How to Enhance SAP Dock Appointment Scheduling

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

Topic Area:
SAP Dock Appointment Scheduling

Version 1.0
November 2013
# Document History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>First release of this guide</td>
</tr>
</tbody>
</table>
## Typographic Conventions

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

## Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔴</td>
<td>Caution</td>
</tr>
<tr>
<td>🔴</td>
<td>Important</td>
</tr>
<tr>
<td>🔴</td>
<td>Note</td>
</tr>
<tr>
<td>🔴</td>
<td>Recommendation or Tip</td>
</tr>
<tr>
<td>🔴</td>
<td>Example</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Business Scenario ............................................................................................................. 1
2. Background Information ................................................................................................... 1
3. BOPF – Overview and Architecture .................................................................................. 3
   3.1 BOPF - Business Object Processing Framework ......................................................... 3
   3.2 BOPF Modeling Tool ................................................................................................... 6
   3.3 BOPF Consumer Implementation Basics ...................................................................... 9
       3.3.1 Service Manager ................................................................................................. 9
       3.3.2 Query .................................................................................................................. 10
       3.3.3 Retrieve ............................................................................................................... 11
       3.3.4 Retrieve By Association (Standard) ...................................................................... 12
       3.3.5 Do Action (with Action Parameters) ................................................................. 12
       3.3.6 Convert Alternative Key .................................................................................... 13
       3.3.7 Modify (Example Coding: Creation of a Planned Appointment) ...................... 13
4. Step-by-Step Procedure .................................................................................................... 17
   4.1 BOPF Enhancement Workbench .................................................................................. 17
       4.1.1 Overview .......................................................................................................... 17
       4.1.2 Creating an Enhancement Object ....................................................................... 18
       4.1.3 Creating Field Extensions ................................................................................. 21
       4.1.4 Creating Actions .............................................................................................. 23
       4.1.5 Creating a Consistency Validation ..................................................................... 24
       4.1.6 Creating a Determination .................................................................................. 28
   4.2 User Interface Enhancements ...................................................................................... 38
       4.2.1 Add additional attribute to a FORM UIBB ....................................................... 39
       4.2.2 Add additional attribute to a List View ............................................................. 47
   4.3 Recommended Enhancements/Best-Practice ................................................................. 51
1. Business Scenario

You are working in a customer implementation of SAP Dock Appointment Scheduling. The customer would like to use SAP Dock Appointment Scheduling for planning inbound or outbound transports, and needs to adapt the application to their particular processes. This guide provides you with an overview of the main enhancement possibilities available, and some step-by-step procedures for common requirements.

2. Background Information

There are multiple technical possibilities for enhancing and adapting SAP Dock Appointment Scheduling. SAP Dock Appointment Scheduling is a Web Dynpro Application using the frameworks Floorplan Manager (FPM), Business Object Framework (BOPF), and Floor Plan Manager BOPF integration (FBI). These frameworks offer a lot of enhancement and adaption options.

SAP Dock Appointment Scheduling also uses Microsoft Silverlight. Enhancements of the user interfaces of SAP Dock Appointment Scheduling implemented with Microsoft Silverlight are not subject of this guide.

This document does not intend to provide a complete and detailed description of all possible enhancements. Instead, it describes the usage of the available enhancement technologies based on some basic examples. These examples are chosen to be representative for similar enhancements in multiple areas of the application. In some cases, links to more detailed descriptions are provided.

In this document you can find:

- Basic information about BOPF objects and the BOPF modeling tool (chapter 3)
- Implementation basics for BOPF consumers (Query, Retrieve…) (chapter 3.3)
- Example coding for creating planned appointments (chapter 3.3.7)
- A step by step description of how to create an enhancement object in the BOPF framework (chapter 4)
- A step by step description of how to add an additional field to the user interfaces of SAP Dock Appointment Scheduling (chapter 4.2)
- A step by step description of how to set a field to mandatory (chapter 4.1.6)
- A step by step description of how to create additional validations and determinations (chapter 4.1.5 + 4.1.6)

**Recommendation**

We highly recommend reading the following detailed documents about enhancement or adaption options in FPM and SPI:

- The [FPM Developer's Cookbook](#)
- The [guide for enhancing FPM applications](#)
- The [FSI/SPI wiki](#)

The rest of this guide assumes familiarity with basic Web Dynpro and FPM concepts.
2.1 Note

⚠️ Caution

This document provides some examples of how dock appointment data can be accessed. The description of functions and methods in this document does not mean that these functions/methods have been released by SAP or that SAP guarantees that they will be kept stable. SAP may change/remove them without notice. Also, do not use any other methods or parameters from the classes mentioned. For example, /scwm/cl_docksched_ui_helper or /scwm/cl_dsapp_helper has many other methods. Only use the methods mentioned in this document.

The methods, functions, and classes mentioned in this document are not official programming interfaces and are not released for customers/partners. They can be changed or deleted by SAP at any time without prior/further notice. Any use is at your own risk.

Any software coding and/or code lines / strings ("code") included in this documentation are only examples and are not intended to be used in a productive system environment. The code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the code given herein, and SAP shall not be liable for errors or damages caused by the usage of the code, except if such damages were caused by SAP intentionally or grossly negligent.
3. BOPF – Overview and Architecture

SAP Dock Appointment Scheduling, as part of SAP EWM 9.1, is based on a set of frameworks that help to realize different aspects of the application.

The business objects are modeled and implemented with the **Business Object Processing Framework (BOPF)**.

The user interface is based on ABAP Web Dynpro and is realized with the **Floor Plan Manager (FPM)** which supports modeling, implementing and configuring the user interfaces.

Some graphical user interfaces are realized using **Microsoft Silverlight**; these parts are not relevant for any enhancements.

The **Floor Plan Manager BOPF Integration (FBI)** is used to connect the backend with the user interface. It provides the connection between the business objects in the backend with the corresponding user interface realized with the FPM.

To utilize the enhancement capabilities of SAP EWM 9.1, some general knowledge of these frameworks is required. Besides these frameworks, general knowledge of the following implementation and configuration technologies is a prerequisite for creating enhancements:

- **BOPF Enhancement Workbench** (Configuration / Implementation, part of the BOPF Framework)
- **Configurations of Web Dynpro Components**

The frameworks and technologies are described in the following sections to provide a very basic insight into how they are involved in the SAP Dock Appointment Scheduling application of SAP EWM 9.1 and how they are used for creating enhancements. This document cannot cover all aspects of these frameworks and technologies. Therefore links to more detailed information sources are provided where appropriate.

### 3.1 BOPF - Business Object Processing Framework

A business object is a representation of a type of uniquely identifiable business entity described by a structural model and an internal process model. For example, SAP Dock Appointment Scheduling uses business objects for dock appointments, loading points, and time slots.

Implemented business processes operate on business objects. Most important for the context of this document, a business object, its characteristics, and its configuration settings can be enhanced. The following is a brief look at the parts of a BOPF business object.

