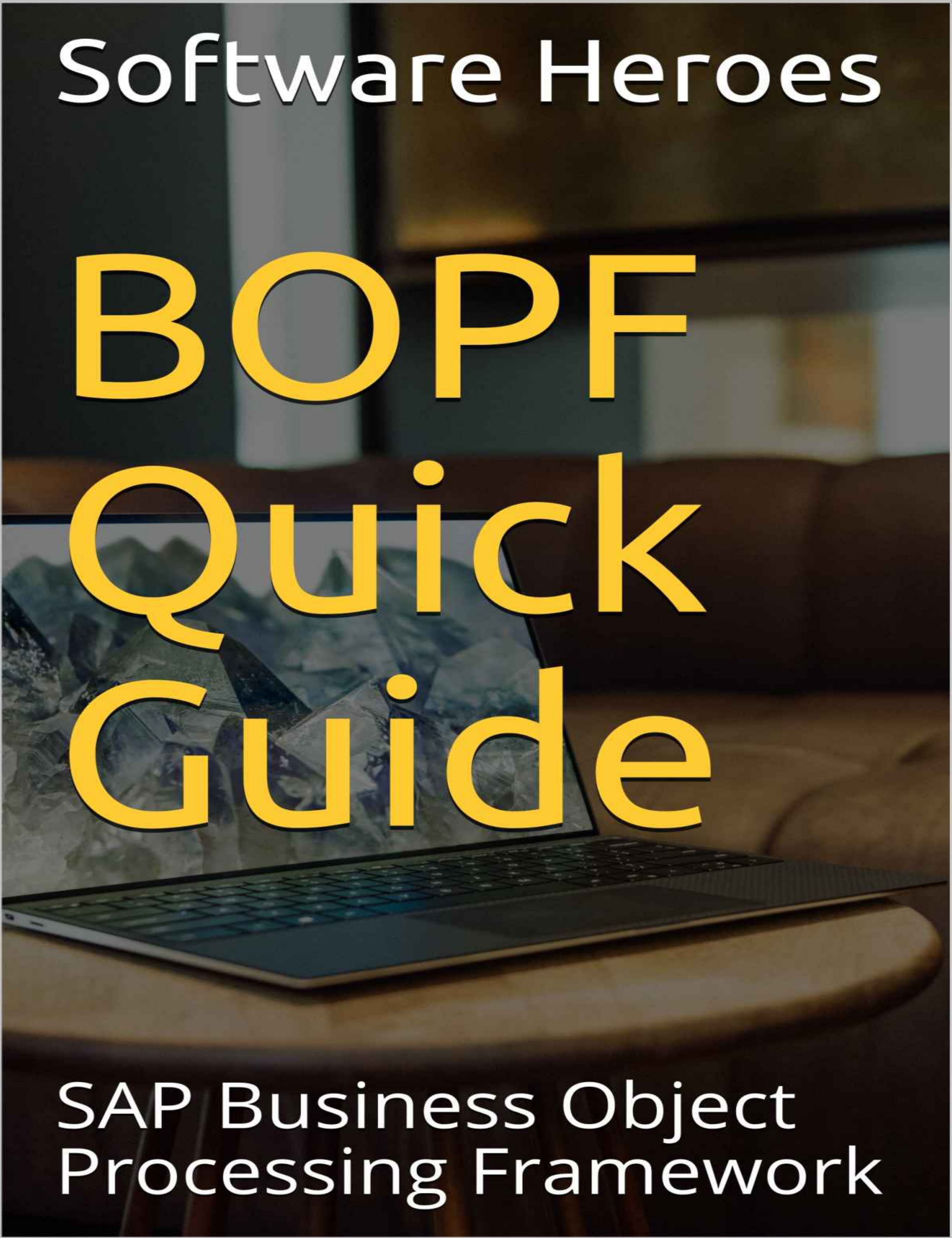
Software Heroes

BOPF Quick Guide

SAP Business Object Processing Framework



BOPF - Quick Guide

Software-Heroes

Table of content

General

Foreword

Introduction

BOPF

Tools

Structure

Define a node

Data model

Test data

Queries

Determination

Validation

Action

Manager

Authorization

Help

Interfaces

Custom query

Helper methods

Performance

Eclipse

End

Ending words

Foreword

First of all, I want to thank you for purchasing this e-book. This gives you the key to get started with BOPF quickly and learn the most important aspects of the framework in a short time. In the following chapters you will find general information, but also more in-depth knowledge to implement your first data models.

