology agnostic - there are no UI-specific artefacts present in the method signatures, nor in the public attributes. It is intended that each implementation must obey this principle, and implement only UI-independent logic in there.

The FPM-specific handling, like field description, action description, drag-and-drop definition, as well as non-BO event processing (navigation, raising dialogs, reacting to closed dialogs) and UI-technology-specific specialty during data and property definition, is supposed to be done during the methods contained in the second ABAP interface.

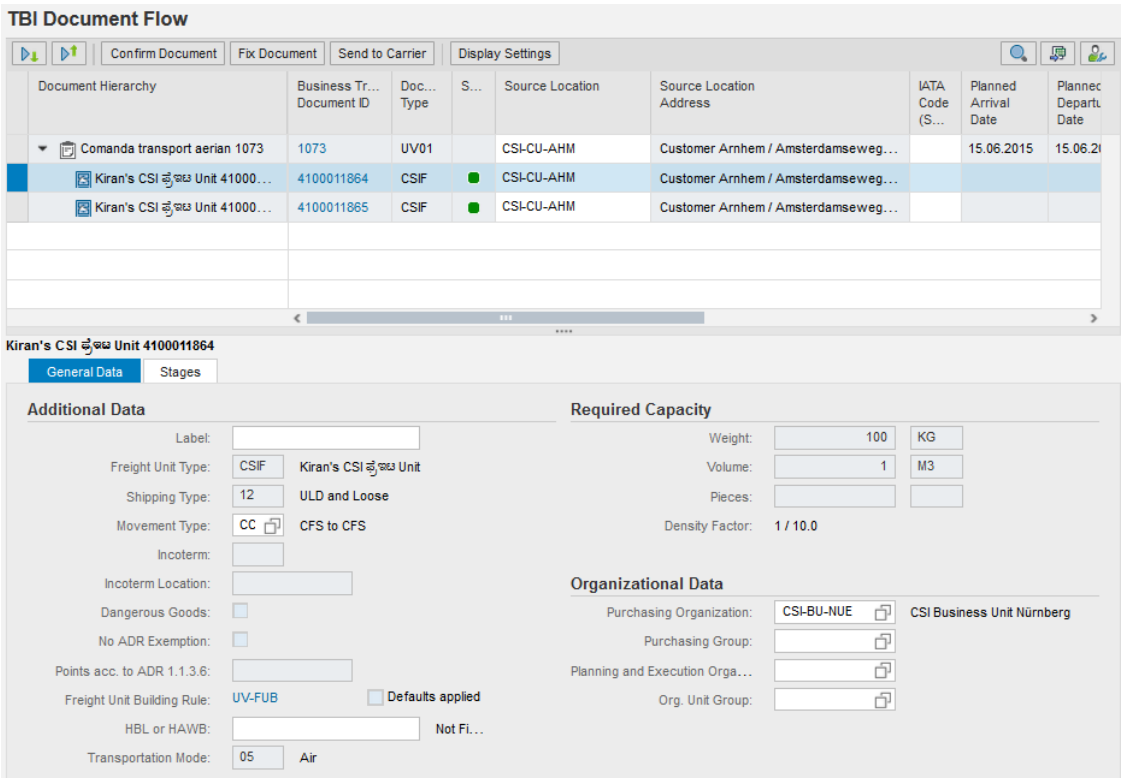
IV.CASE STUDY IMPLEMENTATION

In this chapter I will describe the necessary steps that must be performed to implement a TBI-based UIBB.

The requirement is to build two UIBBs based on the mock-up presented in the first chapter, but with the following additions:

- A column containing a hyperlink for opening the displayed documents.
- A further column containing a fixation status icon (red LED for fixed documents, green LED for the rest) – displayed only for FU and FO hierarchy levels.
- Some toolbar buttons in the hierarchy: a button "Confirm" valid only for forwarding order level, a button "Fix" valid only for freight unit and freight document levels, and a button "Send to carrier" valid only for freight document level. The buttons corresponds to BO actions and are "instance"-independent (they must be executed only for selected lines).
- Another button in the hierarchy, "open UI settings" that should open the UI settings dialog. This button does not correspond to a BO action, obviously, and should be executed regardless of the selection.
- A "detail" area underneath, in which a "general data" tab and a "stages" tab are contained; obviously, the content of this detail area depends too on the element category. We'll reuse however the existing UIBBs from the corresponding UIs.
- One UIBB is air-specific, having the IATA code in addition to the Location ID column, whereas the second one is ocean-specific, with UN/LOCODE as additional column.

The result should look like below:



A. STEP 1 – CREATE REQUIRED CLASSES

I need a class that encapsulates the logic, and a conversion class.

Technically, the only requirement towards the first one is to implement the interface **/SCMTMS/IF_UI_TBI_CORE**. Not required, but almost necessary in FPM context is the interface **/SCMTMS/IF_UI_TBI_FPM**.

There is no explicit requirement towards the format of a conversion class. The TBI class above is responsible of converting data - it can do it in every way it deems necessary.

However, for consistency reasons across TM and also for reducing implementation efforts, it is highly recommended to derive both classes from the available superclasses delivered in standard.

Therefore, I create:

```
class ZMDF_CL_UI_TBIDEMO definition
  public
  inheriting from /SCMTMS/CL_UI_TBI_CORE
  create public .
```

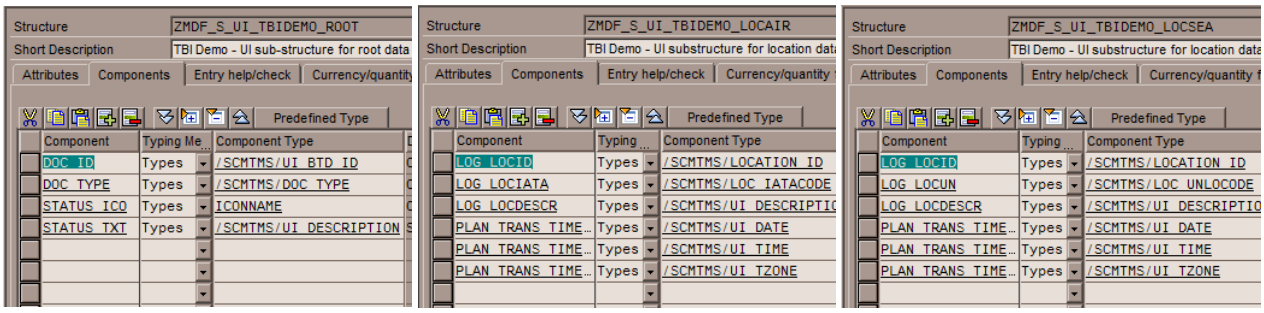
```
class ZMDF_CL_UI_TBIDEMO_CONV definition
  public
  inheriting from /SCMTMS/CL_UI_TBI_CONVERSION
  create public .
```

In the appendix, the complete source code of those two classes is listed for further reference.

B. STEP 2 – DEFINE UI STRUCTURE

We must define two main UI structures: one for air and another one for ocean. Following the principle of separation between UI and internal buffering, we include in the UI structures only the fields that are on the screen.

Following the principle of group separation, we need a substructure for root data – **ZMDF_S_UI_TBIDEMO_ROOT**, one substructure for location data air - **ZMDF_S_UI_TBIDEMO_LOCAIR** and one for location data ocean – **ZMDF_S_UI_TBIDEMO_LOCSEA**:



As such, our two UI structures **ZMDF_S_UI_TBIDEMO_AIR** and **ZMDF_S_UI_TBIDEMO_SEA** contain exclusively these substructures, along with the "tree control fields" substructure:

Structure: ZMDF_S_UI_TBIDEMO_AIR — Active

Short Description: TBI Demo - UI Structure for Air

Attributes | Components | Entry help/check | Currency/quantity fields

Predefined Type — 1

Component	Component Type	Short Description	Group
.INCLUDE	/SCMTMS/S_UI_CMN_TREECTR	Technical fields for Tree UIBBs	TBI
.INCLUDE	ZMDF_S_UI_TBIDEMO_ROOT	TBI Demo - UI sub-structure for root d...	ROOT
.INCLU-DEP	ZMDF_S_UI_TBIDEMO_LOCAIR	TBI Demo - UI substructure for location...	DEP
.INCLU-ARR	ZMDF_S_UI_TBIDEMO_LOCAIR	TBI Demo - UI substructure for location...	ARR

Structure: ZMDF_S_UI_TBIDEMO_SEA — Active

Short Description: TBI Demo - UI Structure for ocean

Attributes | Components | Entry help/check | Currency/quantity fields

Predefined Type — 1

Component	Component Type	Short Description	Group
.INCLUDE	/SCMTMS/S_UI_CMN_TREECTR	Technical fields for Tree UIBBs	TBI
.INCLUDE	ZMDF_S_UI_TBIDEMO_ROOT	TBI Demo - UI sub-structure for root d...	ROOT
.INCLU-DEP	ZMDF_S_UI_TBIDEMO_LOCSEA	TBI Demo - UI substructure for location...	DEP
.INCLU-ARR	ZMDF_S_UI_TBIDEMO_LOCSEA	TBI Demo - UI substructure for location...	ARR

Notice that every included group has a proper name. Suffixes however are required only when naming conflicts occur, in our case for location data.

When expanding the included components, the structures look like this:

Structure: ZMDF_S_UI_TBIDEMO_AIR — Active

Short Description: TBI Demo - UI Structure for Air

Attributes | Components | Entry help/check | Currency/quantity fields

Predefined Type

Component	Component Type	Short Description	Group
.INCLUDE	/SCMTMS/S_UI_CMN_TREECTR	Technical fields for Tree ...	TBI
KEY	/BOBF/CONF_KEY	NodeID	
TREE_PARENT_KEY	/BOBF/CONF_KEY	NodeID	
IS_LEAF	BOOLEAN	Boolean Variable (X=Tru...	
EXPANDED	BOOLEAN	Boolean Variable (X=Tru...	
IMAGE_SRC	FPMGB_IMAGE_SRC	Image Source	
CHILDREN_LOADED	BOOLEAN	Boolean Variable (X=Tru...	
TEXT	FPM_TEXT	FPM: Text for the Explana...	
.INCLUDE	ZMDF_S_UI_TBIDEMO_ROOT	TBI Demo - UI sub-structu...	ROOT
DOC_ID	/SCMTMS/UI_BTD_ID	Document ID	
DOC_TYPE	/SCMTMS/DOC_TYPE	Communication Structure ...	
STATUS_ICO	ICONNAME	Name of an Icon	
STATUS_TEXT	/SCMTMS/UI_DESCRIPTION	Description	
.INCLU-DEP	ZMDF_S_UI_TBIDEMO_LOCAIR	TBI Demo - UI substructur...	DEP
LOG_LOCIDDEP	/SCMTMS/LOCATION_ID	Location	
LOG_LOCIATADEP	/SCMTMS/LOC_IATACODE	International Air Transpor...	
LOG_LOCDESCRDEP	/SCMTMS/UI_DESCRIPTION	Description	
PLAN_TRANS_TIME_D...	/SCMTMS/UI_DATE	Date	
PLAN_TRANS_TIME_T...	/SCMTMS/UI_TIME	Time	
PLAN_TRANS_TIME_T...	/SCMTMS/UI_TZONE	Time Zone	
.INCLU-ARR	ZMDF_S_UI_TBIDEMO_LOCAIR	TBI Demo - UI substructur...	ARR
LOG_LOCIDARR	/SCMTMS/LOCATION_ID	Location	
LOG_LOCIATAARR	/SCMTMS/LOC_IATACODE	International Air Transpor...	
LOG_LOCDESCRARR	/SCMTMS/UI_DESCRIPTION	Description	
PLAN_TRANS_TIME_D...	/SCMTMS/UI_DATE	Date	
PLAN_TRANS_TIME_T...	/SCMTMS/UI_TIME	Time	
PLAN_TRANS_TIME_T...	/SCMTMS/UI_TZONE	Time Zone	

Structure: ZMDF_S_UI_TBIDEMO_SEA — Active

Short Description: TBI Demo - UI Structure for ocean

Attributes | Components | Entry help/check | Currency/quantity fields

Predefined Type

Component	Component Type	Short Description	Group
.INCLUDE	/SCMTMS/S_UI_CMN_TREECTR	Technical fields for Tree ...	TBI
KEY	/BOBF/CONF_KEY	NodeID	
TREE_PARENT_KEY	/BOBF/CONF_KEY	NodeID	
IS_LEAF	BOOLEAN	Boolean Variable (X=Tru...	
EXPANDED	BOOLEAN	Boolean Variable (X=Tru...	
IMAGE_SRC	FPMGB_IMAGE_SRC	Image Source	
CHILDREN_LOADED	BOOLEAN	Boolean Variable (X=Tru...	
TEXT	FPM_TEXT	FPM: Text for the Explana...	
.INCLUDE	ZMDF_S_UI_TBIDEMO_ROOT	TBI Demo - UI sub-structu...	ROOT
DOC_ID	/SCMTMS/UI_BTD_ID	Document ID	
DOC_TYPE	/SCMTMS/DOC_TYPE	Communication Structure ...	
STATUS_ICO	ICONNAME	Name of an Icon	
STATUS_TEXT	/SCMTMS/UI_DESCRIPTION	Description	
.INCLU-DEP	ZMDF_S_UI_TBIDEMO_LOCSEA	TBI Demo - UI substructur...	DEP
LOG_LOCIDDEP	/SCMTMS/LOCATION_ID	Location	
LOG_LOCUNDEP	/SCMTMS/LOC_UNLOCODE	United Nations Code for T...	
LOG_LOCDESCRDEP	/SCMTMS/UI_DESCRIPTION	Description	
PLAN_TRANS_TIME_...	/SCMTMS/UI_DATE	Date	
PLAN_TRANS_TIME_...	/SCMTMS/UI_TIME	Time	
PLAN_TRANS_TIME_...	/SCMTMS/UI_TZONE	Time Zone	
.INCLU-ARR	ZMDF_S_UI_TBIDEMO_LOCSEA	TBI Demo - UI substructur...	ARR
LOG_LOCIDARR	/SCMTMS/LOCATION_ID	Location	
LOG_LOCUNARR	/SCMTMS/LOC_UNLOCODE	United Nations Code for T...	
LOG_LOCDESCRARR	/SCMTMS/UI_DESCRIPTION	Description	
PLAN_TRANS_TIME_...	/SCMTMS/UI_DATE	Date	
PLAN_TRANS_TIME_...	/SCMTMS/UI_TIME	Time	
PLAN_TRANS_TIME_...	/SCMTMS/UI_TZONE	Time Zone	