A BOPF business object model consists of the following entities:

**Nodes:**

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

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

**Associations:**

An association is a direct, unidirectional, binary relationship between two business object nodes.
Associations can be used to relate two nodes in a defined direction. The association can be used to navigate from one node (source node) to the related node (target node). The associated nodes can be nodes within one business object or in different business objects (cross business object association).

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

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

An action can be used to allow the external triggering of business logic (in contrast to a determination). When the action is performed, you must specify the key for the instances on which it is to be performed, unless it is a static action, and any input parameters that the action requires. An action can only be performed with the number of instances that is configured in the cardinality of the action. It is performed for all instances, unless an error in the action validation has occurred. If errors occur, then the behavior depends on the action settings.

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

Examples:
- **Sums** *(for example, planned appointment duration = sum of the durations of the assigned time slots)*
- **Statuses**
- **Properties to control the visibility of attributes in the screens**

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

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

<table>
<thead>
<tr>
<th>Determination Time</th>
<th>Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Retrieve</td>
<td>Determining contents of transient nodes before their first retrieval. After the first retrieval of a node instance, determinations for this determination time are not executed, as changes to data during retrieval are not allowed.</td>
</tr>
<tr>
<td>After Modify</td>
<td>Recalculation of fields that depend on changed fields. This is especially useful for derived fields that are of interest to the “outside world” and need to be updated immediately.</td>
</tr>
<tr>
<td>After Validation</td>
<td>This point in time can be used to modify data based on the outcome of consistency validations in the Determination &amp; Validation cycle. A typical use case is to perform some follow-up actions depending on whether there were error messages in the consistency validations.</td>
</tr>
<tr>
<td>Before Save (Finalize)</td>
<td>Determine data that must not be determined until the node is ready to be saved or for data that is not visible to the “outside world” (so its determination can be postponed until saving for performance reasons).</td>
</tr>
</tbody>
</table>
Before Save (Draw Numbers)
Determine data that must not be determined unless the transaction succeeds but may be used by other business objects. A typical use case for such very late changes is drawing numbers to assure gapless numbering.

During Save
Determine data that must not be determined unless the transaction succeeds. Determinations for this determination time are executed once at most in a LUW.

After Commit
Determine data after a transaction was successfully committed. A typical use case for this determination time is starting asynchronous processes.

After Failed Save Attempt
Perform cleanups after an attempt to save a transaction was rejected during the Finalize or Check before Save stages. A determination is only triggered if request nodes are assigned to it and instances of these request nodes are changed.

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

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

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

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

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

The models of the SAP Dock Appointment Scheduling business objects can be displayed with the BOPF Modeling Tool. It can be started with transaction /BOBF/CONF_UI. It allows browsing through the list of the business objects of the application. From here, you can navigate to the details of each business object to display its node structure and hierarchy, the configuration, the DDIC structures for each node, the node elements (e.g. associations, actions, determinations, validations and queries), etc. Moreover, it allows navigating to the implementing ABAP classes of the business object.

On the initial screen (Picture 1) the user can browse through the available business objects of SAP Dock Appointment Scheduling. These are:
Business Process Objects:
- /SCWM/DSAPP: Appointments of Dock Appointment Scheduling
- /SCWM/DSSLOT: Time Slots of Dock Appointment Scheduling

Master Data Objects:
- /SCWM/DSLOAP: Loading Point and Docking Location of Dock Appointment Scheduling

For each business object you can show the Business Object Detail Browser:

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

- **Combined Structure:**
  This DDIC structure includes the data structure of a node. In addition, it includes a fixed BOPF DDIC structure that contains the node instance key (KEY), the key of the direct parent node instance (PARENT_KEY), and the key of the related business object instance (ROOT_KEY).
- **Data Structure:**
  This DDIC structure contains the attributes of the node, representing the node data.

- **Transient Structure (tr.):**
  Contains the transient attributes of a node, that is, attributes which do not persist but are only filled and used during runtime.

- **Extension Include & Extension Include (tr.):**
  The extension include is important for field extensions on a node. With this include, all extension fields that are to persist are added (using append structures). Extension fields which are only relevant at runtime and do not persist are placed in the corresponding transient extension include.

- **Database Table:**
  Shows the database table where the persistent node information gets stored.

---

**Picture 3: Business Object Detail Browser – Node Elements (Example: Determination)**

When expanding the node elements, you can also navigate to a node and the elements assigned to it (for example, associations, determinations, validations, actions and queries, as described in the previous sections). Moreover, the details for each of these elements can be displayed from here. For example, the details of an action include a link to the implementing class of this action and – if the action has parameters – the corresponding parameter structure.

The details of the node elements like actions, validations, determinations, etc. are the starting point for identifying places in the coding where a specific functionality is implemented. Within the implementing classes of the node elements, of course, further classes and their methods are used to realize its functionality.
3.3 BOPF Consumer Implementation Basics

In this section, we give examples on how to implement BOPF consumers, that is, how to use core services that allow creating, accessing, and modifying business object instances.

**Note**

For simple enhancements like adding new fields on a user interface, you do not require this section. This section describes how to access the dock appointment scheduling objects in your own code.

### 3.3.1 Service Manager

A business object can be accessed via a service manager.

**Example**

The following coding shows how to get an instance of the service manager, for example, of the business object Dock Appointment:

```plaintext
*---------------------------------------------------------------------*
* Report ZREP_SRV_MGR
*---------------------------------------------------------------------*
* How to get a service manager instance and use it to access BOPF  *
*---------------------------------------------------------------------*

REPORT zrep_srv_mgr.

DATA: lo_srv_mgr TYPE REF TO /bobf/if_tra_service_manager.

* Get an instance of a servicemanager for e.g. BO Dock Appointments
lo_srv_mgr = /bobf/cl_tra_serv_mgr_factory=>get_service_manager(  
    iv_bo_key = /scwm/if_dsapp_c=>sc_bo_key ).
```

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY</td>
<td>Search, execute a business object query.</td>
</tr>
<tr>
<td>RETRIEVE</td>
<td>Read data for a given set of node instance keys.</td>
</tr>
<tr>
<td>RETRIEVE_BY_ASSOCIATION</td>
<td>Read data via association.</td>
</tr>
<tr>
<td>DO_ACTION</td>
<td>Execute a given action of a business object node.</td>
</tr>
<tr>
<td>CONVERT_ALTERN_KEY</td>
<td>Convert an alternative key to the technical key.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>Create, change and delete business object node instances.</td>
</tr>
</tbody>
</table>

The following coding examples and descriptions of the semantics of the corresponding method parameters illustrate the usage of the service manager methods to access BOPF business objects. We also include corresponding examples in a small step by step demo of report ZREP_SRV_MGR.

In chapter 3.3.7 Modify, we show a demo report which creates planned appointments.
3.3.2 Query

**Example**

The coding example shows how to call a BOPF query to get the keys of all appointments of a loading point for one day.

The declaration part is also used in the examples that follow.