It is our goal that after reading the book you will have all the essential tools available to master BOPF and not to waste too much time on deep insights. In any case, you will only collect most of the practice during application and development.

Have fun working through the chapters and learning the framework!

Introduction

What does BOPF stand for? How can you use it successfully and effectively? What is behind it? These questions we want to clarify in this small book around the topic BOPF and bring you the benefits of the framework something closer.

Definition

BOPF stands for "Business Object Processing Framework" and is a framework written by SAP for processing and maintaining related data. The framework was written completely object-oriented, so that the access to the data is only possible through the provided classes and methods.

Availability

BOPF has been officially released by SAP since Release 7.4 and comes standard with no additional activated extensions. It does not matter which database is in the system, the framework works with and without a HANA database. If you own a HANA system, then there are a few minor processing steps.

Usage

What do you need the framework for, and where can it support you? In fact, using complex data models can help you keep your data in more than one database table.

A simple example of a business object is, for example, the FI document. There is a document header (BKPF) and the line items (BSEG), but also other items for taxes (BSET) or the auxiliary indices (BSIK, BSAK, etc.). All data belong together, if you delete the document header, all positions and entries in the other tables should also be deleted. Only in this way does the database always remain completely consistent.

Advantages

- Time and cost savings
- Standard access to all data
- Separation of business logic and UI logic
- Central maintenance of the object
- Easy expandability

Disadvantages

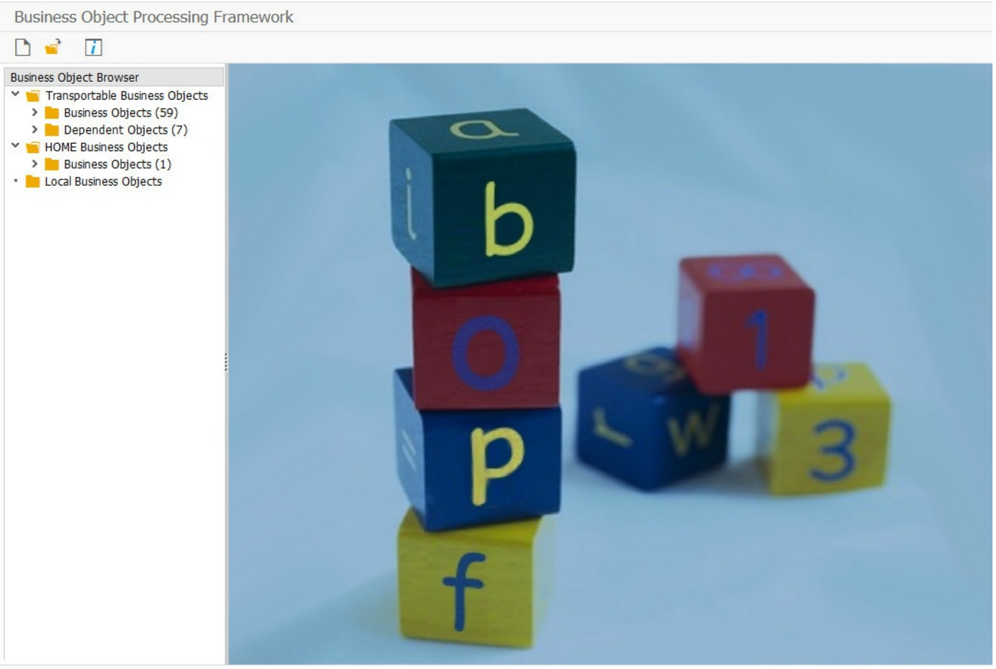
- Knowledge and understanding of the technology (design pattern, OOP)
- Compliance with the defined purpose of use