For internal processing reasons, our TBI class must hold more data than displayed on the screen. Therefore we need an additional structure (and table type). There is not required to define them in data dictionary, though. To avoid maintenance overhead, I prefer to declare them as class-types. However, I stick to the concept of group separation, therefore we need for each segment on own internal type:

```
types:
  begin of TS_ROOT_INT .
    include type ZMDF_S_UI_TBIDEMO_ROOT.
types:
    doc_key type /bobf/conf_key,
    doc_cat type /scmtms/doc_category,
    status  type /scmtms/tor_fixation_status,
  end of TS_ROOT_INT .

types:
  begin of TS_LOC_INT .
    include type ZMDF_S_UI_TBIDEMO_LOCAIR.
types:
    log_loc_uuid    type /bobf/conf_key,
    log_locun       type /scmtms/loc_unlocode,
    plan_trans_time type /scmtms/stop_plan_date,
  end of TS_LOC_INT .

types:
  begin of TS_DATA_INT,
    element_cat type /scmtms/if_ui_tbi_core=>ty_element_cat.
    include type /scmtms/s_ui_cmn_treectr as tbi.
    include type ts_root_int as root .
    include type ts_loc_int as dep renaming with suffix dep .
    include type ts_loc_int as arr renaming with suffix arr .
types:
  end of TS_DATA_INT ,
  TT_DATA_INT type standard table of TS_DATA_INT
              with non-unique sorted key KEY components KEY .
```

Notice the extra fields: element category, root key of the displayed documents (needed for navigation), the actual status value, the location key and the full timestamps (needed for conversions). The internal type for location contains both air and ocean specific fields (in these cases, UN/LOCODE and IATA code).

The group names are added to the public section of the class. Since I am here, I create some other constants (for element categories and for UI actions needed later):

```
public section.
  constants:
    begin of SC_ELEM_CAT,
      fwo type /scmtms/if_ui_tbi_core=>ty_element_cat value 'FW',
      fu  type /scmtms/if_ui_tbi_core=>ty_element_cat value 'FU',
      fo  type /scmtms/if_ui_tbi_core=>ty_element_cat value 'FO',
    end of sc_elem_cat .
  constants:
    begin of SC_GROUP,
      root      type ddgroup value 'ROOT',
      departure type ddgroup value 'DEP',
      arrival   type ddgroup value 'ARR',
    end of sc_group .
  constants:
    begin of SC_ACTION,
      confirm   type char30 value 'TBI_CONFIRM',
      fix       type char30 value 'TBI_FIX',
      send      type char30 value 'TBI_SEND',
      uisett    type char30 value 'TBI_OPEN_UISETT',
      doc_id    type char30 value 'DOC_ID',
    end of SC_ACTION .
  constants:
    begin of SC_ASSOC,
      trq_root  type string value 'TRQ Root',
      tor_root  type string value 'TOR Root',
    end of SC_ASSOC .
```

For actions, I used CHAR30 since this is the exact length of the data element FPM_EVENT_ID. We will consume this class in an FPM-based screen, but I wanted to follow the separation principle and decouple this constant from the FPM data element.

The value 'DOC_ID' of the "action" constant is intentionally equal to the component **DOC_ID** of the UI structure – since this column is to be displayed as a hyperlink, I am going to reuse a concept

similar to the one in FBI, where clicks on hyperlinks are transformed in FPM events with an ID equal to the name of the affected column. This approach is not mandatory; technically one could intercept the events 'FPM_GUIBB_TREE_CELL_ACTION' and 'FPM_GUIBB_LIST_CELL_ACTION' and perform own handling, but in my example it would have been a waste to skip the generic handling of the super class (more to that, later).

C. STEP 3 – DEFINE CONVERSION

As mentioned, I am using the standard delivered superclass, since in this case it suffices to declare the mapping (conversion) rules in method **ADAPT_MAPTABLE_SUBSTR**.

The superclass is based on the normal FBI conversion class (it is actually derived from the TM-specific **/SCMTMS/CL_UI_CONVERSION**), therefore the usual concepts apply.

There is one fundamental difference, though: the conversion is executed separately, for each group – as each group is supposed to represent a set of attributes come from one BO node. At this point, the field names do not contain the suffixes that may exist when including the groups in the main structure. This allows a certain reuse.

```
METHOD adapt_maptable_substr.

  CASE iv_group.
    WHEN zmdf_cl_ui_tbidemo=>sc_group-root.
      "normally, one would have created a conversion doc_key <-> doc_id,
      "but since this field isn't editable on the screen, we can skip this

    WHEN zmdf_cl_ui_tbidemo=>sc_group-departure
      OR zmdf_cl_ui_tbidemo=>sc_group-arrival.
      "conversion rule for location
      INSERT VALUE #(
        conv_cat    = sc_conv_cat-uuid
        node_field1 = 'LOG_LOC_UUID'
        ui_field1   = 'LOG_LOCID'
        ui_field2   = 'LOG_LOCUN'
        ui_field4   = 'LOG_LOCIATA'
        conf_1      = /scmtms/if_location_c=>sc_bo_key
        conf_2      = /scmtms/if_location_c=>sc_node-root
        conf_3      = /scmtms/if_location_c=>sc_alternative_key-root-location_id
        msgid       = '/SCMTMS/MSG'
        msgno       = '003' ) INTO TABLE ct_map_data.

      "conversion rule for location address
      INSERT VALUE #(
        conv_cat    = sc_conv_cat-address
        node_field1 = 'LOG_LOC_UUID'
        ui_field1   = 'LOG_LOCDESCR'
        conf_1      = /scmtms/if_location_c=>sc_bo_key ) INTO TABLE ct_map_data.

     "conversion rule for transp. time
      READ TABLE ct_map_data ASSIGNING FIELD-SYMBOL(<map_data>)
        WITH KEY conv_cat    = sc_conv_cat-timestamp
                 node_field1 = 'PLAN_TRANS_TIME'.
      IF sy-subrc EQ 0.
        <map_data>-conf_1 = 'LOG_LOC_UÙID'.
      ELSE.
        INSERT VALUE #(
          conv_cat    = sc_conv_cat-timestamp
          node_field1 = 'PLAN_TRANS_TIME'
          ui_field1   = 'PLAN_TRANS_TIME_D'
          ui_field2   = 'PLAN_TRANS_TIME_T'
          ui_field3   = 'PLAN_TRANS_TIME_TZ'
          conf_1      = 'LOG_LOC_UUID' ) INTO TABLE ct_map_data.
      ENDIF.

    WHEN others.
```

```
    ENDCASE.

  ENDMETHOD.
```

Some further remarks:

- For the "node field" parts of the rules, I should have used the BO constants, but for readability reasons I exceptionally used literals.
- The conversion between **KEY** and **ID** is not required in this case, since the UI field is not editable (it'll be a hyperlink) and I'll make sure that when data is retrieved both fields are filled.
- The conversion superclass adds mapping rules based on naming convention, that's why I had to write special coding for **PLAN_TRANS_TIME** (in this case, location key is required for location-specific time zone and the superclass had no chance to guess which field is the one needed); that is also why there was no need for an explicit "code list" conversion between **STATUS** and **STATUS_TXT**.

D. STEP 4 – FINISH DESIGN-TIME OPERATIONS

It's time I finished the "declarative" methods of the TBI class. Normally, that means writing coding in the "core" methods **INITIALIZE** and **DESCRIBE** and in the "FPM specific" method **ADAPT_DEFINITION**.

INITIALIZE does what the name suggest – merely taking over the optional parameters from the consumer (we'll see later how to define them). In this example, one parameter of high importance is the MOT category (air/ocean).

But since I want to use the superclass, I need to redefine the method **_INIT_INTERNAL** (fill all exporting parameters) – the rest of the initialization is done by the superclass.

First, define an internal container for data (the "buffer") and create a redefinition for the needed method:

```
  data MT_DATA   type tt_data_int .
  data MV_MOTCAT type /scmtms/trmodcat .

  methods _INIT_INTERNAL
    redefinition .
```

In the coding, after the preparatory steps (clear exporting parameters, call superclass), I add these rather straightforward coding lines:

```
  GET REFERENCE OF mt_data INTO er_data.
  es_object-bo   = /scmtms/if_trq_c=>sc_bo_key.
  es_object-node = /scmtms/if_trq_c=>sc_node-root.
  ev_convclass   = 'ZMDF_CL_UI_TBIDEMO_CONV'.

  et_elemcat = VALUE #(
    ( sc_elem_cat-fwo ) ( sc_elem_cat-fu ) ( sc_elem_cat-fo ) ).
```

The reference to data table must be returned to the caller of this method (the superclass), since it needs to be exposed as interface attribute **/SCMTMS/IF_UI_TBI_CORE=>MR_DATA**.

In the next step, I define the supported BO actions. One of them has parameter structure, which is to be filled with a default value. Notice that each action is element category specific. The group helps the generic coding of the superclass to determine the actual BO instance key for which the action is to be executed (the key mapping is used).

```
  DATA lr_par_confirm TYPE REF TO /scmtms/s_trq_a_confirm.
```

```
CREATE DATA lr_par_confirm.
lr_par_confirm->no_check = abap_true.
et_action = VALUE #(
  ( ui_action    = sc_action-confirm
    element_cat  = sc_elem_cat-fwo
    group        = sc_group-root
    bo_key       = /scmtms/if_trq_c=>sc_bo_key
    bo_action    = /scmtms/if_trq_c=>sc_action-root-confirm
    action_param = lr_par_confirm
    )
  ( ui_action    = sc_action-fix
    element_cat  = sc_elem_cat-fu
    group        = sc_group-root
    bo_key       = /scmtms/if_tor_c=>sc_bo_key
    bo_action    = /scmtms/if_tor_c=>sc_action-root-fix_tor
    )
  ( ui_action    = sc_action-fix
    element_cat  = sc_elem_cat-fo
    group        = sc_group-root
    bo_key       = /scmtms/if_tor_c=>sc_bo_key
    bo_action    = /scmtms/if_tor_c=>sc_action-root-fix_tor
    )
  ( ui_action    = sc_action-send
    element_cat  = sc_elem_cat-fo
    group        = sc_group-root
    bo_key       = /scmtms/if_tor_c=>sc_bo_key
    bo_action    = /scmtms/if_tor_c=>sc_action-root-send_tor
    )
 ) .
```

Another functionality that the superclass offers is to execute automatically navigation operations, provided that all required information is available in the internal data table. In some cases, the generic handling might be too simple, but in this example it fits perfectly. All I need is to add further records to the action mapping table (notice the “navigation” flag):

```
"define TOR_ID as navigation column
DATA lr_par_navigation TYPE REF TO tt_parameter.
CREATE DATA lr_par_navigation.
lr_par_navigation->* = VALUE #(
  ( name  = 'CHANGE_MODE'
    value = space
    )
  ( name  = 'KEY'
    value = '(DOC_KEY)'
    )
  ( name  = 'CATEGORY'
    value = '(DOC_CAT)'
    )
).
INSERT VALUE #(
  ui_action    = sc_action-doc_id
  element_cat  = sc_elem_cat-fwo
  group        = sc_group-root
  bo_key       = /scmtms/if_trq_c=>sc_bo_key
  navigation   = abap_true
  action_param = lr_par_navigation
  ) INTO TABLE et_action.
INSERT VALUE #(
  ui_action    = sc_action-doc_id
  element_cat  = sc_elem_cat-fu
  group        = sc_group-root
  bo_key       = /scmtms/if_tor_c=>sc_bo_key
  navigation   = abap_true
  action_param = lr_par_navigation
  ) INTO TABLE et_action.
INSERT VALUE #(
  ui_action    = sc_action-doc_id
  element_cat  = sc_elem_cat-fo
  group        = sc_group-root
  bo_key       = /scmtms/if_tor_c=>sc_bo_key
  navigation   = abap_true
  action_param = lr_par_navigation
  ) INTO TABLE et_action.
```

The next step concerns support for the generic notification handling. TBI classes are called too often (at every event), and therefore they must not compute data every time again and again. The idea is: once computed, the data from MT_DATA is to be returned "as is" to the consumer, until it is invalidated by change notifications from backend. The TBI superclass implements also the interface **/SCMTMS/IF_UI_TBI_NOTIF**, which offers the method **RECEIVE_NOTIFICATION**. This method should be called by the consumer, which must provide the change notifications at every interaction with BO layer.

The consumer is this case the FPM-based UI served by the FBI controller, which indeed sends notifications at every interaction. The TBI superclass by default monitors all BO node instances specified in the key mapping. However, when new node instances are created by another function, some of them may be relevant for the TBI class. In the current example, when a new FU is created for this FWO, the data of this UIBB becomes invalid.