To start a query, method QUERY of the service manager instance is used:

```
REPORT ZREP_SRV_MGR.

DATA: lo_srv_mgr TYPE REF TO /bobf/if_traservice_manager.
```

**Declarations for query**

```
DATA: lt_selpar TYPE /bobf/t_frw_query_selparam,
     ls_selpar TYPE /bobf/s_frw_query_selparam,
     lo_message TYPE REF TO /bobf/if_frw_message,
     ls_key TYPE /bobf/s_frw_key,
     lt_key TYPE /bobf/t_frw_key.
```

**Declarations for retrieve**

```
DATA: lt_failed_key TYPE /bobf/t_frw_key,
     lt_root TYPE /scwm/t_dsapp_root_k.
```

**Declarations for retrieve by association**

```
DATA: lt_slotassign TYPE /scwm/t_dsapp_slotassign_k,
     lt_slotassign_key TYPE /bobf/t_frw_key.
```

**Declarations for action**

```
DATA: lv_proctime TYPE /scwm/dsapp_proctime,
     lv_timeutc TYPE timestamp,
     lr_action_param TYPE REF TO /scwm/s_dsapp_a_status.
```

**Declarations for convert altern key**

```
DATA: lt_docno TYPE TABLE OF /scwm/s_dsapp_doc,
     lt_root_key TYPE /bobf/t_frw_key.
```

FIELD-SYMBOLS:

```
<ls_root> TYPE /scwm/s_dsapp_root_k.
```
Get an instance of a servicemanager for e.g. BO Dock Appointments

```java
lo_srv_mgr = /bobf/cl_tru_serv_mgr_factory=>get_service_manager(
    iv_bo_key = /scwm/if_dsapp_c=>sc_bo_key )
```

* Query example: get appointments of one day of a loading point

```java
ls_selpar-attribute_name = /scwm/if_dsapp_c=>sc_query_attribute-root-select_by_elements_dt-loadpoint.
ls_selpar-sign = wmegc_sign_inclusive.
ls_selpar-option = wmegc_option_eq.
ls_selpar-low = 'LP_TEST_01'.
APPEND ls_selpar TO lt_selpar.

ls_selpar-attribute_name = /scwm/if_dsapp_c=>sc_query_attribute-root-select_by_elements_dt-start_date_sel.
ls_selpar-sign = wmegc_sign_inclusive.
ls_selpar-option = wmegc_option_eq.
ls_selpar-low = '20131112'.
APPEND ls_selpar TO lt_selpar.

lo_srv_mgr->query(
    EXPORTING
        iv_query_key = /scwm/if_dsapp_c=>sc_query-root-
        select_by_elements_dt
        it_selection_parameters = lt_selpar
    IMPORTING
        eo_message = lo_message
        et_key = lt_key )
```

### 3.3.3 Retrieve

**Example**

The coding example shows how to retrieve the data for the root node keys that were found by the query in 3.3.2. Method RETRIEVE of the service manager instance is used.

* retrieve appointment data

```java
lo_srv_mgr->retrieve(
    EXPORTING
        iv_node_key = /scwm/if_dsapp_c=>sc_node-root
        it_key = lt_key
        iv_edit_mode = /bobf/if_conf_c=>sc_edit_read_only
    IMPORTING
        eo_message = lo_message
        et_data = lt_root
        et_failed_key = lt_failed_key ).
```
3.3.4 Retrieve By Association (Standard)

**Example**
The coding example shows how to retrieve the data of the associated time slots assigned to the appointment node keys that were found by the query in 3.3.2. Method RETRIEVE_BY_ASSOCIATION of the service manager instance is used with the composition association from root to node DSAPP_SLOTASSIGN of the assigned time slots.

* retrieve time slots of the appointments by an association
lo_srv_mgr->retrieve_by_association(
  EXPORTING
  iv_node_key = /scwm/if_dsapp_c=>sc_node-root
  lt_key = lt_key
  iv_association = /scwm/if_dsapp_c=>sc_association-root-dsapp_slotassign
  iv_fill_data = abap_true
  iv_edit_mode = /bobf/if_conf_c=>sc_edit_read_only
  IMPORTING
  eo_message = lo_message
  et_data = lt_slotassign
  et_target_key = lt_slotassign_key
  et_failed_key = lt_failed_key).

3.3.5 Do Action (with Action Parameters)

**Example**
The coding example shows how to start an action for a given set of appointment root node instances represented by the corresponding root node keys. In this example, the action CHECKIN (Arrival at Checkpoint) of the appointment root node is called.

* perform action "arrival at checkpoint" (check-in)
CREATE DATA lr_action_param.
* set arrival at checkpoint time
GET TIME STAMP FIELD lv_timeutc.
lv_proctime = lv_timeutc.
lr_action_param->proctime = lv_timeutc.
lo_srv_mgr->do_action(
  EXPORTING
  iv_act_key = /scwm/if_dsapp_c=>sc_action-root-checkin
  it_key = lt_key
  is_parameters = lr_action_param
  IMPORTING
  eo_message = lo_message
  et_failed_key = lt_failed_key ).
3.3.6 Convert Alternative Key

**Example**
The coding example shows how to convert a list of appointment numbers (docno) into the corresponding root node keys. Method CONVERT_ALTERN_KEY of the service manager instance is used.

```plaintext
LOOP AT lt_root ASSIGNING <ls_root>. 
    APPEND <ls_root>-docno TO lt_docno.
ENDLOOP.

* convert alternative keys into BOBF key
lo_srv_mgr->convert_altern_key( 
  EXPORTING 
    iv_node_key = /scwm/if_dsapp_c=>sc_node-root 
    iv_altkey_key = /scwm/if_dsapp_c=>sc_alternative_key-root-docno 
    it_key = lt_docno 
  IMPORTING 
    et_key = lt_root_key ).
```

3.3.7 Modify (Example Coding: Creation of a Planned Appointment)

While the coding examples of the previous subsections demonstrated how to access BOPF business objects, we now take a look at how to create business object (node) instances. For these purposes, method MODIFY of the service manager is used.

**Example**
The coding example shows how to create planned appointments using action “ASSIGNSLOTS” of the appointment business object.

The loading point of the appointment to be created is read to get the time zone.

```plaintext
REPORT z_create_appointment.

DATA:
  lv_loadpoint   TYPE /scwm/dsloadpoint, 
  lv_date        TYPE d, 
  lv_time        TYPE t, 
  ls_lp_data     TYPE /scwm/s_dslp_root_k, 
  lv_start_time  TYPE /scwm/dsslot_start, 
  lv_length      TYPE /scwm/dsappdura_pl, 
  lv_timezone    TYPE ttzz-tzone.
```

```plaintext
DATA:
  lv_key         TYPE /bobf/conf_key, 
  lt_key         TYPE /bobf/t_frw_key, 
```
ls_key TYPE /bobf/s_frw_key,
lt_app_mod TYPE /bobf/t_frw_modification,
ls_app_mod TYPE /bobf/s_frw_modification.