Tools

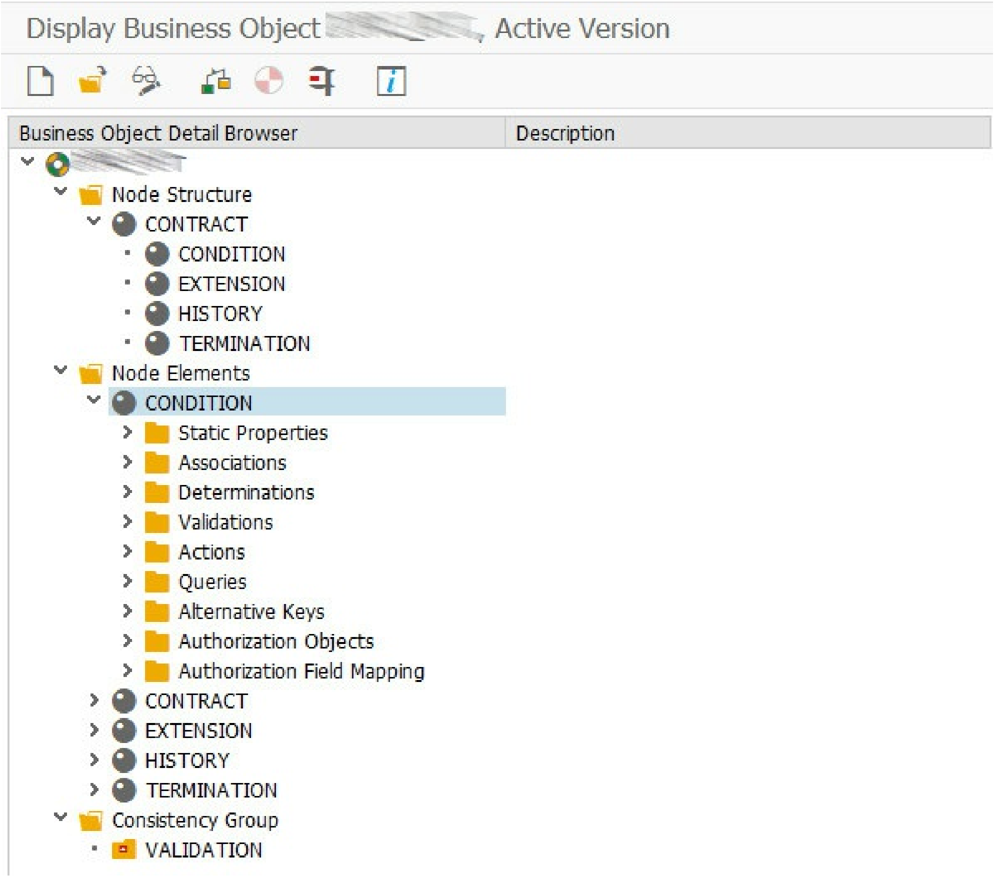
There are special tools in the system for creating and processing the business objects. Basically, the access, in addition to the standard transactions, still includes the definition tool (BOBX) and the test tool (BOBT). In this chapter we want to briefly explain the function and the transactions.

Definition (BOBX)

At this point you will get an overview of all objects in the system in the first step. The folders are structured according to SAP objects, the home and local objects in the system. Customer objects that are transported are created in the home folder. Customer objects that are not transported are available in the local folder and clearly demarcated.



With the forward navigation you will then land in the detail screen. The entire data model is created in this tool. Here you can look at the individual functions, structures, classes and other dependencies.



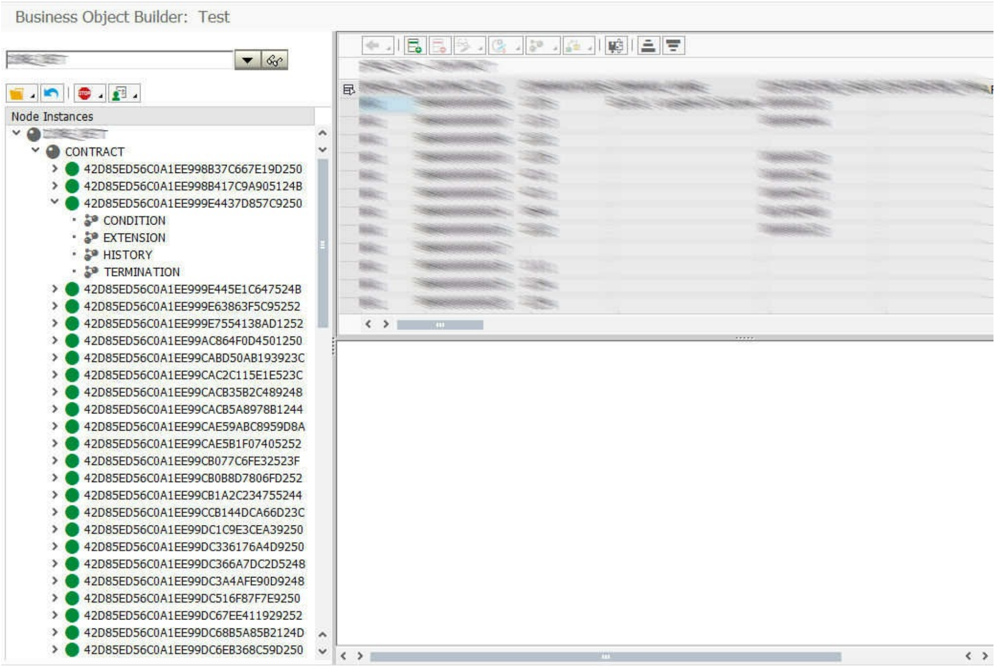
Display Business Object [redacted], Active Version