Therefore, we need to tell the superclass, which are the nodes that are to be monitored for instance creation:

```
et_notifnode = VALUE #(
  ( node_key = /scmtms/if_tor_c=>sc_node-root
    create_is_relevant = abap_true )
  ( node_key = /scmtms/if_tor_c=>sc_node-stop
    create_is_relevant = abap_true )
).
```

The last step in **_INIT_INTERNAL** concerns the trickiest feature of the TBI superclass.

As mentioned previously, the information in LOG_LOC_UUID comes from the similarly named component of the TOR Stop structure (in case of FU/FO element categories) and from the field named SRC_LOC_KEY of the TRQ Root Structure (in case of FWO element category). This does not concern only getting data, but also getting field properties and flushing back changes.

One way to do this is to write own complicated logic in the specific implementation of method GET_PROPERTY and FLUSH from the TBI core interface.

To avoid that, I use the superclass generic handling, which however requires the following field source mapping table:

```
et_fieldsource = VALUE #(
  ( group    = sc_group-departure
    elem     = sc_elem_cat-fwo
    ddname   = 'LOG_LOC_UUID'
    attr     = /scmtms/if_trq_c=>sc_node_attribute-root-src_loc_key
    )
  ( group    = sc_group-departure
    elem     = sc_elem_cat-fwo
    ddname   = 'LOG_LOCID'
    attr     = /scmtms/if_trq_c=>sc_node_attribute-root-src_loc_id
    )
  ( group    = sc_group-departure
    elem     = sc_elem_cat-fwo
    ddname   = 'LOG_LOCIATA'
    attr     = /scmtms/if_trq_c=>sc_node_attribute-root-src_loc_iatacode
    )
  ( group    = sc_group-departure
    elem     = sc_elem_cat-fwo
    ddname   = 'LOG_LOCUN'
    attr     = /scmtms/if_trq_c=>sc_node_attribute-root-src_loc_unlocode
    )
  ( group    = sc_group-departure
    elem     = sc_elem_cat-fwo
    ddname   = 'PLAN_TRANS_TIME'
    attr     = /scmtms/if_trq_c=>sc_node_attribute-root-pic_ear_req
    )
  ( group    = sc_group-arrival
    elem     = sc_elem_cat-fwo
```

```
      ddname   = 'LOG_LOC_UUID'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-des_loc_key
      )
    ( group    = sc_group-arrival
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOCID'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-des_loc_id
      )
    ( group    = sc_group-arrival
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOCIATA'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-des_loc_iatacode
      )
    ( group    = sc_group-arrival
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOCUN'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-des_loc_unlocode
      )
    ( group    = sc_group-arrival
      elem     = sc_elem_cat-fwo
      ddname   = 'PLAN_TRANS_TIME'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-del_lat_req
      )
  ).
```

Method **DESCRIBE** passes to the consumer some identification information (object key, UI structure and supported associations).

I redefine the method:

```
methods /SCMTMS/IF_UI_TBI_CORE~DESCRIBE
  redefinition .
```

I add the call to superclass (important, since it already fills some exporting parameters based on stuff we initialized previously), then I return to the consumer the UI structure descriptor. Remember, this is dependent on a parameter that the consumer must provide (during initialization). All parameters were automatically stored by superclass in **MT_PARAMETER**:

```
READ TABLE mt_parameter ASSIGNING FIELD-SYMBOL(<param>)
  WITH KEY name = 'MOTCAT'.
IF sy-subrc EQ 0 AND
   <param>-value EQ /scmtms/if_common_c=>c_tr_mode_category-air.
  eo_field_description ?=
    cl_abap_structdescr=>describe_by_name( 'ZMDF_S_UI_TBIDEMO_AIR').
ELSE.
  eo_field_description ?=
    cl_abap_structdescr=>describe_by_name( 'ZMDF_S_UI_TBIDEMO_SEA').
ENDIF.
```

And finally, I need to provide the supported associations:

```
et_association = VALUE #(
  ( name = sc_assoc-trq_root
    bo   = /scmtms/if_trq_c=>sc_bo_key
    node = /scmtms/if_trq_c=>sc_node-root
    desc = 'TRQ Root'(001)
    )
  ( name = sc_assoc-tor_root
    bo   = /scmtms/if_tor_c=>sc_bo_key
    node = /scmtms/if_tor_c=>sc_node-root
    desc = 'TOR Root'(002)
    )
).
```

The implementation of the logic behind association is done later (there is a dedicated method for that), but what I do not is inform the consumer that I can provide, for any given UI keys, the keys of relevant TRQ Root and TOR root instances.

Based on the "separation of UI-agnostic from UI-specific handling" principle, the two "core" methods presented above provide the absolutely required information / initialization steps. The extension method **ADAPT_DEFINITION** is called from the FPM feeder class and is used to provide FPM-specific information back to it. Its signature mirrors very closely the signature of **GET_DEFINITION** methods of the FPM feeder interfaces.

```
methods /SCMTMS/IF_UI_TBI_FPM~ADAPT_DEFINITION
  redefinition .
```

I use it to provide descriptions /tooltips for the columns (in this example, it is required, since the structure for location data is used for two groups and without this step we would have had confusing column names; this could have been avoided by defining two separate substructures, with dedicated data elements for "departure" and "arrival" information, but then again, I want to avoid DDIC as much as possible). Additionally, here is the place to specify the tooltip reference field for the icon column.

```
DATA(lt_field) = VALUE /scmtms/cl_ui_tbi_util=>tt_fdescr(
  ( f = 'LOG_LOCIDDEP'          tx = '/SCMTMS/UI_CMN/SOURCE_LOCATION' )
  ( f = 'LOG_LOCIDARR'          tx = '/SCMTMS/UI_CMN/DESTINATION_LOCATION' )
  ( f = 'LOG_LOCUNDEP'          tx = '/SCMTMS/UI_CMN/UNLOCODE_SOURCE' )
  ( f = 'LOG_LOCUNARR'          tx = '/SCMTMS/UI_CMN/UNLOCODE_DESTINATION' )
  ( f = 'LOG_LOCIATADEP'        tx = '/SCMTMS/UI_CMN/IATA_SOURCE' )
  ( f = 'LOG_LOCIATAARR'        tx = '/SCMTMS/UI_CMN/IATA_DESTINATION' )
  ( f = 'LOG_LOCDESCRDEP'       tx = '/SCMTMS/UI_CMN/SOURCE_LOC_ADDRESS' )
  ( f = 'LOG_LOCDESCRARR'       tx = '/SCMTMS/UI_CMN/DESTIN_LOC_ADDRESS' )
  ( f = 'PLAN_TRANS_TIME_DDEP'  tx = '/SCMTMS/UI_CMN/PLAN_DEP_DATE' )
  ( f = 'PLAN_TRANS_TIME_TDEP'  tx = '/SCMTMS/UI_CMN/PLAN_DEP_TIME' )
  ( f = 'PLAN_TRANS_TIME_TZDEP' tx = '/SCMTMS/UI_CMN/PLAN_DEP_TZONE' )
  ( f = 'PLAN_TRANS_TIME_DARR'  tx = '/SCMTMS/UI_CMN/PLANNED_ARRIVAL_DATE' )
  ( f = 'PLAN_TRANS_TIME_TARR'  tx = '/SCMTMS/UI_CMN/PLANNED_ARRIVAL_TIME' )
  ( f = 'PLAN_TRANS_TIME_TZARR' tx = '/SCMTMS/UI_CMN/PLANNED_ARRIVAL_TZONE' )
  ( f = 'STATUS_ICO' tx = '/SCMTMS/UI_CMN/STAT' tr = 'STATUS_TXT') "status icon
).
/scmtms/cl_ui_tbi_util=>chg_fcat_t(
  EXPORTING
    it_text  = lt_field
  CHANGING
    ct_descr = ct_field_descr_tree ).
```

Notice the utility method used to simplify the data transfer towards CT_FIELD_DESCR_TREE, as well as the fact that now the field names contain the suffixes.

The actions defined previously during initializations are automatically added to the action catalogue, but with the texts available in BO meta-model. If needed, this is the place to add more appropriate text. Additionally, we can add the new UI-specific action for the settings dialog:

```
DATA(lt_action) = VALUE /scmtms/cl_ui_tbi_util=>tt_adescr(
  ( id = sc_action-confirm  tx = '/SCMTMS/UI_CMN/CONFIRM_DOCUMENT' )
  ( id = sc_action-fix      tx = '/SCMTMS/UI_CMN/FIX_DOCUMENT' )
  ( id = sc_action-send     tx = '/SCMTMS/UI_CMN/TOR_SEND_CARRIER' )
  ( id = sc_action-uisett   tx = '/SCMTMS/UI_CMN/UISETT' )
).
/scmtms/cl_ui_tbi_util=>chg_action(
  EXPORTING
    it_action_descr      = lt_action
  CHANGING
    ct_action_definition = ct_action_definition ).
```

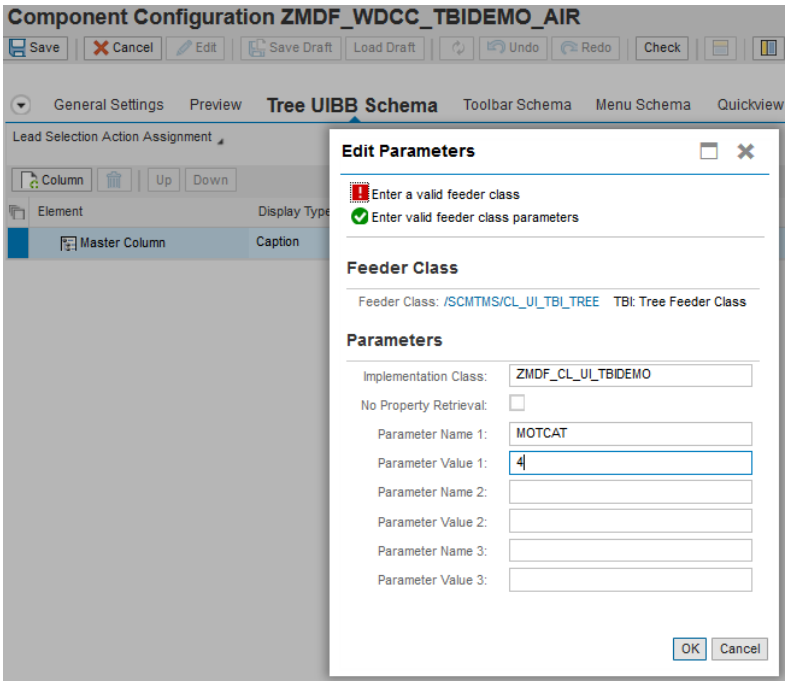
You may have deducted that the utility method can handle both situations: when a specific action is already present in **CT_ACTION_DEFINITION**, and when it's not.

I reached the point where our TBI is complete from a "design-time" perspective – now I can use it to design the UIBB in the FPM editor.

E. STEP 5 – CREATE UIBBS IN FPM EDITOR

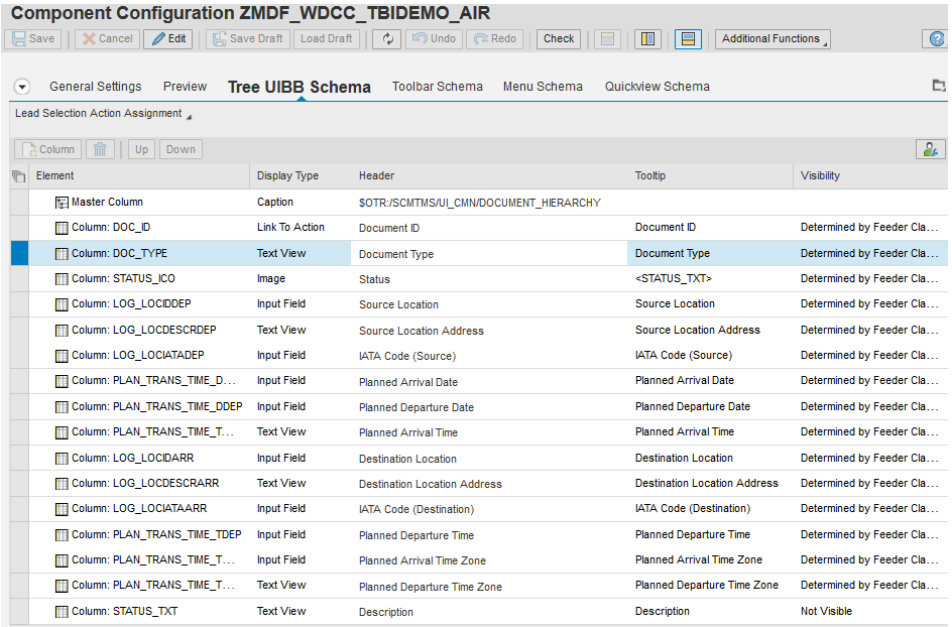
In the well-known CONFIGURE_COMPONENT application, I create a new configuration for the component FPM_TREE_UIBB, this time for air (`ZMDF_WDCC_TBIDEMO_AIR`).

The required FPM feeder class is `/SCMTMS/CL_UI_TBI_TREE`. In the feeder parameters dialog, I provide our TBI class, and also the parameter for MOT category, exactly as it is expected by the coding from `DESCRIBE`.



In the General Settings panel, I activate horizontal scrolling (by disabling the checkbox Fit Width) and activate some personalization features.

Now it's time to add the columns – not forgetting to use the proper UI controls (Image for the status icon, Link to Action for the hyperlink, Text View for the descriptive columns and Input Field for the rest).



After adding buttons for all exposed actions, I save the configuration.