DATA:
lo_srv_mgr TYPE REF TO /bobf/if_tra_service_manager,
lo_tra_mgr TYPE REF TO /bobf/if_tra_transaction_mgr,
lo_message TYPE REF TO /bobf/if_frw_message,
lo_change TYPE REF TO /bobf/if_tra_change.

DATA:
rs_app_data TYPE REF TO /scwm/s_dsapp_root_k,
rs_slotassign TYPE REF TO /scwm/s_dsapp_a_slassign.

DATA:
lv_rejected TYPE abap_bool,
lt_rejecting_bo_key TYPE /bobf/t_frw_key2.

* create an appointment for loading point LP_TEST_01 and date 12.11.2013, start time 10:00
lv_loadpoint = 'LP_TEST_01'.
lv_date = '20131112'.
lv_time = '100000'.
lv_length = 3600.

* get loading point data (slot duration and time zone)
CALL METHOD /scwm/cl_docksched_ui_helper=>get_lp_data_by_altkey
EXPORTING
   iv_loadpoint = lv_loadpoint
RECEIVING
   rs_lp_data = ls_lp_data.

* time zone and slot duration of the loading point
lv_timezone = ls_lp_data-tzone.

* get transaction manager
lo_tra_mgr = /bobf/cl_tra_trans_mgr_factory=>get_transaction_manager( ).

* get service manager to create an appointment
lo_srv_mgr = /bobf/cl_tra_serv_mgr_factory=>get_service_manager( iv_bo_key = /scwm/if_dsapp_c=>sc_bo_key ).

Step 1: Create appointment business object instance
As a first step, a new instance of business object appointment is created. First, the modification table is set up to contain an entry for the creation (change mode is set to CREATE) of a new root node instance. Then the data for the new root node instance is assembled. Finally, the modification table is passed to method MODIFY of the service manager. The business object node key is determined before the creation of the root node is done to use it for the creation of the assigned time slots later.
ls_app_mod-change_mode = 'C'. "create
ls_app_mod-node_cat = /scwm/if_dsapp_c=>sc_node_category-root-root.

* convert date/time using time zone of the loading point
CONVERT DATE lv_date TIME lv_time
    INTO TIME STAMP lv_start_time TIME ZONE lv_timezone.

* get a new key for an appointment
CALL METHOD lo_srv_mgr->get_new_key
    RECEIVING
ev_key = lv_key.

* key of the new appointment instance
ls_app_mod-key = lv_key.

* fill appointment data
CREATE DATA rs_app_data.
    rs_app_data->key = lv_key.
    rs_app_data->root_key = lv_key.
    rs_app_data->loadpoint = lv_loadpoint.
* rs_app_data->carrier = lv_carrier. "Carrier
* rs_app_data->mtr = lv_mtr. "Means of Transport
rs_app_data->transmeansid = 'TRUCK_01'.
* no more data is needed, the other data is filled via a determinations with
hin the appointment
ls_app_mod-data = rs_app_data.
APPEND ls_app_mod TO lt_app_mod.

* create appointment with modify command
CALL METHOD lo_srv_mgr->modify
    EXPORTING
        it_modification = lt_app_mod
    IMPORTING
        eo_change = lo_change
        eo_message = lo_message.

Step 2: Assignments of time slots using an action
The time slots are assigned by using the action ASSIGNSLOTS with the start time and the duration of
the appointment.

* get key of the appointment root to create the slot assignments
ls_key-key = lv_key.
APPEND ls_key TO lt_key.

* start time and appointment duration
CREATE DATA rs_slotassign.
    rs_slotassign->loadpoint = lv_loadpoint.
    rs_slotassign->start_time = lv_start_time.
    rs_slotassign->app_length = lv_length.

* call action to assign time slots
CALL METHOD lo_srv_mgr->do_action
    EXPORTING
        iv_act_key = /scwm/if_dsapp_c=>sc_action-root-assignslots
        it_key = lt_key
is_parameters = rs_slotassign
IMPORTING
eo_change = lo_change
eo_message = lo_message.

Step 3: Save the planned appointment
The final step is to call the save method of the transaction manager instantiated in the beginning.

* call save of the transaction manager
lo_tramgr->save(
IMPORTING
  ev_rejected = lv_rejected
  eo_change = lo_change
  eo_message = lo_message
  et_rejecting_bo_key = lt_rejecting_bo_key
)
4. Step-by-Step Procedure

In this chapter, we provide a list of sample adaptions and enhancements and how to implement them for SAP Dock Appointment Scheduling.

4.1 BOPF Enhancement Workbench

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

4.1.1 Overview

As of SAP EWM 9.1, the BOPF Enhancement Workbench is available to enhance the standard BOPF business objects in SAP Dock Appointment Scheduling. It can be used to create, change or delete enhancements of the standard BOPF business objects. Such enhancements can be enhanced again with the same tool, that is, nested enhancements are also possible. The BOPF Enhancement Workbench supports creating, changing, or deleting the following objects:

- Subnodes
- Actions and action enhancements
- Determinations
- Consistency and action validations
- Queries

The BOPF Enhancement Workbench does not allow you to create new business objects. Moreover, a standard EWM BOPF business object must have been declared to be extensible by SAP Development before it can be enhanced. Only such business objects can be enhanced. The same applies to a business object’s entities, like nodes, actions, etc. They can only be enhanced if SAP Development has declared them to be extensible.

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

Picture 4: BOPF Enhancement Workbench Initial Screen
On the initial screen you can see the business objects of the SAP Dock Appointment Scheduling application that are allowed to be enhanced in general. Whenever you want to enhance one of the listed business objects, the first step is to create an enhancement object for this business object. For the enhancement object, the original business object represents the super business object.

An important attribute of enhancement objects is that the enhancement object neither replaces nor represents a copy of the standard business object. Instead, it serves as a container for all the enhancements that you add to the business object via the Enhancement Workbench. At runtime, the standard business object functionality is still executed, and the enhancements are executed as well.

4.1.2 Creating an Enhancement Object

The first step for enhancing a standard SAP Dock Appointment Scheduling business object is to create an enhancement object for this business object. The business object associated with this enhancement object is also called the super business object of an enhancement object. As described in section 4.1.1, it is important to realize that this enhancement object neither replaces nor represents a copy of the standard business object. Instead, it serves as a container for all the enhancements that you add to the business object via the Enhancement Workbench. At runtime, the standard Business Object functionality is still executed, and the enhancements are executed as well. Of course, all code corrections and changes to the standard business object and its entities will always be present.

1) Start the BOPF Enhancement Workbench (/BOBF/CUST_UI) and select the business object to be enhanced. Let’s assume we create an enhancement object for the appointment business object (the technical name is /SCWM/DSAPP).
2) Right-click on the selected business object and select the option Create Enhancement in the popup menu.
3) A wizard guides you through the next steps for creating the enhancement object.