Business Object Detail Browser	Description

- [redacted]
  - Node Structure
    - CONTRACT
      - CONDITION
      - EXTENSION
      - HISTORY
      - TERMINATION
  - Node Elements
    - CONDITION
      - Static Properties
      - Associations
      - Determinations
      - Validations
      - Actions
      - Queries
      - Alternative Keys
      - Authorization Objects
      - Authorization Field Mapping
    - CONTRACT
    - EXTENSION
    - HISTORY
    - TERMINATION
  - Consistency Group
    - VALIDATION

Test (BOBT)

The test cockpit is used to test all functions of the business object and thus includes all possible functions that are normally possible. With appropriate authorizations, productive data can also be adapted. You should therefore pay attention to which functions you use in production.



- CONTRACT
  - 42D85ED56C0A1EE998B37C667E19D250
  - 42D85ED56C0A1EE998B417C9A905124B
  - 42D85ED56C0A1EE999E4437D857C9250
    - CONDITION
    - EXTENSION
    - HISTORY
    - TERMINATION
  - 42D85ED56C0A1EE999E445E1C647524B
  - 42D85ED56C0A1EE999E63863F5C95252
  - 42D85ED56C0A1EE999E7554138AD1252
  - 42D85ED56C0A1EE99AC864F0D4501250
  - 42D85ED56C0A1EE99CABD50AB193923C
  - 42D85ED56C0A1EE99CAC2C115E1E523C
  - 42D85ED56C0A1EE99CACB35B2C489248
  - 42D85ED56C0A1EE99CACB5A8978B1244
  - 42D85ED56C0A1EE99CAE59ABC8959D8A
  - 42D85ED56C0A1EE99CAE5B1F07405252
  - 42D85ED56C0A1EE99CB077C6FE32523F
  - 42D85ED56C0A1EE99CB0B8D7806FD252
  - 42D85ED56C0A1EE99CB1A2C234755244
  - 42D85ED56C0A1EE99CCB144DCA66D23C
  - 42D85ED56C0A1EE99DC1C9E3CEA39250
  - 42D85ED56C0A1EE99DC336176A4D9250
  - 42D85ED56C0A1EE99DC366A7DC2D5248
  - 42D85ED56C0A1EE99DC3A4AFE90D9248
  - 42D85ED56C0A1EE99DC516F87F7E9250
  - 42D85ED56C0A1EE99DC67EE411929252
  - 42D85ED56C0A1EE99DC68B5A85B2124D
  - 42D85ED56C0A1EE99DC6EB368C59D250

In the first step you select the BOPF over the upper area, which you want to test. When exiting BOBX, the data model is already preassigned. Initially, no data is loaded from the tables, but must first be read via the queries. These can be found in the left menu under the "Folder" icon.

The toolbar above the right image is for editing, setting locks, executing actions and navigate to sub nodes. Here, the majority of editing functions can be performed.

Hint: All data that is changed and saved here are stored in the database. This also applies to productive data (if corresponding authorizations exist).

Editing (SE11, SE24)

Other important tools for you are the SE11 for editing the tables and structures, as well as the SE24 for editing the classes in the system. The tools do not have to be used, but can also be called up via forward navigation from transaction BOBX.