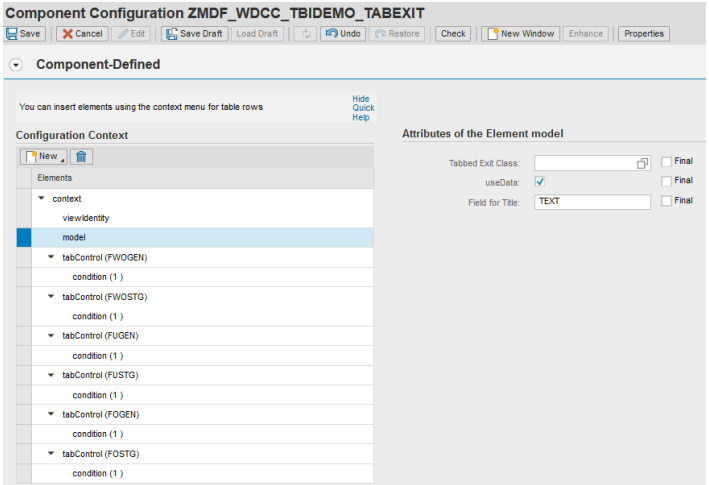
The same steps must be done for the ocean specific configuration (`ZMDF_WDCC_TBIDEMO_SEA`).

F. STEP 6 – CREATE TABBED UIBB FOR MASTER-DETAIL PATTERN

In CONFIGURE_COMPONENT, I create a new configuration for the component FPM_TABBED_UIBB – again, I start with Air (`ZMDF_WDCC_TBIDEMO_TABAIR`).

The puzzle pieces:

- A master list – in this case the new Tree configuration.
- A Tabbed Exit configuration, needed to control the visibility of the individual detail tabs (nothing TBI specific, it's the old known FBI Tabbed Exit feature).



- Six "detail" tabs:
  - General Data for forwarding order element category; I reuse the standard UIBB from Air FWO screen - `/SCMTMS/WDCC_FWO_AIR_GEN`
  - General Data for freight unit element category; I reuse the standard UIBB from FU screen - `/SCMTMS/WDCC_FU_GEN_AD`
  - General Data for freight document element category; I reuse the standard UIBB from Road FO screen - `/SCMTMS/WDCC_FOR_GEN_GNINF`
  - Stages for forwarding order element category; I reuse the "Ordered Route" list from Air FWO screen - `/SCMTMS/WDCC_FWM_AIR_ORDSTG_TREE`
  - Stages for freight unit element category ; I reuse the "Stages" list from FU screen - `/SCMTMS/WDCC_FU_STG`
  - Stages for freight document element category; I reuse the "Stages" list from Road FO screen - `/SCMTMS/WDCC_FOROAD_STG`

Component Configuration ZMDF_WDCC_TBIDEMO_TABAIR, Variant "VARIANT_1"

Save | Cancel | Edit | Save Draft | Load Draft | Undo | Redo | Check | Add

The data was saved

General Settings | Preview | **Tabbed UIBB Schema** | Wire Schema

Master UIBB | Tab | UIBB | Up | Down

Element	Component	Config ID	Window Name
Master UIBB: TREE_WINDOW	FPM_TREE_UIBB	ZMDF_WDCC_TBIDEMO_AIR	TREE_WINDOW
Master UIBB: W_MAIN	/BOFU/WDC_FBI_TABBED_EXIT	ZMDF_WDCC_TBIDEMO_TABEXIT	W_MAIN
Tab: General Data			
UIBB: FORM_WINDOW	FPM_FORM_UIBB_GL2	/SCMTMS/WDCC_FWO_AIR_GEN	FORM_WINDOW
Tab: General Data			
UIBB: FORM_WINDOW	FPM_FORM_UIBB_GL2	/SCMTMS/WDCC_FU_GEN_AD	FORM_WINDOW
Tab: General Data			
UIBB: FORM_WINDOW	FPM_FORM_UIBB_GL2	/SCMTMS/WDCC_FOR_GEN_GNINF	FORM_WINDOW
Tab: Ordered Route			
UIBB: TREE_WINDOW	FPM_TREE_UIBB	/SCMTMS/WDCC_FWM_AIR_ORDST ...	TREE_WINDOW
Tab: Stages			
UIBB: TREE_WINDOW	FPM_TREE_UIBB	/SCMTMS/WDCC_FU_STG	TREE_WINDOW
Tab: Stages			
UIBB: LIST_WINDOW	FPM_LIST_UIBB_ATS	/SCMTMS/WDCC_FOROAD_STG	LIST_WINDOW

I am misusing the road freight order UIBBs also for Bookings (technically all are TOR documents). In a better implementation, one should add further detail tabs - for all mode-specific flavours of the freight documents.

When wiring, notice that all associations declared inside `_INIT_INTERNAL` are automatically available as "Lead Selection" out-ports. This "magic" is accomplished by the feeder class; I didn't have to do anything (apart from declaring the associations, of course).

Component Configuration ZMDF_WDCC_TBIDEMO_TABAIR, Variant "VARIANT_1"

Save | Cancel | Edit | Save Draft | Load Draft | Undo | Redo | Check | Additional Functions

The data was saved

General Settings | Preview | Tabbed UIBB Schema | **Wire Schema**

Wire | Graphical Wire Editor

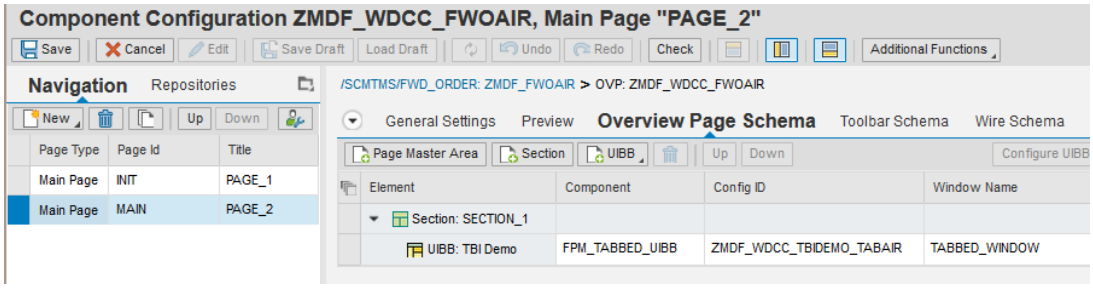
Element	Compo...	Config ID	Sour...	Source Config Name	Port Type	Port Ident...	Connector Class
Wire: /BOF...	/BOFU/ ...	ZMDF_WDCC_TBIDEMO_TABEXIT	FPM_T...	ZMDF_WDCC_TBIDEMO_AIR	Lead Sele...	LS	/BOFU/CL_FBI_CONNECTOR
Wire: Form...	FPM_FO...	/SCMTMS/WDCC_FWO_AIR_GEN	FPM_T...	ZMDF_WDCC_TBIDEMO_AIR	Lead Sele...	TRQ Root	/BOFU/CL_FBI_CONNECTOR
Wire: Tree ...	FPM_TR...	/SCMTMS/WDCC_FWM_AIR_ORDS...	FPM_T...	ZMDF_WDCC_TBIDEMO_AIR	Lead Sele...	TRQ Root	/BOFU/CL_FBI_CONNECTOR
Wire: Form...	FPM_FO...	/SCMTMS/WDCC_FU_GEN_AD	FPM_T...	ZMDF_WDCC_TBIDEMO_AIR	Lead Sele...	TOR Root	/BOFU/CL_FBI_CONNECTOR
Wire: Tree ...	FPM_TR...	/SCMTMS/WDCC_FU_STG	FPM_T...	ZMDF_WDCC_TBIDEMO_AIR	Lead Sele...	TOR Root	/BOFU/CL_FBI_CONNECTOR
Wire: Form...	FPM_FO...	/SCMTMS/WDCC_FOR_GEN_GNINF	FPM_T...	ZMDF_WDCC_TBIDEMO_AIR	Lead Sele...	TOR Root	/BOFU/CL_FBI_CONNECTOR
Wire: List /...	FPM_LIS...	/SCMTMS/WDCC_FOROAD_STG	FPM_T...	ZMDF_WDCC_TBIDEMO_AIR	Lead Sele...	TOR Root	/BOFU/CL_FBI_CONNECTOR
Wire Plug	FPM_TR...	ZMDF_WDCC_TBIDEMO_AIR					

Save everything and create a separate Tabbed UIBB configuration for ocean (`ZMDF_WDCC_TBIDEMO_TABSEA`).

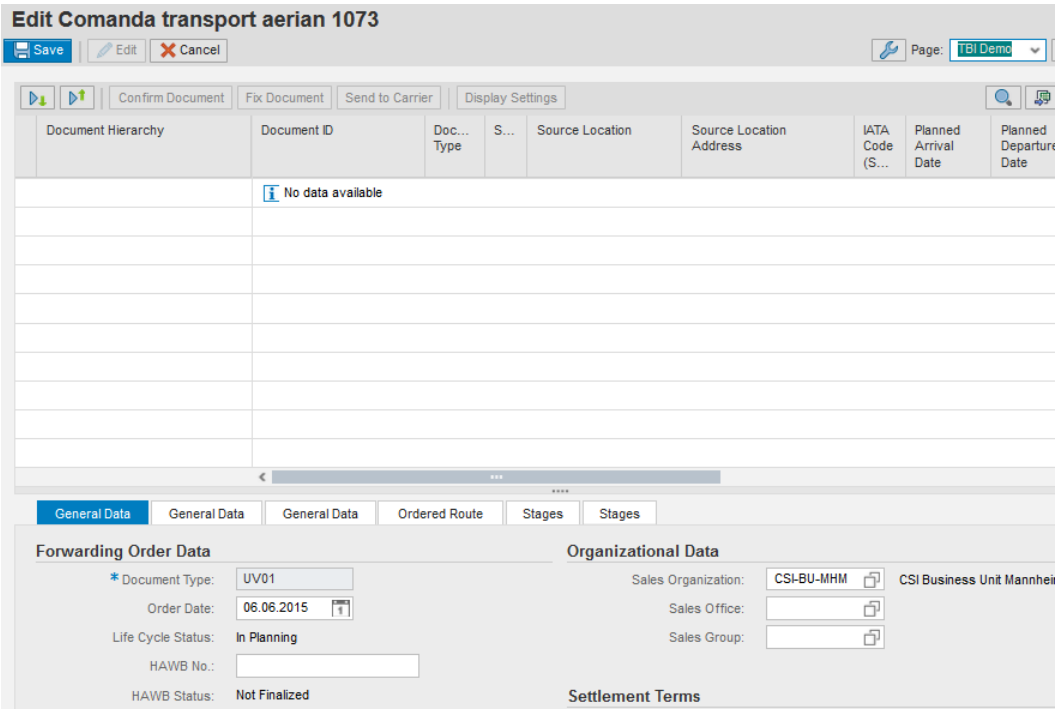
G. STEP 7 – INTEGRATE NEW UIBB INTO A FLOORPLAN CONFIGURATION

Nothing TBI specific at this step – the newly created UIBB must be included and wired in the desired floorplan (OVP) configuration.

For this exercise, I created my own application configurations (`ZMDF_FWOAIR` and `ZMDF_FWOSEA`), with my own simplified OVP configurations (`ZMDF_WDCC_FWOAIR` and `ZMDF_WDCC_FWOSEA`), which I attached to my own FWO types.



If we ran the screen immediately, we can see that the UIBBs are there, but empty. That is normal, since we didn't implement any retrieval methods yet. The fact that the UIBB is otherwise functional can be seen from the fact that the TRQ root key was successfully passed to the "detail" area:



(The fact that the detail area was visible is not normal. I took out – for demo purposes - the tabbed exit configuration from the tabbed UIBB; in the final implementation, however, there is no detail unless a "real" line is selected in the hierarchy).

H. STEP 8 – IMPLEMENT DATA RETRIEVAL METHODS

Now it's time to delve into data retrieval. I redefine **GET_DATA**:

```
methods /SCMTMS/IF_UI_TBI_CORE~GET_DATA
  redefinition .
```

Then use the following coding:

```
METHOD /scmtms/if_ui_tbi_core~get_data.
  CLEAR:
    et_data,
    ev_changed.

  IF mt_data IS INITIAL.
    ev_changed = abap_true.
    get_data_int(
      EXPORTING
        it_key = it_key ).
  ENDIF.

  MOVE-CORRESPONDING mt_data TO et_data.
ENDMETHOD.
```

As you can see, at each roundtrip the content of MT_DATA is moved to the output table, but the member attribute is computed only when it's empty; this happens either at the first time (logically), or after the superclass implementation of the interface method **/SCMTMS/IF_UI_TBI_NOTIF~RECEIVE_NOTIFICATION** clears it, as the backend changes might have invalidated it (based on the key mapping and on the nodes I registered for).

The difficulty is now to write coding into **GET_DATA_INT**, which if course must be defined first:

```
methods GET_DATA_INT
  importing
    !IT_KEY type /BOBF/T_FRW_KEY .
```

You may remember that the TBI class declares (via **DESCRIBE**) that it is a "TRQ Root", therefore the parameter IT_KEY contains the key(s) of the root node. In my example, it'll be one key, as I have integrated the UIBB in a single document UI, but there is nothing that prevents me wire this UIBB from a list of forwarding orders.

The method **GET_DATA_INT** collects the forwarding order data, then the freight unit data and then the freight document data into a temporary "raw" table, of the same type as **MT_DATA**. Then the conversion is executed, building thus the member attribute.

The data retrieval is rather trivial; it's just following some associations.