4) On the first screen of the wizard, click on Continue.
On the next screen of the wizard, you can see the name of the super business object and the following fields, ready for input:

**Enhancement Name:** The name of your enhancement.
Example: ENH_DSAPP

**Description:** A description of your enhancement.
Example: Demo Enhancement for Dock Appointment

**Namespace:** The namespace that your enhancement will be associated with. This will be added at the beginning of the final technical name for your enhancement object.

**Prefix:** A prefix that will be added between the namespace and the enhancement name in the final technical name of your enhancement. It is not a mandatory field and can be left empty if not required.
Example: Z
6) Click on Continue. The next screen of the wizard uses the entries to propose a name for your enhancement, and the name of the constants interface that will be created for the new enhancement object. Example (with the given entries):

*Technical Name: ZENH_DSAPP*
*Constants Interface: ZIF_ENH_DSAPP_C*

7) Click on Continue. On the next screen of the wizard, you define whether you allow your enhancement to be enhanced in further enhancements (that is, nested enhancements are possible).

8) At any step of the wizard you can go back to all preceding steps again by clicking Back. You can adjust the data entered until you have completed the wizard. Click on Complete to finally create your enhancement object with the entered specification.

The new enhancement object can now be used to create enhancements of the corresponding super business object. The creation of the different types of enhancements on node and action level is described in the following sections.
4.1.3 Creating Field Extensions

Customers and partners may require additional fields to be stored with the business objects, to be entered and displayed on the user interface, or to be transferred between external systems (for example, an ERP system) and SAP Dock Appointment Scheduling via corresponding services. The basis for these kinds of enhancements is creating field extensions for the business objects delivered with SAP Dock Appointment Scheduling. Such field extensions can be created using the BOPF Enhancement Workbench.

**Example**

In this example, the dock appointment data in business object /SCWM/DSAPP is enhanced by the driver’s license ID.

1) Start the BOPF Enhancement Workbench (/BOBF/CUST_UI) and select the enhancement object for the business object to be enhanced with additional fields. Let’s assume we create a field extension for the Dock Appointment business object (Enhancement object ZENH_DSAPP created in section 4.1.2).

2) Double click on the node to be enhanced by additional fields (in our example, the root node of the Dock Appointment business object). In the details of the node in the right-hand screen area, you can see the extension includes that will carry the additional fields. Field extensions are always assigned to such an extension include.

- Double-click on the **Persistent Extension Include** to start adding fields that are to persist in the database.
- Double-click on the **Transient Extension Include** to start adding fields that are intended to be only used during runtime and do not persist in the database.

3) On the following screen you can see the DDIC Editor (which is analogous to transaction SE11 for DDIC objects). Click on the **Append Structure** button to create a new append for the chosen extension include. This append will contain your extension fields. Usually, you just need one such append to contain all your extension fields. But it is also possible to create additional appends for the...
extension include, for example, to separate extensions from different partners.

Example append: `<Zzenh_DEmo_DSAPP_ROOT>`.

4) Enter a short description for the append structure and add the extension fields to be included in this append (component, typing method and component type). After the extension fields are correctly specified, save and activate the append structure.

With the described four steps, the new extension fields are now part of the corresponding node structures, the table types and the database table (provided that you have added the extension fields in the Persistent Extension Include). Both the transient and persistent extension fields are now ready to be used within further enhancements, for example, in the business logic or the user interface (see chapter 4.2 User Interface Enhancements).
4.1.4 Creating Actions

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

1) Open the corresponding enhancement object and select the node to be extended with a new action. Right-click to open the context menu of the node, choose Create Action to start the wizard. Example: Root Node of the Dock Appointment in the example Enhancement Object ZENH_DSAPP.

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

   **Example:**

<table>
<thead>
<tr>
<th>Action Name</th>
<th>&lt;ZENH_DSAPP_CARRAPP&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>&lt;Approve Appointment by Carrier&gt;</td>
</tr>
</tbody>
</table>

3) In the second screen, define the following information for the new action.

   a) Implementing Class: Example: <ZCL_ENH_A_DSAPP_CARRAPP>. The implementing class must implement interface /BOBF/IF_FRW_ACTION.
   b) Action Cardinality:
      Defines how many node instances the action can operate on during one action call. The following are the action cardinality types:
      - **Multiple Node Instances**: Select if the action always operates on one or more node instances.
      - **Single Node Instance**: Select if the action operates on exactly one single node instance for each call.
      - **Static Action (No Node Instances)**: Select if the action does not operate on any node instances.
   c) Parameter Structure:
      Some actions need an additional importing parameter. Enter a name for the parameter structure and create the structure by double-clicking the name.

4) On the next screen of the wizard, specify whether the new action will be extensible or not. Set the flag “Action can be enhanced” if you want to allow the addition of enhancements to the new action (that is, adding pre- and post-action Enhancements and action validations).

5) Click on Complete to finalize the creation of the new action.

6) Finally, you need to implement your business logic in the action’s implementing class that you have specified in the previous steps. Double-click on the implementing class in the action details to start the implementation.
4.1.5 Creating a Consistency Validation

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

**Example**

In this example, a validation for the driver’s license ID of a dock appointment defined in chapter 4.1.3 Creating Field Extensions is implemented.

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

1) Open the corresponding enhancement object and select the node to be extended with a new consistency validation. Right-click to open the context menu of the node, and choose Create Consistency Validation to start the wizard.

   **Example:**
   The root node of the appointment in the example enhancement object ZENH_DSAPP created in section 4.1.2.

2) On the first screen of the wizard, specify the name for the new consistency validation, and a description on the semantic and purpose of this new entity.

   **Example:**
<table>
<thead>
<tr>
<th>Validation Name</th>
<th>&lt;ZENH_DSAPP_CONVAL&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>&lt;Demo Enh. Cons. Validation&gt;</td>
</tr>
</tbody>
</table>

3) On the second screen, define the implementing class of the new consistency validation. The implementing class must implement interface /BOBF/IF_FRW_VALIDATION.

   **Example:** <ZCL_ENH_V_DSAPP_CONVAL>.

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

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

   **Example:**
<table>
<thead>
<tr>
<th>Request Node</th>
<th>Node</th>
<th>Create</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Root</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>DSAPP_REFDOC</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>DSAP_SLOTASSIGN</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
With these settings, the consistency validation will be triggered when instances of the root node are created or updated.

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

Options:

- **Return messages:**
  This is the default behavior of consistency validation. The validation implementation returns messages for inconsistent instances to the consumer.

- **Return messages and prevent saving:**
  Select this option for the validation impact if the inconsistency of a node instance must be solved before saving the transaction.

- **Return messages and set a consistency status:**
  If the base object (super object) contains a consistency status, this status can be influenced by a consistency validation. Choose the appropriate status variable. If a changed instance fails the consistency validation, this status is automatically set to inconsistent.

For this example choose **Return messages and prevent saving:**
6) Click on Complete to finalize the creation of the new consistency validation.