Eclipse

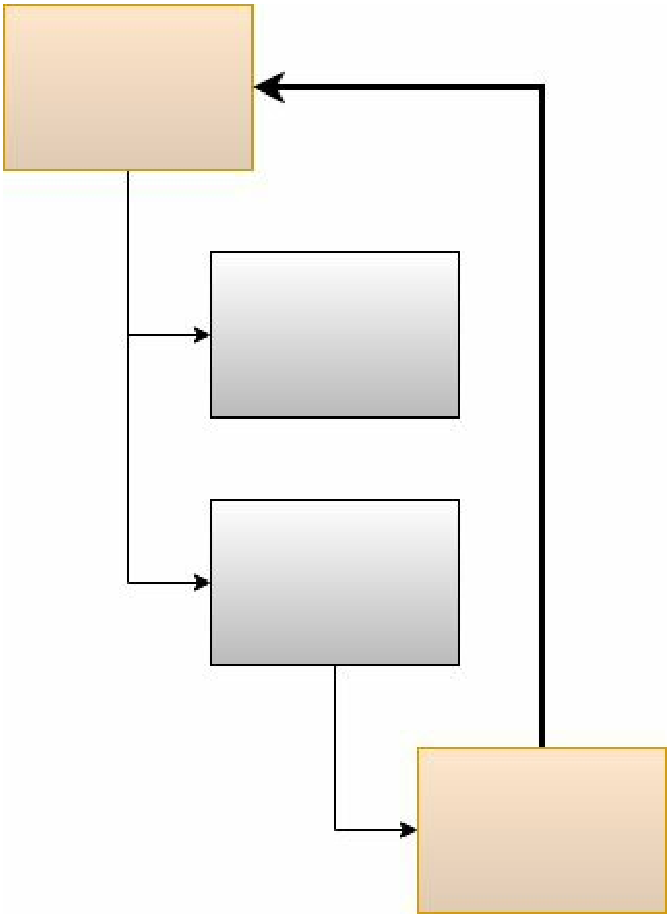
Of course, editing the model is also possible via Eclipse if you prefer this development environment. Most of the time we will be in the area of SAP GUI, but later we will show you the navigation in Eclipse and the ADTs a bit more.

Structure

When defining the structure of the business object, some minor points should be noted, but they are very important for the object to work. In this chapter we want to go into the structure and the way it works and what the object brings with it.

Structure

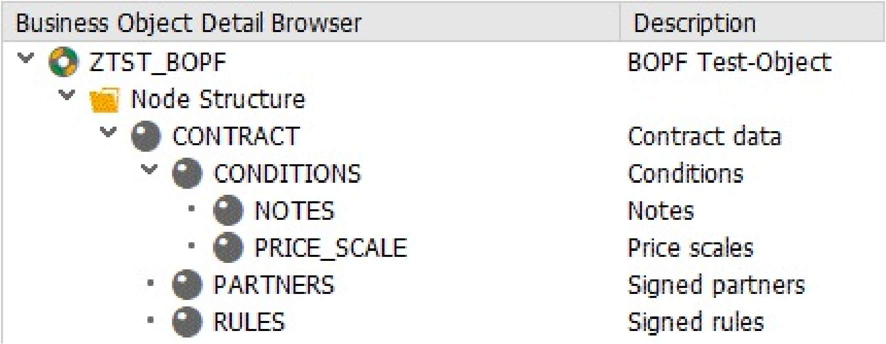
The structure of the object always begins with a root node. This node is the central component, the header data, of the business object. We will create a data model in the next series. There can be only one root node in the entire model, as it has to be unique.



All other nodes below the root are referred to as sub nodes and are connected via relationships to the respective top node. In addition to this relationship, there is always a connection to the root node, no matter how deep you are in the hierarchy.

Data model

The data model has the contract as the root node. There are the relevant headers and simple relationships to data such as creator, partner, name and contract ID for example. Under the contract, the terms of the contract, so what is the signatory, the individual partners or parties that conclude the contract and certain rules to be observed. Under the conditions, there is also other data, such as notes and price ranges, which may be related to the terms.