Afterwards, the records corresponding to forwarding orders are added:

```
LOOP AT lt_d_trqroot ASSIGNING FIELD-SYMBOL(<trqroot>).

  INSERT VALUE #( key = <trqroot>-key element_cat = sc_elem_cat-fwo )
    INTO TABLE lt_data_raw
    ASSIGNING FIELD-SYMBOL(<data_raw>).

  "fill data; (1) tree information
  <data_raw>-tbi-children_loaded = abap_true.
  <data_raw>-tbi-expanded = abap_true.
  <data_raw>-tbi-image_src = '~Icon/Order'.
  <data_raw>-tbi-text =
  /scmtms/cl_trq_helper=>return_descr_for_trq( is_trq_root = <trqroot> ).
  READ TABLE lt_kl_trqfu TRANSPORTING NO FIELDS
    WITH KEY source_key = <trqroot>-key.
  IF sy-subrc GT 0.
    <data_raw>-tbi-is_leaf = abap_true.
  ENDIF.

  "fill data; (2) root information
```

```
    <data_raw>-root-doc_key  = <trqroot>-key.
    <data_raw>-root-doc_cat  = <trqroot>-trq_cat.
    <data_raw>-root-doc_type = <trqroot>-trq_type.
    <data_raw>-root-doc_id   = <trqroot>-trq_id.

    "fill data; (3) departure information
    <data_raw>-dep-log_loc_uuid    = <trqroot>-src_loc_key.
    <data_raw>-dep-log_locid       = <trqroot>-src_loc_id.
    <data_raw>-dep-log_locun       = <trqroot>-src_loc_unlocode.
    <data_raw>-dep-log_lociata     = <trqroot>-src_loc_iatacode.
    <data_raw>-dep-plan_trans_time = <trqroot>-pic_ear_req.

    "fill data; (4) arrival information
    <data_raw>-arr-log_loc_uuid    = <trqroot>-src_loc_key.
    <data_raw>-arr-log_locid       = <trqroot>-src_loc_id.
    <data_raw>-arr-log_locun       = <trqroot>-src_loc_unlocode.
    <data_raw>-arr-log_lociata     = <trqroot>-src_loc_iatacode.
    <data_raw>-arr-plan_trans_time = <trqroot>-pic_ear_req.

    "set key mapping records;
    "data from all three groups come from the same node (TRQ root)
    set_key_map(
      EXPORTING
        iv_ui_key       = <data_raw>-key    " TBI Instance Key
        iv_ui_group     = sc_group-root     " Group name for named includes
        iv_bo_key       = /scmtms/if_trq_c=>sc_bo_key    " Business Object
        iv_node_key     = /scmtms/if_trq_c=>sc_node-root " BO Node Key
        iv_instance_key = <trqroot>-key     " Instance key
        iv_src_key      = <trqroot>-key     " Source key in IT_KEY
        iv_root_key     = <trqroot>-key ).  " Root key of the instance
    set_key_map(
      EXPORTING
        iv_ui_key       = <data_raw>-key
        iv_ui_group     = sc_group-departure
        iv_bo_key       = /scmtms/if_trq_c=>sc_bo_
        iv_node_key     = /scmtms/if_trq_c=>sc_node-root
        iv_instance_key = <trqroot>-key
        iv_src_key      = <trqroot>-key
        iv_root_key     = <trqroot>-key ).
    set_key_map(
      EXPORTING
        iv_ui_key       = <data_raw>-key
        iv_ui_group     = sc_group-arrival
        iv_bo_key       = /scmtms/if_trq_c=>sc_bo_key
        iv_node_key     = /scmtms/if_trq_c=>sc_node-root
        iv_instance_key = <trqroot>-key
        iv_src_key      = <trqroot>-key
        iv_root_key     = <trqroot>-key ).

  ENDLOOP.
```

Notice the usage of the superclass method **SET_KEY_MAP**.

Next, the freight units are added; the parent information in the hierarchy is determined by looking up the key link table between FWO and FU:

```
  LOOP AT lt_d_furoot ASSIGNING FIELD-SYMBOL(<furoot>).

    INSERT VALUE #( key = <furoot>-key element_cat = sc_elem_cat-fu )
      INTO TABLE lt_data_raw
      ASSIGNING <data_raw>.

    "locate 'parent' forwarding order
    READ TABLE lt_kl_trqfu ASSIGNING FIELD-SYMBOL(<trqfu>)
      WITH KEY target_key COMPONENTS target_key = <furoot>-key.
    CHECK sy-subrc EQ 0.

    "fill data; (1) tree information
    <data_raw>-tree_parent_key = <trqfu>-source_key.
    <data_raw>-tbi-children_loaded = abap_true.
    <data_raw>-tbi-expanded = abap_true.
    <data_raw>-tbi-image_src = '~Icon/Record'.
    <data_raw>-tbi-text =
      /scmtms/cl_tor_helper_root=>return_ident_fortor_key( is_tor_root = <furoot> ).
```

```
    READ TABLE lt_kl_fufo TRANSPORTING NO FIELDS
      WITH KEY source_key = <furoot>-key.
    IF sy-subrc GT 0.
      <data_raw>-tbi-is_leaf = abap_true.
    ENDIF.

    "fill data; (2) root information
    <data_raw>-root-doc_key  = <furoot>-key.
    <data_raw>-root-doc_cat  = <furoot>-tor_cat.
    <data_raw>-root-doc_type = <furoot>-tor_type.
    <data_raw>-root-doc_id   = <furoot>-tor_id.
    <data_raw>-root-status   = <furoot>-fixation.
    IF <data_raw>-root-status IS INITIAL.
      <data_raw>-root-status_ico = '~Icon/GreenLed'.
    ELSE.
      <data_raw>-root-status_ico = '~Icon/RedLed'.
    ENDIF.
    set_key_map(
      EXPORTING
        iv_ui_key       = <data_raw>-key
        iv_ui_group     = sc_group-root
        iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
        iv_node_key     = /scmtms/if_tor_c=>sc_node-root
        iv_instance_key = <furoot>-key
        iv_src_key      = <trqfu>-source_key
        iv_root_key     = <furoot>-key ).

    "fill data; (3) departure information
    READ TABLE lt_d_stop_f ASSIGNING FIELD-SYMBOL(<stop>)
      WITH KEY root_key COMPONENTS root_key = <furoot>-key.
    IF sy-subrc EQ 0.
      <data_raw>-dep = CORRESPONDING #( <stop> ).
      set_key_map(
        EXPORTING
          iv_ui_key       = <data_raw>-key
          iv_ui_group     = sc_group-departure
          iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
          iv_node_key     = /scmtms/if_tor_c=>sc_node-stop
          iv_instance_key = <stop>-key
          iv_src_key      = <trqfu>-source_key
          iv_root_key     = <furoot>-key ).
    ENDIF.

    "fill data; (4) arrival information
    READ TABLE lt_d_stop_l ASSIGNING <stop>
      WITH KEY root_key COMPONENTS root_key = <furoot>-key.
    IF sy-subrc EQ 0.
      <data_raw>-arr = CORRESPONDING #( <stop> ).
      set_key_map(
        EXPORTING
          iv_ui_key       = <data_raw>-key
          iv_ui_group     = sc_group-arrival
          iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
          iv_node_key     = /scmtms/if_tor_c=>sc_node-stop
          iv_instance_key = <stop>-key
          iv_src_key      = <trqfu>-source_key
          iv_root_key     = <furoot>-key ).
    ENDIF.

  ENDLOOP.
```

Notice in this case how STOP data was moved from source to target.

Very similarly, the freight document data is added; now, a double key link had to be evaluated:

```
  LOOP AT lt_d_foroot ASSIGNING FIELD-SYMBOL(<foroot>).

    INSERT VALUE #( key = <foroot>-key element_cat = sc_elem_cat-fo )
      INTO TABLE lt_data_raw
      ASSIGNING <data_raw>.

    "locate 'parent' freight unit
    READ TABLE lt_kl_fufo ASSIGNING FIELD-SYMBOL(<fufo>)
      WITH KEY target_key COMPONENTS target_key = <foroot>-key.
```

```
CHECK sy-subrc EQ 0.
"and parent TRQ
READ TABLE lt_kl_trqfu ASSIGNING <trqfu>
  WITH KEY target_key COMPONENTS target_key = <fufo>-source_key.
CHECK sy-subrc EQ 0.

"fill data; (1) tree information
<data_raw>-tree_parent_key = <fufo>-source_key.
<data_raw>-tbi-is_leaf = abap_true.
<data_raw>-tbi-children_loaded = abap_true.
<data_raw>-tbi-expanded = abap_true.
<data_raw>-tbi-text =
  /scmtms/cl_tor_helper_root=>return_ident_fortor_key( is_tor_root = <foroot> ).
IF <foroot>-tor_cat EQ /scmtms/if_tor_const=>sc_tor_cat_bo.
  IF <foroot>-booking_trmo EQ /scmtms/if_common_c=>c_trmodcat-air.
    <data_raw>-tbi-image_src = '~Icon/Airplane01'.
  ELSE.
    <data_raw>-tbi-image_src = '~Icon/Ship'.
  ENDIF.
ELSE.
  <data_raw>-tbi-image_src = '~Icon/Truck'.
ENDIF.

"fill data; (2) root information
<data_raw>-root-doc_key  = <foroot>-key.
<data_raw>-root-doc_cat  = <foroot>-tor_cat.
<data_raw>-root-doc_type = <foroot>-tor_type.
<data_raw>-root-doc_id   = <foroot>-tor_id.
<data_raw>-root-status   = <foroot>-fixation.
IF <data_raw>-root-status IS INITIAL.
  <data_raw>-root-status_ico = '~Icon/GreenLed'.
ELSE.
  <data_raw>-root-status_ico = '~Icon/RedLed'.
ENDIF.
set_key_map(
  EXPORTING
    iv_ui_key       = <data_raw>-key
    iv_ui_group     = sc_group-root
    iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
    iv_node_key     = /scmtms/if_tor_c=>sc_node-root
    iv_instance_key = <foroot>-key
    iv_src_key      = <trqfu>-source_key
    iv_root_key     = <foroot>-key ).

"fill data; (3) departure information
READ TABLE lt_d_stop_f ASSIGNING <stop>
  WITH KEY root_key COMPONENTS root_key = <foroot>-key.
IF sy-subrc EQ 0.
  <data_raw>-dep = CORRESPONDING #( <stop> ).
  set_key_map(
    EXPORTING
      iv_ui_key       = <data_raw>-key
      iv_ui_group     = sc_group-departure
      iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
      iv_node_key     = /scmtms/if_tor_c=>sc_node-stop
      iv_instance_key = <stop>-key
      iv_src_key      = <trqfu>-source_key
      iv_root_key     = <foroot>-key ).
ENDIF.

"fill data; (4) arrival information
READ TABLE lt_d_stop_l ASSIGNING <stop>
  WITH KEY root_key COMPONENTS root_key = <foroot>-key.
IF sy-subrc EQ 0.
  <data_raw>-arr = CORRESPONDING #( <stop> ).
  set_key_map(
    EXPORTING
      iv_ui_key       = <data_raw>-key
      iv_ui_group     = sc_group-arrival
      iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
      iv_node_key     = /scmtms/if_tor_c=>sc_node-stop
      iv_instance_key = <stop>-key
      iv_src_key      = <trqfu>-source_key
      iv_root_key     = <foroot>-key ).
ENDIF.
```

```
  ENDLOOP.
```

The last step is to execute conversion on the raw table:

```
*-----------------------------------------------------------------------*
* Execute conversion and store data in the buffer
*-----------------------------------------------------------------------*
  mo_conversion->map_data_bo_node_to_ui(
    EXPORTING
      it_node_data = lt_data_raw
    IMPORTING
      et_ui_data   = mt_data ).
```

That's about it.

Having defined the key mapping (along with the previous definition of field source mapping) allowed me to skip further implementation steps, as they are handled by the superclass. I don't have to take care of field properties, and I don't have to take care of the flush phase. Based on these mappings, the generic logic is able do to a thorough job.

In some circumstances, though, the generic logic may not fit completely, case in which one must redefine the affected methods of the core interface.

I. STEP 9 - IMPLEMENT ASSOCIATION RESOLVING

There is one little step I must accomplish, namely to implement some logic in the method **RESOLVE_ASSOC** of the core interface.

```
  methods /SCMTMS/IF_UI_TBI_CORE~RESOLVE_ASSOC
    redefinition .
```

In my case, it's pretty straightforward - I need to return TRQ Root and TOR Root keys, task for which the mere evaluation of the key mapping suffices. Remember, in IT_KEY there are "UI instance" keys:

```
  LOOP AT it_key ASSIGNING FIELD-SYMBOL(<key>).
    LOOP AT mt_keymap ASSIGNING FIELD-SYMBOL(<keymap>)
      WHERE ui_inst_key = <key>-key
        AND group = sc_group-root.
      "check which keys should we provide
      CASE iv_association.
        WHEN sc_assoc-trq_root.
          CHECK <keymap>-node = /scmtms/if_trq_c=>sc_node-root.
        WHEN sc_assoc-tor_root.
          CHECK <keymap>-node = /scmtms/if_tor_c=>sc_node-root.
      ENDCASE.
      IF <keymap>-inst_key IS NOT INITIAL.
        INSERT VALUE #( key = <keymap>-inst_key )
          INTO TABLE et_target_key.
        INSERT VALUE #( source_key = <key>-key
                        target_key = <keymap>-inst_key )
          INTO TABLE et_key_link.
      ENDIF.
    ENDLOOP.
  ENDLOOP.
```

I didn't even have to inquire the element category. But there might be more complicated situations, where one must read, as an intermediate step, the corresponding record in MT_DATA, to evaluate element category and/or to find out information that is not present in key mapping. In my example, it would have been the case, should I have defined an association to BO Location.

End.