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

In this example, first use transaction SE91 and create a new message class `<ZENH_DSAPP>`
with the following message:

<table>
<thead>
<tr>
<th>Message No.</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>&lt;Driver license ID &amp;1 is not valid&gt;</td>
</tr>
</tbody>
</table>

Then implement your chosen method, for example, a method which checks if the driver license ID is valid. In this example, method CHECK_LICENSE, which holds the customer’s business logic, is used.

If the check fails, send the error message.

Example coding:

METHOD /bobf/if_frw_validation~execute.

DATA: lt_app_root  TYPE /scwm/t_dsapp_root_k.
DATA: ls_key      TYPE /bobf/s_frw_key.
DATA: lv_msg      TYPE string.                        "#EC NEEDED
DATA: ls_msg      TYPE symsg.
DATA: lv_is_valid TYPE boole_d.

FIELD-SYMBOLS:
   <ls_app_root> TYPE /scwm/s_dsapp_root_k.

IF it_key IS INITIAL. " Property key
   RETURN.
ENDIF.

* clear exporting parameter
   CLEAR: eo_message, et_failed_key.

* get current appointment data
   io_read->retrieve(
      EXPORTING
         iv_node = /scwm/if_dsapp_c=>sc_node-root
         it_key  = it_key
         iv_fill_data = abap_true
      IMPORTING
         et_data  = lt_app_root ).

   LOOP AT lt_app_root ASSIGNING <ls_app_root>.

   * no check for provisional appointments
      IF <ls_app_root>-procstatus <> 'F'.

      lv_is_valid = check_license(
         i_driver = <ls_app_root>-driver
         i_license =
            <ls_app_root>-zz_driver_license_id ).

      IF lv_is_valid = abap_false.
      IF eo_message IS NOT BOUND.
         eo_message = /bobf/cl_frw_factory=>get_message( ).
      ENDIF.
   ENDLOOP.
MESSAGE e000{zenh_dsapp}
WITH <ls_app_root>-zz_driver_license_id INTO lv_msg.
MOVE-CORRESPONDING sy TO ls_msg.
eo_message->add_message(
  ls_msg = ls_msg
  iv_node = /scwm/if_dsapp_c=>sc_node-root
  iv_key = <ls_app_root>-key
  iv_attribute = zif_enh_dsapp_c=>sc_node_attribute-root-
  zz_driver_license_id ) .
  ls_key-key = <ls_app_root>-key.
  APPEND ls_key TO et_failed_key.
  CONTINUE.
ENDIF.

ENDIF.

ENDLOOP.
ENDMETHOD.

4.1.6 Creating a Determination

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

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

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

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

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

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

Example: As soon as new time slots are assigned to an appointment root node instance, the appointment length must be recalculated immediately in order to show the new appointment length on the consumer’s user interface.
B) Derive dependent data before saving (Before Save):
The trigger condition is checked as soon as the consumer saves the whole transaction. If the save of the transaction fails, these determinations could run multiple times. In contrast to the “Derive dependent data immediately after modification” pattern, the framework evaluates all changes done so far in the current transaction to check the trigger condition. Because this evaluation only takes place at the save phase of the transaction, this pattern is recommended for time consuming determinations.

C) Fill transient attributes of persistent nodes (After Loading):
The determination is automatically executed before the consumer accesses a transient node attribute of the assigned node for the first time. This allows you to derive the initial values of the attribute. In addition, these determinations are executed after each modification of a node instance. This allows you to recalculate the transient field if its derivation source attribute has been changed by the modification.

Example: The standard code for the means of transport is displayed in the user interface. Therefore, you can use a determination that determines the standard code from the table of means of transport at this point in time.

D) Derive instances of transient nodes (Before Retrieve):
The determination is executed before the consumer accesses the assigned transient node of the determination and allows the creation, update or deletion of transient node instances. These determinations are used to create and update instances of transient nodes. Because the determinations are executed before each access to their assigned transient node, they must ensure that the requested instances are in a consistent state.

E) Create Property:
The determination is used to set properties for nodes or node attributes.

Example: You can define an attribute to be mandatory for an external appointment planer of a carrier.
New enhancement determinations can be added to extensible standard nodes. These examples demonstrate how to do this:

### 4.1.6.1 Example 1: Create a Determination to set an Attribute to Mandatory

1) Open the enhancement object ZENH_DSAPP and select the root node to be extended with a new determination. Right click to open the context menu of the node, and choose Create Determination to start the wizard.

2) In the first screen of the wizard, specify the name for the new determination, and a description on the semantic and purpose of this new entity. Example: `<ZENH_DSAPP_DEMODET>`

![Picture 15: Define determination name and description]

3) In the second screen, define the implementing class of the new determination. The implementing class must implement interface /BOBF/IF_FRW_DETERMINATION. Example: `<ZCL_ENH_D_DSAPP_DEMODET>`.
4) Maintain the determination pattern to determine properties, choose option *Create Properties*.

![Image of the 'Create Properties' option](image)

**Picture 16: Define determination pattern**

The enhancement determination is created.

![Image of Business Object Builder](image)

**Picture 17: The enhancement determination in the BOPF enhancement browser**
5) Click on the implementing class and implement method “EXECUTE”

An instance of class /BOBF/CL_LIB_H_SET_PROPERTY, used to set the properties, is already part of the method interface and can be used to set properties.

Example coding:

```
METHOD execute.
  DATA: lt_app_root  TYPE /scwm/t_dsapp_root_k.
  DATA: lt_app_root_lock  TYPE /bobf/t_frw_lock_node.
  DATA: lv_only_limited_scope  TYPE boole_d.
  DATA: lv_display_mode  TYPE boole_d.

  FIELD-SYMBOLS:
    <ls_app_root>  TYPE /scwm/s_dsapp_root_k.

  IF it_key  IS INITIAL.  " Property key
                      RETURN.
  ENDIF.
  * check if called by carrier user.
  lv_only_limited_scope =
    /scwm/cl_dsapp_helper=>limited_scope_for_carr_planner( ).
  IF lv_only_limited_scope = abap_true.
    io_read->retrieve( EXPORTING
      iv_node     = /scwm/if_dsapp_c=>sc_node-root
      it_key      = it_key
      iv_fill_data = abap_true
    IMPORTING
      et_data     = lt_app_root ).

  * check if there is a lock node to find out if display mode valid
  CLEAR lt_app_root_lock.
  io_read->retrieve_by_association( EXPORTING
    iv_node     = /scwm/if_dsapp_c=>sc_node-root
    it_key      = it_key
    iv_association = /scwm/if_dsapp_c=>sc_association-root-lock
    iv_fill_data = abap_true
  IMPORTING
    et_data     = lt_app_root_lock ).
```
LOOP AT lt_app_root ASSIGNING <ls_app_root>.

* Appointment status is not provisional
   IF <ls_app_root>-procstatus <> 'F'.

* In display mode mandatory attribute is not set
   READ TABLE lt_app_root_lock
     WITH KEY parent_key = <ls_app_root>-key
     TRANSPORTING NO FIELDS.