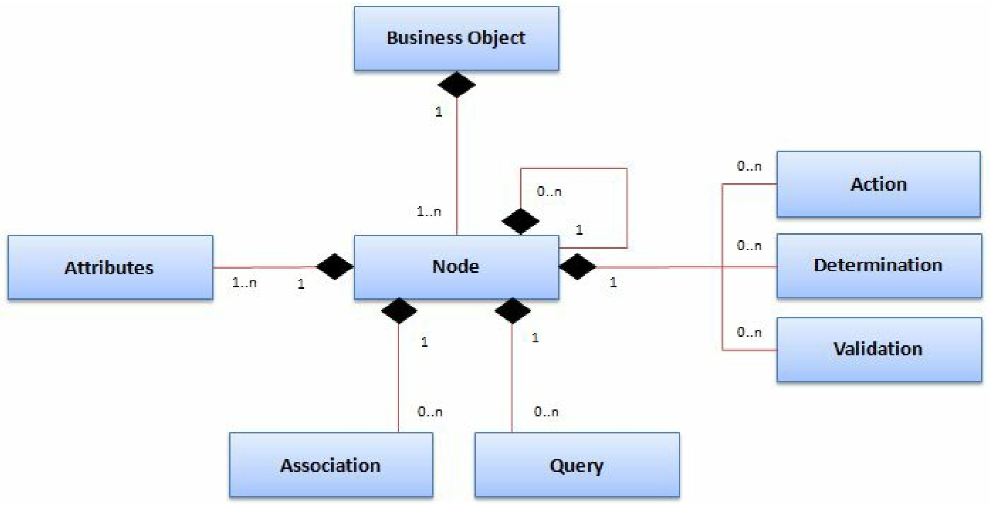


Business Object Detail Browser	Description
ZTST_BOPF	BOPF Test-Object
Node Structure	
CONTRACT	Contract data
CONDITIONS	Conditions
NOTES	Notes
PRICE_SCALE	Price scales
PARTNERS	Signed partners
RULES	Signed rules

The structure and composition of the individual nodes, we will introduce to you in one of the coming chapters and jointly create the structures and databases.

Functions

In the following picture, which comes from transaction BOB, all actions and possibilities on a node in the BOPF are displayed in a compact way. All these actions are needed to be able to work well with your model and to get the best out of it. One or more nodes belong to a business object.



Explanation:

- Attribute - Each node has different attributes, structures, tables
- Association - There are defined relationships to top nodes or subnodes in the tree
- Query - A defined interface for retrieving data from the model
- Action - Performs an action on the model. This is usually associated with changes to the data
- Determination - Are passive data collection, read routines and additions to the data in the model
- Validation - Checking the data for consistency and completeness for various defined actions

Define a node

In the last chapter, we showed you how a structure can look like out of the various nodes in a BOPF, how they are connected and what you can do with them. In this chapter we want to create a node, in this case the root node of our model.

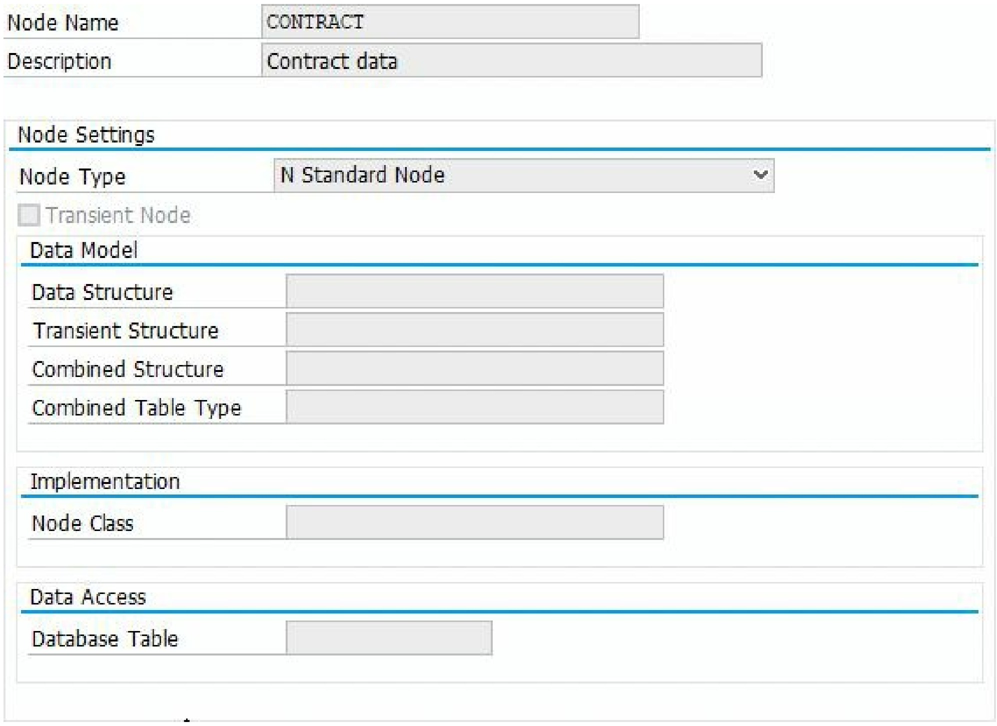
The root

Everything starts in the data model with the root node and the actual header data. After creating a new model, a root node is created immediately as it is needed. By default, the root node is called ROOT, but can be customized and renamed at any time to fit your own data structure.