V. APPENDIX

A. COMPLETE CODING OF TBI CLASS

```
class ZMDF_CL_UI_TBIDEMO definition
  public
  inheriting from /SCMTMS/CL_UI_TBI_CORE
  create public .

public section.

  constants:
    begin of SC_ELEM_CAT,
      fwo type /scmtms/if_ui_tbi_core=>ty_element_cat value 'FW',
      fu  type /scmtms/if_ui_tbi_core=>ty_element_cat value 'FU',
      fo  type /scmtms/if_ui_tbi_core=>ty_element_cat value 'FO',
    end of sc_elem_cat .
  constants:
    begin of SC_GROUP,
      root      type ddgroup value 'ROOT',
      departure type ddgroup value 'DEP',
      arrival   type ddgroup value 'ARR',
    end of sc_group .
  constants:
    begin of SC_ACTION,
      confirm   type char30 value 'TBI_CONFIRM',
      fix       type char30 value 'TBI_FIX',
      send      type char30 value 'TBI_SEND',
      uisett    type char30 value 'TBI_OPEN_UISETT',
      doc_id    type char30 value 'DOC_ID',
    end of SC_ACTION .
  constants:
    begin of SC_ASSOC,
      trq_root  type string value 'TRQ Root',
      tor_root  type string value 'TOR Root',
    end of SC_ASSOC .

  methods /SCMTMS/IF_UI_TBI_CORE~DESCRIBE
    redefinition .
  methods /SCMTMS/IF_UI_TBI_CORE~GET_DATA
    redefinition .
  methods /SCMTMS/IF_UI_TBI_FPM~ADAPT_DEFINITION
    redefinition .
  methods /SCMTMS/IF_UI_TBI_CORE~RESOLVE_ASSOC
    redefinition .
protected section.

  types:
    begin of TS_ROOT_INT .
      include type ZMDF_S_UI_TBIDEMO_ROOT.
  types:
      doc_key type /bobf/conf_key,
      status  type /scmtms/tor_root_fixation,
    end of TS_ROOT_INT .
  types:
    begin of TS_LOC_INT .
      include type ZMDF_S_UI_TBIDEMO_LOCAIR.
  types:
      log_loc_uuid    type /bobf/conf_key,
      log_locun       type /scmtms/loc_unlocode,
      plan_trans_time type /scmtms/stop_plan_date,
    end of TS_LOC_INT .
  types:
    begin of TS_DATA_INT,
      element_cat type /scmtms/if_ui_tbi_core=>ty_element_cat.
      include type /scmtms/s_ui_cmn_treectr as tbi.
      include type ts_root_int as root .
      include type ts_loc_int as dep renaming with suffix dep .
      include type ts_loc_int as arr renaming with suffix arr .
  types:
    end of TS_DATA_INT .
```

```
  types:
    TT_DATA_INT type standard table of TS_DATA_INT
                with non-unique sorted key KEY components KEY .

  data MT_DATA type TT_DATA_INT .
  data MV_MOTCAT type /SCMTMS/TRMODCAT .

  methods GET_DATA_INT
    importing
      !IT_KEY type /BOBF/T_FRW_KEY .

  methods _INIT_INTERNAL
    redefinition .
private section.

  methods DET_CHANGE_MODE
    importing
      !IT_KEY type /BOBF/T_FRW_KEY
    returning
      value(RV_MODE) type /BOBF/CONF_EDIT_MODE .
ENDCLASS.



CLASS ZMDF_CL_UI_TBIDEMO IMPLEMENTATION.


* <SIGNATURE>-----------------------------------------------------------------------------------+
* | Instance Public Method ZMDF_CL_UI_TBIDEMO->/SCMTMS/IF_UI_TBI_CORE~DESCRIBE
* +----------------------------------------------------------------------------------------------+
* | [<---] ES_OBJECT_KEY                  TYPE        TS_OBJECT
* | [<---] ES_FIELDNAME                   TYPE        TS_FIELDNAME
* | [<---] EO_FIELD_DESCRIPTION           TYPE REF TO CL_ABAP_STRUCTDESCR
* | [<---] ET_TABLE_SECKEY                TYPE        ABAP_TABLE_KEYDESCR_TAB
* | [<---] ET_FIELD_PROPERTY              TYPE        TT_FIELDPROP
* | [<---] ET_ASSOCIATION                 TYPE        TT_ASSOC
* +----------------------------------------------------------------------------------------</SIGNATURE>
METHOD /scmtms/if_ui_tbi_core~describe.

  super->/scmtms/if_ui_tbi_core~describe(
    IMPORTING
      es_object_key        = es_object_key
      es_fieldname         = es_fieldname
      eo_field_description = eo_field_description
      et_table_seckey      = et_table_seckey
      et_field_property    = et_field_property
      et_association       = et_association ).

  READ TABLE mt_parameter ASSIGNING FIELD-SYMBOL(<param>)
    WITH KEY name = 'MOTCAT'.
  IF sy-subrc EQ 0 AND
     <param>-value EQ /scmtms/if_common_c=>c_tr_mode_category-air.
    eo_field_description ?=
      cl_abap_structdescr=>describe_by_name( 'ZMDF_S_UI_TBIDEMO_AIR').
  ELSE.
    eo_field_description ?=
      cl_abap_structdescr=>describe_by_name( 'ZMDF_S_UI_TBIDEMO_SEA').
  ENDIF.

  et_association = VALUE #(
    ( name = sc_assoc-trq_root
      bo   = /scmtms/if_trq_c=>sc_bo_key
      node = /scmtms/if_trq_c=>sc_node-root
      desc = 'TRQ Root'(001)
      )
    ( name = sc_assoc-tor_root
      bo   = /scmtms/if_tor_c=>sc_bo_key
      node = /scmtms/if_tor_c=>sc_node-root
      desc = 'TOR Root'(002)
      )
  ).

ENDMETHOD.
```

```
* <SIGNATURE>-------------------------------------------------------------------------------------+
* | Instance Public Method ZMDF_CL_UI_TBIDEMO->/SCMTMS/IF_UI_TBI_CORE~GET_DATA
* +-------------------------------------------------------------------------------------------------+
* | [--->] IT_KEY                          TYPE        /BOBF/T_FRW_KEY
* | [--->] IT_REQ_ATTR                     TYPE        /SCMTMS/T_STRING(optional)
* | [--->] IT_PARAMETER                    TYPE        TT_PARAMETER(optional)
* | [--->] IS_OBJECT_KEY                   TYPE        TS_OBJECT(optional)
* | [--->] IR_ADDTN_DATA                   TYPE REF TO DATA(optional)
* | [<---] ET_DATA                         TYPE        INDEX TABLE
* | [<---] EV_CHANGED                      TYPE        BOOLE_D
* +-------------------------------------------------------------------------------------</SIGNATURE>
METHOD /scmtms/if_ui_tbi_core~get_data.
  CLEAR:
    et_data,
    ev_changed.

  IF mt_data IS INITIAL.
    ev_changed = abap_true.
    get_data_int(
      EXPORTING
        it_key = it_key ).
  ENDIF.

  MOVE-CORRESPONDING mt_data TO et_data.
ENDMETHOD.


* <SIGNATURE>-------------------------------------------------------------------------------------+
* | Instance Public Method ZMDF_CL_UI_TBIDEMO->/SCMTMS/IF_UI_TBI_CORE~RESOLVE_ASSOC
* +-------------------------------------------------------------------------------------------------+
* | [--->] IV_ASSOCIATION                  TYPE        /BOFU/FBI_ASSOCIATION
* | [--->] IT_KEY                          TYPE        /BOBF/T_FRW_KEY
* | [<---] ET_KEY_LINK                     TYPE        /BOBF/T_FRW_KEY_LINK
* | [<---] ET_TARGET_KEY                   TYPE        /BOBF/T_FRW_KEY
* +-------------------------------------------------------------------------------------</SIGNATURE>
METHOD /scmtms/if_ui_tbi_core~resolve_assoc.

  CLEAR:
    et_key_link,
    et_target_key.

  LOOP AT it_key ASSIGNING FIELD-SYMBOL(<key>).
    LOOP AT mt_keymap ASSIGNING FIELD-SYMBOL(<keymap>)
      WHERE ui_inst_key = <key>-key
        AND group = sc_group-root.
      "check which keys should we provide
      CASE iv_association.
        WHEN sc_assoc-trq_root.
          CHECK <keymap>-node = /scmtms/if_trq_c=>sc_node-root.
        WHEN sc_assoc-tor_root.
          CHECK <keymap>-node = /scmtms/if_tor_c=>sc_node-root.
      ENDCASE.
      IF <keymap>-inst_key IS NOT INITIAL.
        INSERT VALUE #( key = <keymap>-inst_key )
          INTO TABLE et_target_key.
        INSERT VALUE #( source_key = <key>-key
                        target_key = <keymap>-inst_key )
          INTO TABLE et_key_link.
      ENDIF.
    ENDLOOP.
  ENDLOOP.

ENDMETHOD.


* <SIGNATURE>-------------------------------------------------------------------------------------+
* | Instance Public Method ZMDF_CL_UI_TBIDEMO->/SCMTMS/IF_UI_TBI_FPM~ADAPT_DEFINITION
* +-------------------------------------------------------------------------------------------------+
* | [--->] IS_UIBB_KEY                     TYPE        CL_FPM_EVENT=>TY_S_UIBB_KEY
* | [<-->] CT_FIELD_DESCR_FORM             TYPE        FPMGB_T_FORMFIELD_DESCR(optional)
* | [<-->] CT_FIELD_DESCR_LIST             TYPE        FPMGB_T_LISTFIELD_DESCR(optional)
* | [<-->] CT_FIELD_DESCR_TREE             TYPE        FPMGB_T_TREEFIELD_DESCR(optional)
* | [<-->] CT_ACTION_DEFINITION            TYPE        FPMGB_T_ACTIONDEF(optional)
* | [<-->] CT_DND_DEFINITION               TYPE        FPMGB_T_DND_DEFINITION(optional)
* | [<-->] CT_ROW_ACTIONS                  TYPE        FPMGB_T_ROW_ACTION(optional)
```

```
* | [<-->] CS_OPTIONS_FORM                TYPE        FPMGB_S_FORM_OPTIONS(optional)
* | [<-->] CS_OPTIONS_LIST                TYPE        FPMGB_S_LIST_OPTIONS(optional)
* | [<-->] CS_OPTIONS_TREE                TYPE        FPMGB_S_TREE_OPTIONS(optional)
* +--------------------------------------------------------------------------------------</SIGNATURE>
METHOD /scmtms/if_ui_tbi_fpm~adapt_definition.

  super->/scmtms/if_ui_tbi_fpm~adapt_definition(
    EXPORTING
      is_uibb_key          = is_uibb_key
    CHANGING
      ct_field_descr_form  = ct_field_descr_form
      ct_field_descr_list  = ct_field_descr_list
      ct_field_descr_tree  = ct_field_descr_tree
      ct_action_definition = ct_action_definition
      ct_dnd_definition    = ct_dnd_definition
      ct_row_actions       = ct_row_actions
      cs_options_form      = cs_options_form
      cs_options_list      = cs_options_list
      cs_options_tree      = cs_options_tree
         ).

  DATA(lt_field) = VALUE /scmtms/cl_ui_tbi_util=>tt_fdescr(
    ( f = 'LOG_LOCIDDEP'          tx = '/SCMTMS/UI_CMN/SOURCE_LOCATION' )
    ( f = 'LOG_LOCIDARR'          tx = '/SCMTMS/UI_CMN/DESTINATION_LOCATION' )
    ( f = 'LOG_LOCUNDEP'          tx = '/SCMTMS/UI_CMN/UNLOCODE_SOURCE' )
    ( f = 'LOG_LOCUNARR'          tx = '/SCMTMS/UI_CMN/UNLOCODE_DESTINATION' )
    ( f = 'LOG_LOCIATADEP'        tx = '/SCMTMS/UI_CMN/IATA_SOURCE' )
    ( f = 'LOG_LOCIATAARR'        tx = '/SCMTMS/UI_CMN/IATA_DESTINATION' )
    ( f = 'LOG_LOCDESCRDEP'       tx = '/SCMTMS/UI_CMN/SOURCE_LOC_ADDRESS' )
    ( f = 'LOG_LOCDESCRARR'       tx = '/SCMTMS/UI_CMN/DESTIN_LOC_ADDRESS' )
    ( f = 'PLAN_TRANS_TIME_DDEP'  tx = '/SCMTMS/UI_CMN/PLAN_DEP_DATE' )
    ( f = 'PLAN_TRANS_TIME_TDEP'  tx = '/SCMTMS/UI_CMN/PLAN_DEP_TIME' )
    ( f = 'PLAN_TRANS_TIME_TZDEP' tx = '/SCMTMS/UI_CMN/PLAN_DEP_TZONE' )
    ( f = 'PLAN_TRANS_TIME_DARR'  tx = '/SCMTMS/UI_CMN/PLANNED_ARRIVAL_DATE' )
    ( f = 'PLAN_TRANS_TIME_TARR'  tx = '/SCMTMS/UI_CMN/PLANNED_ARRIVAL_TIME' )
    ( f = 'PLAN_TRANS_TIME_TZARR' tx = '/SCMTMS/UI_CMN/PLANNED_ARRIVAL_TZONE' )
    ( f = 'STATUS_ICO' tx = '/SCMTMS/UI_CMN/STAT' tr = 'STATUS_TXT') "status icon
  ).
  /scmtms/cl_ui_tbi_util=>chg_fcat_t(
    EXPORTING
      it_text  = lt_field
    CHANGING
      ct_descr = ct_field_descr_tree ).

  DATA(lt_action) = VALUE /scmtms/cl_ui_tbi_util=>tt_adescr(
    ( id = sc_action-confirm  tx = '/SCMTMS/UI_CMN/CONFIRM_DOCUMENT' )
    ( id = sc_action-fix      tx = '/SCMTMS/UI_CMN/FIX_DOCUMENT' )
    ( id = sc_action-send     tx = '/SCMTMS/UI_CMN/TOR_SEND_CARRIER' )
    ( id = sc_action-uisett   tx = '/SCMTMS/UI_CMN/UISETT' )
  ).
  /scmtms/cl_ui_tbi_util=>chg_action(
    EXPORTING
      it_action_descr      = lt_action
    CHANGING
      ct_action_definition = ct_action_definition ).

ENDMETHOD.


* <SIGNATURE>----------------------------------------------------------------------------------+
* | Instance Private Method ZMDF_CL_UI_TBIDEMO->DET_CHANGE_MODE
* +--------------------------------------------------------------------------------------------+
* | [--->] IT_KEY                         TYPE        /BOBF/T_FRW_KEY
* | [<-()] RV_MODE                        TYPE        /BOBF/CONF_EDIT_MODE
* +--------------------------------------------------------------------------------------</SIGNATURE>
METHOD det_change_mode.

  rv_mode = /bobf/if_conf_c=>sc_edit_read_only.

  IF it_key IS INITIAL.
    RETURN.
  ENDIF.

  mo_data_interface->retrieve_properties(
    EXPORTING
```

```
      iv_bo_key        = /scmtms/if_trq_c=>sc_bo_key
      iv_node_key      = /scmtms/if_trq_c=>sc_node-root
      it_key           = it_key
      iv_node_property = abap_true
    IMPORTING
      et_node_property = DATA(lt_prop) ).
  READ TABLE lt_prop ASSIGNING FIELD-SYMBOL(<prop>)
    WITH TABLE KEY key = it_key[ 1 ]-key.
  IF sy-subrc EQ 0 AND
     <prop>-update_enabled EQ abap_true.
    rv_mode = /bobf/if_conf_c=>sc_edit_optimistic.
  ENDIF.

endmethod.


* <SIGNATURE>-----------------------------------------------------------------------------------------+
* | Instance Protected Method ZMDF_CL_UI_TBIDEMO->GET_DATA_INT                                          |
* +-----------------------------------------------------------------------------------------------------+
* | [--->] IT_KEY                          TYPE        /BOBF/T_FRW_KEY                                  |
* +-----------------------------------------------------------------------------------------</SIGNATURE>
METHOD get_data_int.

  DATA:
    lt_data_raw  TYPE tt_data_int,
    lt_d_trqroot TYPE /scmtms/t_trq_root_k,
    lt_d_furoot  TYPE /scmtms/t_tor_root_k,
    lt_d_foroot  TYPE /scmtms/t_tor_root_k,
    lt_d_stop_f  TYPE /scmtms/t_tor_stop_k,
    lt_d_stop_l  TYPE /scmtms/t_tor_stop_k.

* Collect Data
  mo_data_interface->retrieve(
    EXPORTING
      iv_bo_key       = /scmtms/if_trq_c=>sc_bo_key
      iv_node_key     = /scmtms/if_trq_c=>sc_node-root
      it_key          = it_key
      iv_fill_data    = abap_true
    IMPORTING
      et_data         = lt_d_trqroot ).

  mo_data_interface->retrieve_by_association(
    EXPORTING
      iv_bo_key       = /scmtms/if_trq_c=>sc_bo_key
      iv_node_key     = /scmtms/if_trq_c=>sc_node-root
      iv_association  = /scmtms/if_trq_c=>sc_association-root-tor_root_fu
      it_key          = it_key
      iv_fill_data    = abap_true
    IMPORTING
      et_data         = lt_d_furoot
      et_target_key   = DATA(lt_k_furoot)
      et_key_link     = DATA(lt_kl_trqfu) ).

  mo_data_interface->retrieve_by_association(
    EXPORTING
      iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
      iv_node_key     = /scmtms/if_tor_c=>sc_node-root
      iv_association  = /scmtms/if_tor_c=>sc_association-root-capa_tor
      it_key          = lt_k_furoot
      iv_fill_data    = abap_true
    IMPORTING
      et_data         = lt_d_foroot
      et_target_key   = DATA(lt_k_foroot)
      et_key_link     = DATA(lt_kl_fufo) ).

  INSERT LINES OF lt_k_foroot INTO TABLE lt_k_furoot.

  mo_data_interface->retrieve_by_association(
    EXPORTING
      iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
      iv_node_key     = /scmtms/if_tor_c=>sc_node-root
      iv_association  = /scmtms/if_tor_c=>sc_association-root-stop_first_direct
      it_key          = lt_k_furoot
      iv_fill_data    = abap_true
      iv_edit_mode    = det_change_mode( it_key )
```

```
    IMPORTING
      et_data         = lt_d_stop_f ).

  mo_data_interface->retrieve_by_association(
    EXPORTING
      iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
      iv_node_key     = /scmtms/if_tor_c=>sc_node-root
      iv_association  = /scmtms/if_tor_c=>sc_association-root-stop_last_direct
      it_key          = lt_k_furoot
      iv_fill_data    = abap_true
      iv_edit_mode    = det_change_mode( it_key )
    IMPORTING
      et_data         = lt_d_stop_l ).

*-----------------------------------------------------------------------*
* Add FWO level to the raw table
*-----------------------------------------------------------------------*
  LOOP AT lt_d_trqroot ASSIGNING FIELD-SYMBOL(<trqroot>).

    INSERT VALUE #( key = <trqroot>-key element_cat = sc_elem_cat-fwo )
      INTO TABLE lt_data_raw
      ASSIGNING FIELD-SYMBOL(<data_raw>).

    "fill data; (1) tree information
    <data_raw>-tbi-children_loaded = abap_true.
    <data_raw>-tbi-expanded = abap_true.
    <data_raw>-tbi-image_src = '~Icon/Order'.
    <data_raw>-tbi-text = /scmtms/cl_trq_helper=>return_descr_for_trq( is_trq_root = <trqroot> ).