   IF sy-subrc = 0.
     lv_display_mode = abap_false.
   ELSE.
     lv_display_mode = abap_true.
   ENDIF.

   IF lv_display_mode = abap_false.
   CALL METHOD io_property->set_attribute_mandatory
     EXPORTING
       iv_attribute_name = /scwm/if_dsapp_c=>sc_node_attribute-root-transmeansid
       iv_key = <ls_app_root>-key
       iv_value = abap_true.
   ENDIF.

   ENDIF.

ENDLOOP.

ENDIF.

ENDMETHOD.

In this example the Means of Transport ID field is set to mandatory for an external planer of a carrier:

![TCL_EL_G kissing.svg](https://openclipart.org/image/300px/svg_to_png/30019/TCL_EL_G_kissing.svg)

Picture 19: Means of Transport ID is mandatory
**Recommendation**
Whenever you want to check in a validation or determination if an appointment planner of a carrier is working on the appointment, you can use method

```
/scwm/cl_dsapp_helper=>limited_scope_for_carr_planner()
```

as shown in the coding example above.

### 4.1.6.2 Example 2: Create a Determination to Disable a Button (an Action)

You can also use a determination to set properties to deactivate a button in a user interface, in the same way as described in the previous chapter.

In the implementation of the determination class, you just have to call the following method:

* Example: set button "Cancel" to inactive

```call_method
CALL METHOD io_property->set_action_enabled
EXPORTING
  iv_action_key = /scwm/if_dsapp_c=>sc_action-root-cancel
  iv_key = <ls_app_root>-key
  iv_value = abap_false.
```

This coding example disables the Cancel button.

### 4.1.6.3 Example 3: Create a Determination to Determine an Attribute Value

In this example, the driver’s license ID is determined by the driver’s name, if the driver is known by the system.

1) Create an enhancement determination for the root node of enhancement object ZENH_DSAPP with pattern A = Derive dependent data immediately after modification (After Modify).

2) Enter determination name and description
3) Define the implementing class, for example <ZCL_ENH_D_DSAPP_DETLC>. The implementing class must implement interface /BOBF/IF_FRW_DETERMINATION.

4) Maintain the determination pattern to determine data after modification:
5) Define the requested nodes that trigger the processing of the determination:

![Diagram showing the determination pattern setup](image)

**Picture 22: Define determination pattern**

In this example, creating or changing the root node triggers the determination.

6) Implement your logic to determine the driver’s license in class `<ZCL_ENH_D_DSAPP_DETLC>`.

Implement a method (for example: `<check_license>`) to determine the driver's license ID by the driver’s name with your business logic.

If the license ID is changed, update the data with the modify command:

```abap
METHOD /bobf/if_frw_determination-execute.

DATA: lt_app_root TYPE /scwm/t_dsapp_root_k,
    lv_license_id TYPE char20,
    rs_app_data TYPE REF TO /scwm/s_dsapp_root_k,
    lt_app_mod TYPE /bobf/t_frw_modification,
    ls_app_mod TYPE /bobf/s_frw_modification.

FIELD-SYMBOLS:
    <ls_app_root> TYPE /scwm/s_dsapp_root_k.

IF it_key IS INITIAL. " Property key
    RETURN.
ENDIF.

io_read->retrieve(
    EXPORTING
        iv_node = /scwm/if_dsapp_c=>sc_node-root
        it_key = it_key
        iv_fill_data = abap_true
    IMPORTING
        et_data = lt_app_root).
```
LOOP AT lt_app_root ASSIGNING <ls_app_root>.

* determine the license ID
  lv_license_id = det_license( i_driver = <ls_app_root>-driver ).
  IF lv_license_id <> <ls_app_root>-zz_driver_license_id.
    CREATE DATA rs_app_data.
    * first take all previous data
    CLEAR rs_app_data->*.
    MOVE-CORRESPONDING <ls_app_root> TO rs_app_data->*.
    * take determined license ID
    rs_app_data->zz_driver_license_id = lv_license_id.
  ENDIF.

  * create modify table for appointments
    ls_app_mod-node = /scwm/if_dsapp_c=>sc_node-root.
    ls_app_mod-change_mode = 'U'. "update
    ls_app_mod-node_cat = /scwm/if_dsapp_c=>sc_node_category-root-root.
    ls_app_mod-key = <ls_app_root>-key.
    ls_app_mod-data = rs_app_data.
    APPEND ls_app_mod TO lt_app_mod.
  ENDIF.

  ENDLOOP.

  * update all changes
  IF lt_app_mod IS NOT INITIAL.
    CALL METHOD io_modify->do_modify
      EXPORTING
        it_modification = lt_app_mod.
  ENDIF.

ENDMETHOD.
4.2 User Interface Enhancements

Some general remarks for creating user interface enhancements:

- Basic enhancements ideally can be done without any coding. For more complex user interfaces and enhancements, coding might be required. The focus of this document version is the enhancements that can be done with configuration only.

- Each SAP Dock Appointment Scheduling user interface is build up from User Interface Building Blocks (UIBBs). Each of these building blocks has a configuration that can be adapted by partners and customers.

- The standard configurations remain untouched. When adapting a configuration, the system creates a corresponding customizing record, which contains your enhancements. Adapted configurations can be created in a development system, for example, and get transported to a test or production system. The client of the system where you make the UI enhancements must be set up in such a way that it allows transporting configurations.

- The adaptations can also be deleted again. After deleting, for example, an adapted configuration, the original standard configuration is used again for processing the corresponding user interface.

- Before you can adapt any user interface configuration, you need to set the parameter &sapid-config-mode=X in the URL of the user interface to be enhanced.
4.2.1 Add additional attribute to a FORM UIBB

A very basic and common example for extending the user interface is adding additional fields on an identified building block that carries this additional information.

**Example**

The following example shows how to add an extension field onto the tab *Planning Data*

1) Start the *Direct Access to Appointment - Textual UI* (application /SCWM/DSAPP_MAINT) for displaying appointments in your browser from the SAP user menu.

![Picture 23: Call application Direct Access to Appointment - Textual in the browser](image)

2) At the end of the URL, enter the following additional parameter: \&Change_mode=C. Now the maintenance UIBB is displayed

![Picture 24: Add parameter \&Change_mode=C at the end of the URL](image)
3) At the end of the URL, enter the following additional parameter: &sap-config-mode=X to enable the adaptation of configurations related to this user interface.

4) Click Show Customizable Areas.

When starting the configuration editor for a specific UIBB configuration that has not yet been enhanced, the system always asks you to create a customizing data record. In the first dialog box, just click on OK. In the second dialog box, enter a description for the customizing data record and click on OK.
Picture 27: Create a new component customizing

The name of the standard configuration of the General Data tab page for this example is:

<SCWM/WDCC_DSAPP_T_PLAN_TRANS>
5) The system starts the configuration editor.

Picture 28: Configuration editor of a UIBB

Switch into edit mode.