Terms

The data model consists of different structures, table types, and tables. Here is a short explanation of the terms that will fall over and over again:

- Data structure - Corresponds to the data structure that is stored on the database at the end, here all data fields to be stored are defined.
- Transient structure - Includes fields that are read during runtime or selection. These fields are not backed up to the database and must be populated manually. These are mostly fields that result from existing master data.
- Combined Structure - As the name implies, this is a structure that links persistent and transient data together and makes them available to the user.
- Combined table type - Is the table type of the combined structure.
- Database table - The name of the database under which the persistent data is stored.



Node Name	CONTRACT
Description	Contract data

Node Settings

Node Type	N Standard Node

☐ Transient Node

Data Model

Data Structure	
Transient Structure	
Combined Structure	
Combined Table Type	

Implementation

Node Class	

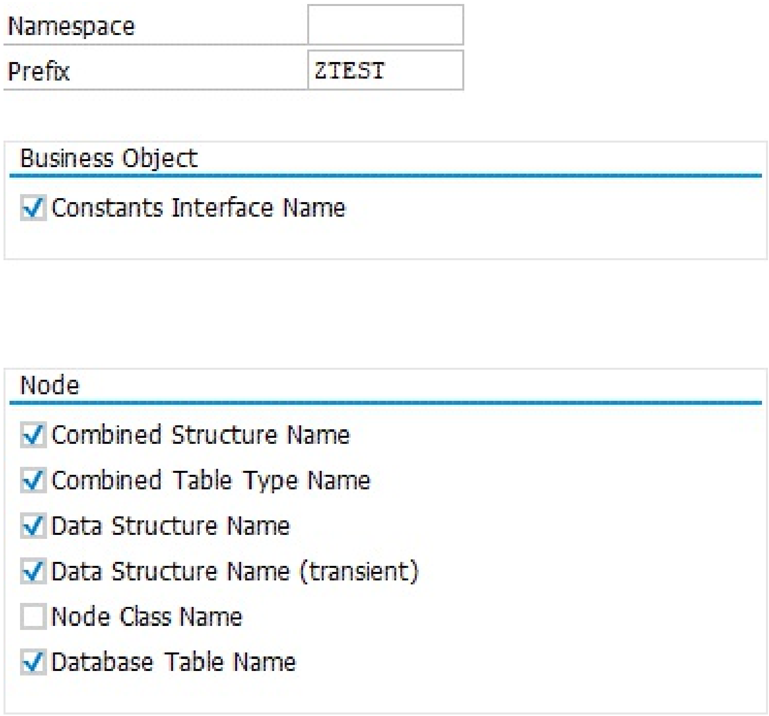
Data Access

Database Table	

Definition

Using the example of the root node, we want to perform the installation for you and go through the individual steps. In addition to the header data stored on the database, we also want to take over a field in the transient data, which should be read at runtime.

In the first step we have to define the names of the individual structures and table types. SAP also offers a nice feature that makes us think. Via the menu item "Extras -> Propose repository name" you will get a popup, which you can use to generate all names. Mark the checkboxes as below in the picture to suggest all necessary names.