    READ TABLE lt_kl_trqfu TRANSPORTING NO FIELDS
      WITH KEY source_key = <trqroot>-key.
    IF sy-subrc GT 0.
      <data_raw>-tbi-is_leaf = abap_true.
    ENDIF.

    "fill data; (2) root information
    <data_raw>-root-doc_key  = <trqroot>-key.
    <data_raw>-root-doc_cat  = <trqroot>-trq_cat.
    <data_raw>-root-doc_type = <trqroot>-trq_type.
    <data_raw>-root-doc_id   = <trqroot>-trq_id.

    "fill data; (3) departure information
    <data_raw>-dep-log_loc_uuid    = <trqroot>-src_loc_key.
    <data_raw>-dep-log_locid       = <trqroot>-src_loc_id.
    <data_raw>-dep-log_locun       = <trqroot>-src_loc_unlocode.
    <data_raw>-dep-log_lociata     = <trqroot>-src_loc_iatacode.
    <data_raw>-dep-plan_trans_time = <trqroot>-pic_ear_req.

    "fill data; (4) arrival information
    <data_raw>-arr-log_loc_uuid    = <trqroot>-src_loc_key.
    <data_raw>-arr-log_locid       = <trqroot>-src_loc_id.
    <data_raw>-arr-log_locun       = <trqroot>-src_loc_unlocode.
    <data_raw>-arr-log_lociata     = <trqroot>-src_loc_iatacode.
    <data_raw>-arr-plan_trans_time = <trqroot>-pic_ear_req.

    "set key mapping records;
    "data from all three groups come from the same node (TRQ root)
    set_key_map(
      EXPORTING
        iv_ui_key       = <data_raw>-key    " TBI Instance Key
        iv_ui_group     = sc_group-root     " Group name for named includes
        iv_bo_key       = /scmtms/if_trq_c=>sc_bo_key    " Business Object
        iv_node_key     = /scmtms/if_trq_c=>sc_node-root " BO Node Key
        iv_instance_key = <trqroot>-key     " Instance key
        iv_src_key      = <trqroot>-key     " Source key in IT_KEY triggering this record
        iv_root_key     = <trqroot>-key ).  " Root key of the instance
    set_key_map(
      EXPORTING
        iv_ui_key       = <data_raw>-key    " TBI Instance Key
        iv_ui_group     = sc_group-departure" Group name for named includes
        iv_bo_key       = /scmtms/if_trq_c=>sc_bo_key    " Business Object
        iv_node_key     = /scmtms/if_trq_c=>sc_node-root " BO Node Key
        iv_instance_key = <trqroot>-key     " Instance key
        iv_src_key      = <trqroot>-key     " Source key in IT_KEY triggering this record
        iv_root_key     = <trqroot>-key ).  " Root key of the instance
    set_key_map(
```

```
      EXPORTING
        iv_ui_key       = <data_raw>-key    " TBI Instance Key
        iv_ui_group     = sc_group-arrival  " Group name for named includes
        iv_bo_key       = /scmtms/if_trq_c=>sc_bo_key    " Business Object
        iv_node_key     = /scmtms/if_trq_c=>sc_node-root " BO Node Key
        iv_instance_key = <trqroot>-key     " Instance key
        iv_src_key      = <trqroot>-key     " Source key in IT_KEY triggering this record
        iv_root_key     = <trqroot>-key ).  " Root key of the instance

  ENDLOOP.

*-----------------------------------------------------------------------*
* Add FU level to the raw table
*-----------------------------------------------------------------------*
  LOOP AT lt_d_furoot ASSIGNING FIELD-SYMBOL(<furoot>).

    INSERT VALUE #( key = <furoot>-key element_cat = sc_elem_cat-fu )
      INTO TABLE lt_data_raw
      ASSIGNING <data_raw>.

    "locate 'parent' forwarding order
    READ TABLE lt_kl_trqfu ASSIGNING FIELD-SYMBOL(<trqfu>)
      WITH KEY target_key COMPONENTS target_key = <furoot>-key.
    CHECK sy-subrc EQ 0.

    "fill data; (1) tree information
    <data_raw>-tree_parent_key = <trqfu>-source_key.
    <data_raw>-tbi-children_loaded = abap_true.
    <data_raw>-tbi-expanded = abap_true.
    <data_raw>-tbi-image_src = '~Icon/Record'.
    <data_raw>-tbi-text =
      /scmtms/cl_tor_helper_root=>return_ident_fortor_key( is_tor_root = <furoot> ).
    READ TABLE lt_kl_fufo TRANSPORTING NO FIELDS
      WITH KEY source_key = <furoot>-key.
    IF sy-subrc GT 0.
      <data_raw>-tbi-is_leaf = abap_true.
    ENDIF.

    "fill data; (2) root information
    <data_raw>-root-doc_key  = <furoot>-key.
    <data_raw>-root-doc_cat  = <furoot>-tor_cat.
    <data_raw>-root-doc_type = <furoot>-tor_type.
    <data_raw>-root-doc_id   = <furoot>-tor_id.
    <data_raw>-root-status   = <furoot>-fixation.
    IF <data_raw>-root-status IS INITIAL.
      <data_raw>-root-status_ico = '~Icon/GreenLed'.
    ELSE.
      <data_raw>-root-status_ico = '~Icon/RedLed'.
    ENDIF.
    set_key_map(
      EXPORTING
        iv_ui_key       = <data_raw>-key
        iv_ui_group     = sc_group-root
        iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
        iv_node_key     = /scmtms/if_tor_c=>sc_node-root
        iv_instance_key = <furoot>-key
        iv_src_key      = <trqfu>-source_key
        iv_root_key     = <furoot>-key ).

    "fill data; (3) departure information
    READ TABLE lt_d_stop_f ASSIGNING FIELD-SYMBOL(<stop>)
      WITH KEY root_key COMPONENTS root_key = <furoot>-key.
    IF sy-subrc EQ 0.
      <data_raw>-dep = CORRESPONDING #( <stop> ).
      set_key_map(
        EXPORTING
          iv_ui_key       = <data_raw>-key
          iv_ui_group     = sc_group-departure
          iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
          iv_node_key     = /scmtms/if_tor_c=>sc_node-stop
          iv_instance_key = <stop>-key
          iv_src_key      = <trqfu>-source_key
          iv_root_key     = <furoot>-key ).
    ENDIF.
```

```
    "fill data; (4) arrival information
    READ TABLE lt_d_stop_l ASSIGNING <stop>
      WITH KEY root_key COMPONENTS root_key = <furoot>-key.
    IF sy-subrc EQ 0.
      <data_raw>-arr = CORRESPONDING #( <stop> ).
      set_key_map(
        EXPORTING
          iv_ui_key       = <data_raw>-key
          iv_ui_group     = sc_group-arrival
          iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
          iv_node_key     = /scmtms/if_tor_c=>sc_node-stop
          iv_instance_key = <stop>-key
          iv_src_key      = <trqfu>-source_key
          iv_root_key     = <furoot>-key ).
    ENDIF.

  ENDLOOP.

*-----------------------------------------------------------------------*
* Add FO level to the raw table
*-----------------------------------------------------------------------*
  LOOP AT lt_d_foroot ASSIGNING FIELD-SYMBOL(<foroot>).

    INSERT VALUE #( key = <foroot>-key element_cat = sc_elem_cat-fo )
      INTO TABLE lt_data_raw
      ASSIGNING <data_raw>.

    "locate 'parent' freight unit
    READ TABLE lt_kl_fufo ASSIGNING FIELD-SYMBOL(<fufo>)
      WITH KEY target_key COMPONENTS target_key = <foroot>-key.
    CHECK sy-subrc EQ 0.
    "and parent TRQ
    READ TABLE lt_kl_trqfu ASSIGNING <trqfu>
      WITH KEY target_key COMPONENTS target_key = <fufo>-source_key.
    CHECK sy-subrc EQ 0.

    "fill data; (1) tree information
    <data_raw>-tree_parent_key = <fufo>-source_key.
    <data_raw>-tbi-is_leaf = abap_true.
    <data_raw>-tbi-children_loaded = abap_true.
    <data_raw>-tbi-expanded = abap_true.
    <data_raw>-tbi-text =
      /scmtms/cl_tor_helper_root=>return_ident_fortor_key( is_tor_root = <foroot> ).
    IF <foroot>-tor_cat EQ /scmtms/if_tor_const=>sc_tor_cat_bo.
      IF <foroot>-booking_trmo EQ /scmtms/if_common_c=>c_trmodcat-air.
        <data_raw>-tbi-image_src = '~Icon/Airplane01'.
      ELSE.
        <data_raw>-tbi-image_src = '~Icon/Ship'.
      ENDIF.
    ELSE.
      <data_raw>-tbi-image_src = '~Icon/Truck'.
    ENDIF.

    "fill data; (2) root information
    <data_raw>-root-doc_key  = <foroot>-key.
    <data_raw>-root-doc_cat  = <foroot>-tor_cat.
    <data_raw>-root-doc_type = <foroot>-tor_type.
    <data_raw>-root-doc_id   = <foroot>-tor_id.
    <data_raw>-root-status   = <foroot>-fixation.
    IF <data_raw>-root-status IS INITIAL.
      <data_raw>-root-status_ico = '~Icon/GreenLed'.
    ELSE.
      <data_raw>-root-status_ico = '~Icon/RedLed'.
    ENDIF.
    set_key_map(
      EXPORTING
        iv_ui_key       = <data_raw>-key
        iv_ui_group     = sc_group-root
        iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
        iv_node_key     = /scmtms/if_tor_c=>sc_node-root
        iv_instance_key = <foroot>-key
        iv_src_key      = <trqfu>-source_key
        iv_root_key     = <foroot>-key ).

    "fill data; (3) departure information
```

```
    READ TABLE lt_d_stop_f ASSIGNING <stop>
      WITH KEY root_key COMPONENTS root_key = <foroot>-key.
    IF sy-subrc EQ 0.
      <data_raw>-dep = CORRESPONDING #( <stop> ).
      set_key_map(
        EXPORTING
          iv_ui_key       = <data_raw>-key
          iv_ui_group     = sc_group-departure
          iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
          iv_node_key     = /scmtms/if_tor_c=>sc_node-stop
          iv_instance_key = <stop>-key
          iv_src_key      = <trqfu>-source_key
          iv_root_key     = <foroot>-key ).
    ENDIF.

    "fill data; (4) arrival information
    READ TABLE lt_d_stop_l ASSIGNING <stop>
      WITH KEY root_key COMPONENTS root_key = <foroot>-key.
    IF sy-subrc EQ 0.
      <data_raw>-arr = CORRESPONDING #( <stop> ).
      set_key_map(
        EXPORTING
          iv_ui_key       = <data_raw>-key
          iv_ui_group     = sc_group-arrival
          iv_bo_key       = /scmtms/if_tor_c=>sc_bo_key
          iv_node_key     = /scmtms/if_tor_c=>sc_node-stop
          iv_instance_key = <stop>-key
          iv_src_key      = <trqfu>-source_key
          iv_root_key     = <foroot>-key ).
    ENDIF.

  ENDLOOP.

*--------------------------------------------------------------------*
* Execute conversion and store data in the buffer
*--------------------------------------------------------------------*
  mo_conversion->map_data_bo_node_to_ui(
    EXPORTING
      it_node_data = lt_data_raw
    IMPORTING
      et_ui_data   = mt_data ).

ENDMETHOD.


* <SIGNATURE>-----------------------------------------------------------------------------------------+
* | Instance Protected Method ZMDF_CL_UI_TBIDEMO->_INIT_INTERNAL
* +-----------------------------------------------------------------------------------------------------+
* | [--->] IT_PARAMETER                   TYPE        TT_PARAMETER(optional)
* | [<---] ER_DATA                        TYPE REF TO DATA
* | [<---] EV_CONVCLASS                   TYPE        SEOCLSNAME
* | [<---] ES_OBJECT                      TYPE        /SCMTMS/IF_UI_TBI_CORE=>TS_OBJECT
* | [<---] ET_ACTION                      TYPE        /SCMTMS/IF_UI_TBI_CORE=>TT_ACTION
* | [<---] ET_NOTIFNODE                   TYPE        /SCMTMS/IF_UI_TBI_CORE=>TT_NOTIFICATION_NODE
* | [<---] ET_ELEMCAT                     TYPE        /SCMTMS/CL_UI_TBI_UTIL=>TT_ELEMCAT
* | [<---] ET_FIELDSOURCE                 TYPE        /SCMTMS/CL_UI_TBI_UTIL=>TT_FIELDSOURCEMAP
* +-----------------------------------------------------------------------------------------------</SIGNATURE>
METHOD _init_internal.

  CLEAR:
    er_data,
    ev_convclass,
    es_object,
    et_action,
    et_elemcat,
    et_fieldsource,
    et_notifnode.

  super->_init_internal(
    EXPORTING
      it_parameter   = it_parameter
    IMPORTING
      er_data        = er_data
      ev_convclass   = ev_convclass
      es_object      = es_object
```

```
    et_action      = et_action
    et_notifnode   = et_notifnode
    et_elemcat     = et_elemcat
    et_fieldsource = et_fieldsource ).

GET REFERENCE OF mt_data INTO er_data.
es_object-bo   = /scmtms/if_trq_c=>sc_bo_key.
es_object-node = /scmtms/if_trq_c=>sc_node-root.
ev_convclass   = 'ZMDF_CL_UI_TBIDEMO_CONV'.

et_elemcat = VALUE #(
  ( sc_elem_cat-fwo )
  ( sc_elem_cat-fu )
  ( sc_elem_cat-fo )
).

DATA lr_par_confirm TYPE REF TO /scmtms/s_trq_a_confirm.
CREATE DATA lr_par_confirm.
lr_par_confirm->no_check = abap_true.

et_action = VALUE #(
  ( ui_action    = sc_action-confirm
    element_cat  = sc_elem_cat-fwo
    group        = sc_group-root
    bo_key       = /scmtms/if_trq_c=>sc_bo_key
    bo_action    = /scmtms/if_trq_c=>sc_action-root-confirm
    action_param = lr_par_confirm
    )
  ( ui_action    = sc_action-fix
    element_cat  = sc_elem_cat-fu
    group        = sc_group-root
    bo_key       = /scmtms/if_tor_c=>sc_bo_key
    bo_action    = /scmtms/if_tor_c=>sc_action-root-fix_tor
    )
  ( ui_action    = sc_action-fix
    element_cat  = sc_elem_cat-fo
    group        = sc_group-root
    bo_key       = /scmtms/if_tor_c=>sc_bo_key
    bo_action    = /scmtms/if_tor_c=>sc_action-root-fix_tor
    )
  ( ui_action    = sc_action-send
    element_cat  = sc_elem_cat-fo
    group        = sc_group-root
    bo_key       = /scmtms/if_tor_c=>sc_bo_key
    bo_action    = /scmtms/if_tor_c=>sc_action-root-send_tor
    )
 ) .

"define TOR_ID as navigation column
DATA lr_par_navigation TYPE REF TO tt_parameter.
CREATE DATA lr_par_navigation.
lr_par_navigation->* = VALUE #(
  ( name  = 'CHANGE_MODE'
    value = space
    )
  ( name  = 'KEY'
    value = '(DOC_KEY)'
    )
  ( name  = 'CATEGORY'
    value = '(DOC_CAT)'
    )
).
INSERT VALUE #(
  ui_action    = sc_action-doc_id
  element_cat  = sc_elem_cat-fwo
  group        = sc_group-root
  bo_key       = /scmtms/if_trq_c=>sc_bo_key
  navigation   = abap_true
  action_param = lr_par_navigation
  ) INTO TABLE et_action.
INSERT VALUE #(
  ui_action    = sc_action-doc_id
  element_cat  = sc_elem_cat-fu
  group        = sc_group-root
  bo_key       = /scmtms/if_tor_c=>sc_bo_key
```

```
    navigation   = abap_true
    action_param = lr_par_navigation
    ) INTO TABLE et_action.
   INSERT VALUE #(
    ui_action    = sc_action-doc_id
    element_cat  = sc_elem_cat-fo
    group        = sc_group-root
    bo_key       = /scmtms/if_tor_c=>sc_bo_key
    navigation   = abap_true
    action_param = lr_par_navigation
    ) INTO TABLE et_action.


  "Register Notifications
  et_notifnode = VALUE #(
    ( node_key = /scmtms/if_tor_c=>sc_node-root
      create_is_relevant = abap_true )
    ( node_key = /scmtms/if_tor_c=>sc_node-stop
      create_is_relevant = abap_true )
  ).

  "define field source mapping
  et_fieldsource = VALUE #(
    ( group    = sc_group-departure
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOC_UUID'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-src_loc_key
      )
    ( group    = sc_group-departure
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOCID'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-src_loc_id
      )
    ( group    = sc_group-departure
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOCIATA'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-src_loc_iatacode
      )
    ( group    = sc_group-departure
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOCUN'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-src_loc_unlocode
      )
    ( group    = sc_group-departure
      elem     = sc_elem_cat-fwo
      ddname   = 'PLAN_TRANS_TIME'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-pic_ear_req
      )
    ( group    = sc_group-arrival
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOC_UUID'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-des_loc_key
      )
    ( group    = sc_group-arrival
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOCID'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-des_loc_id
      )
    ( group    = sc_group-arrival
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOCIATA'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-des_loc_iatacode
      )
    ( group    = sc_group-arrival
      elem     = sc_elem_cat-fwo
      ddname   = 'LOG_LOCUN'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-des_loc_unlocode
      )
    ( group    = sc_group-arrival
      elem     = sc_elem_cat-fwo
      ddname   = 'PLAN_TRANS_TIME'
      attr     = /scmtms/if_trq_c=>sc_node_attribute-root-del_lat_req
      )
   ).
```

```
ENDMETHOD.
ENDCLASS.
```