Close the *General Settings* pane and the *Preview* pane, and open the *Form UIBB Schema* pane.
Click the Add Element button and choose Add Element at Current Level. In the screen, you see a list with the available attributes. At the end of the list, you see the new enhancement fields defined in chapter 4.1.3.
Select the required attribute (for example: `<Z_Z_DRIVER_LICENSE_ID>`) and click OK.

Open the preview pane again to move the new attribute to the desired position on the screen using drag & drop.

After moving the attribute to the correct position, you can change the label text, for example `<Driver License ID>`:
Picture 32: New attribute “Driver License ID” from customer's enhancement in the screen

Save your configuration. The enhancement can now be tested by starting the application again.

The new attribute should be visible and the additional data can be maintained:
New attribute “Driver License ID” from customer’s enhancement in the screen
4.2.2 Add additional attribute to a List View

**Example**

The following example shows how to add an extension field onto the Result List of the *Maintain Appointments – Textual UI*.

1) Start the *Maintain Appointments – Textual UI* (application /SCWM/DSAPP_LIST) for displaying appointments in your browser from the SAP user menu.

2) At the end of the URL, enter the following additional parameter: `&sap-config-mode=X` to enable the adaptation of configurations related to this user interface. In the customizing mode, click the *Show Customizable Areas* button.

3) The component configurator of the list UIBB is displayed. Close the *Preview* pane and the *General Settings* pane and open the *List UIBB Schema*.

Switch to edit mode.
4) Click the *Add Column* button. From the list of available columns, choose the attribute you want to add to the list and click *OK*.

![List of attribute in the UIBB to the option to add new elements](image)

**Picture 35**: List of attribute in the UIBB to the option to add new elements

![List of available attributes to shown in the list UIBB](image)

**Picture 36**: List of available attributes to shown in the list UIBB
5) Change the sequence of attributes in the list by using the *Up* and *Down* buttons. Change the *Header* attribute to change the label, if required. Save the changes.

The enhancement can now be tested by starting the application again.
The new attribute should be visible in the result list:

<table>
<thead>
<tr>
<th>Country Reference</th>
<th>Carrier Reference</th>
<th>Start Date</th>
<th>Start Time</th>
<th>Finish Date</th>
<th>Finish Time</th>
<th>Appointment Date</th>
<th>Status</th>
<th>Carrier</th>
<th>Partner</th>
<th>Driver</th>
<th>License ID</th>
<th>Mode of Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_TEST_01</td>
<td>LP_TEST_01</td>
<td>12.11.2022</td>
<td>08:15</td>
<td>12.11.2022</td>
<td>09:15</td>
<td>2008094236</td>
<td>planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL_TEST_01</td>
<td>LP_TEST_01</td>
<td>12.11.2022</td>
<td>02:05</td>
<td>12.11.2022</td>
<td>12:00</td>
<td>2008094230</td>
<td>planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL_TEST_01</td>
<td>LP_TEST_01</td>
<td>12.11.2022</td>
<td>08:45</td>
<td>12.11.2022</td>
<td>11:45</td>
<td>2008094232</td>
<td>planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL_TEST_01</td>
<td>LP_TEST_01</td>
<td>12.11.2022</td>
<td>10:25</td>
<td>12.11.2022</td>
<td>11:30</td>
<td>2008094235</td>
<td>planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New attribute is shown in the list.
4.3 Recommended Enhancements/Best-Practice

In this chapter, we describe enhancements that we recommend creating, for example, for performance reasons.

4.3.1 Read Deliveries of the Transportation Unit Assigned to an Appointment

If you want to get information about the deliveries assigned to a transportation unit which is related to an appointment you should use methods of class </SCWM/CL_SR_TUDLV >.

This example coding shows how to access the transportation unit and determine the assigned deliveries.

DATA: ls_selection TYPE /scmb/s_sp_selection.
DATA: ls_exclude TYPE /scwm/s_sr_query_excl.
DATA: lo_tu_query TYPE REF TO /scwm/cl_sr_tu_query.
DATA: lt_bo_tu TYPE /scwm/tt_bo_tunit.
DATA: lo_sr_bom TYPE REF TO /scwm/cl_sr_bom.
DATA: lo_log TYPE REF TO /scwm/cl_log.
DATA: lo_tudlv_manager TYPE REF TO /scwm/cl_sr_tudlv.
DATA: ls_sel_das_docno TYPE /scwm/s_sel_das_docno.
DATA: lv_dsapp_docno TYPE /scwm/dsappid.
DATA: lt_docid TYPE /scwm/dlv_doccat_docid_tab.
DATA: ls_tu_act_key TYPE /scwm/s_tu_sr_act_num.
DATA: lt_tu_act_key TYPE /scwm/tt_tu_sr_act_num.

FIELD-SYMBOLS: <ls_bo_tu> TYPE /scwm/s_bo_tunit.
FIELD-SYMBOLS: <ls_tu_act_key> TYPE /scwm/s_tu_sr_act_num.

lo_sr_bom = /scwm/cl_sr_bom->get_instance( ).
lo_log = lo_sr_bom->get_log( ).
lo_log->init( ).

CREATE OBJECT lo_tu_query
  EXPORTING
    io_log = lo_log.

* search for TU of the appointment
lv_dsapp_docno = '6500004914'.

ls_selection-sign = wmgc_sign_inclusive.
ls_selection-option = wmgc_option_eq.
ls_selection-low = lv_dsapp_docno.

MOVE-CORRESPONDING ls_selection TO ls_sel_das_docno.
lo_tu_query->add_das_docno( is_sel_das_docno = ls_sel_das_docno ).

CLEAR ls_exclude.
lo_sr_bom->query(
   EXPORTING
   - io_tu_query = lo_tu_query
   - is_exclude  = ls_exclude
   IMPORTING
   - et_bo_tu    = lt_bo_tu
).

READ TABLE lt_bo_tu ASSIGNING <ls_bo_tu> INDEX 1.

IF sy-subrc = 0.
   ls_tu_act_key-tu_num   = <ls_bo_tu>-tu_num.
   ls_tu_act_key-tu_sr_act_num = <ls_bo_tu>-tu_sr_act_num.
   APPEND ls_tu_act_key TO lt_tu_act_key.
ENDIF.

* Preparation to get deliveries assigned to the TU
lo_tudlv_manager = /scwm/cl_sr_tudlv=>get_instance( ).
lo_tudlv_manager->setup( it_tu_act_key = lt_tu_act_key ).

* Get all assigned delivery headers of the TU
lo_tudlv_manager->get_dlvh_by_tu(
   EXPORTING
   - is_tu_act_key  = ls_tu_act_key
   IMPORTING
   - et_docid   = lt_docid ).

1) Reassure that you call the query of class </scwm/cl_sr_bom> to select the transportation always with defined selection criteria; otherwise you may read all transportation unit of the system
2) Class </SCWM/CL_SR_TUDLV > also contains methods to get information about the assigned delivery items or handling units.