B. COMPLETE CODING OF TBI CONVERSION CLASS

```
class ZMDF_CL_UI_TBIDEMO_CONV definition
  public
  inheriting from /SCMTMS/CL_UI_TBI_CONVERSION
  create public .

public section.
protected section.

  methods ADAPT_MAPTABLE_SUBSTR
    redefinition .
private section.
ENDCLASS.



CLASS ZMDF_CL_UI_TBIDEMO_CONV IMPLEMENTATION.


* <SIGNATURE>---------------------------------------------------------------------------------------+
* | Instance Protected Method ZMDF_CL_UI_TBIDEMO_CONV->ADAPT_MAPTABLE_SUBSTR
* +--------------------------------------------------------------------------------------------------+
* | [--->] IO_STRUCTDESCR                 TYPE REF TO CL_ABAP_STRUCTDESCR
* | [--->] IV_GROUP                       TYPE        STRING
* | [--->] IV_SUFFIX                      TYPE        STRING
* | [<-->] CT_MAP_DATA                    TYPE        TT_MAP_DATA_EXT
* +--------------------------------------------------------------------------------------</SIGNATURE>
METHOD adapt_maptable_substr.

  CASE iv_group.

    WHEN zmdf_cl_ui_tbidemo=>sc_group-root.
      "normally, one would have created a conversion doc_key <-> doc_id,
      "but since this field isn't editable on the screen, we can skip this

    WHEN zmdf_cl_ui_tbidemo=>sc_group-departure
      OR zmdf_cl_ui_tbidemo=>sc_group-arrival.

      "conversion rule for location
      INSERT VALUE #(
        conv_cat    = sc_conv_cat-uuid
        node_field1 = 'LOG_LOC_UUID'
        ui_field1   = 'LOG_LOCID'
        ui_field2   = 'LOG_LOCUN'
        ui_field4   = 'LOG_LOCIATA'
        conf_1      = /scmtms/if_location_c=>sc_bo_key
        conf_2      = /scmtms/if_location_c=>sc_node-root
        conf_3      = /scmtms/if_location_c=>sc_alternative_key-root-location_id
        msgid       = '/SCMTMS/MSG'
        msgno       = '003' ) INTO TABLE ct_map_data.

      "conversion rule for location address
      INSERT VALUE #(
        conv_cat    = sc_conv_cat-address
        node_field1 = 'LOG_LOC_UUID'
        ui_field1   = 'LOG_LOCDESCR'
        conf_1      = /scmtms/if_location_c=>sc_bo_key ) INTO TABLE ct_map_data.

      "conversion rule for the transportation time
      READ TABLE ct_map_data ASSIGNING FIELD-SYMBOL(<map_data>)
        WITH KEY conv_cat    = sc_conv_cat-timestamp
                 node_field1 = 'PLAN_TRANS_TIME'.
      IF sy-subrc EQ 0.
        <map_data>-conf_1 = 'LOG_LOC_UÙID'.
      ELSE.
        INSERT VALUE #(
          conv_cat    = sc_conv_cat-timestamp
```

```
          node_field1 = 'PLAN_TRANS_TIME'
          ui_field1   = 'PLAN_TRANS_TIME_D'
          ui_field2   = 'PLAN_TRANS_TIME_T'
          ui_field3   = 'PLAN_TRANS_TIME_TZ'
          conf_1      = 'LOG_LOC_UUID' ) INTO TABLE ct_map_data.
      ENDIF.

    WHEN others.
  ENDCASE.

ENDMETHOD.
ENDCLASS.
```