Namespace	
Prefix	ZTEST

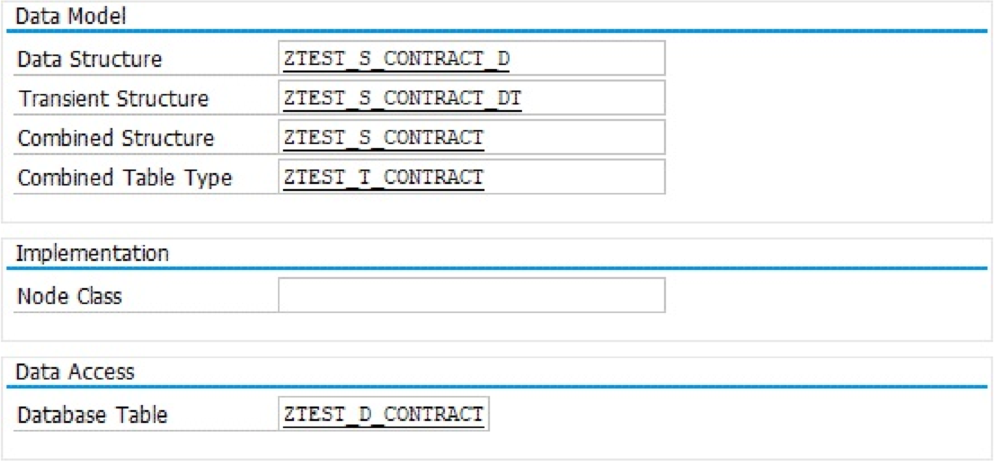
Business Object

☑ Constants Interface Name

Node

☑ Combined Structure Name
☑ Combined Table Type Name
☑ Data Structure Name
☑ Data Structure Name (transient)
□ Node Class Name
☑ Database Table Name

After confirming the popup you will receive the suggestions for all structures.



Data Model

Data Structure	ZTEST_S_CONTRACT_D
Transient Structure	ZTEST_S_CONTRACT_DT
Combined Structure	ZTEST_S_CONTRACT
Combined Table Type	ZTEST_T_CONTRACT

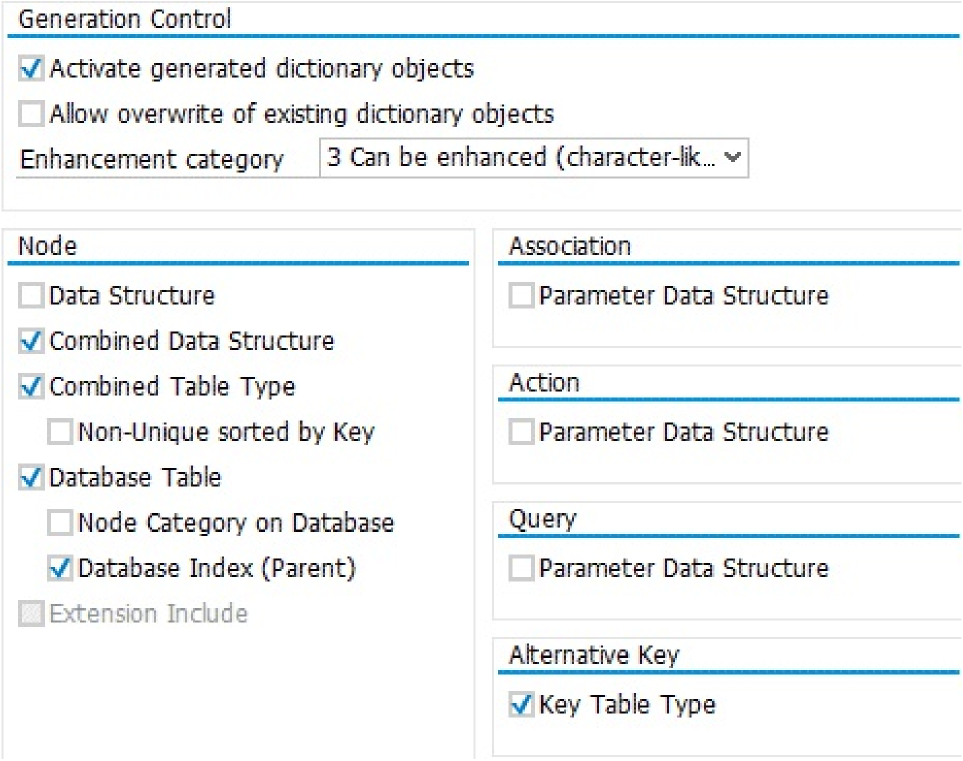
Implementation

Node Class	

Data Access

Database Table	ZTEST_D_CONTRACT

By double-clicking navigation, we create in the first step the data structure that should hold the data in the database. For the example we use a contract number, the creditor and the creation date. In the transient structure we want to read the name of the creditor from the master data. After activating the two structures you only need to activate the data type and the database table. Also, for this SAP has already deposited a function. Again with "Extras -> Generate Dictionary Elements" the following pop-up will be displayed.



Generation Control

☑ Activate generated dictionary objects

□ Allow overwrite of existing dictionary objects

Enhancement category: 3 Can be enhanced (character-lik...

Node

☑ Combined Data Structure

□ Data Structure

☑ Combined Table Type

□ Non-Unique sorted by Key

☑ Database Table

□ Node Category on Database

☑ Database Index (Parent)

□ Extension Include

Association

□ Parameter Data Structure

Action

□ Parameter Data Structure

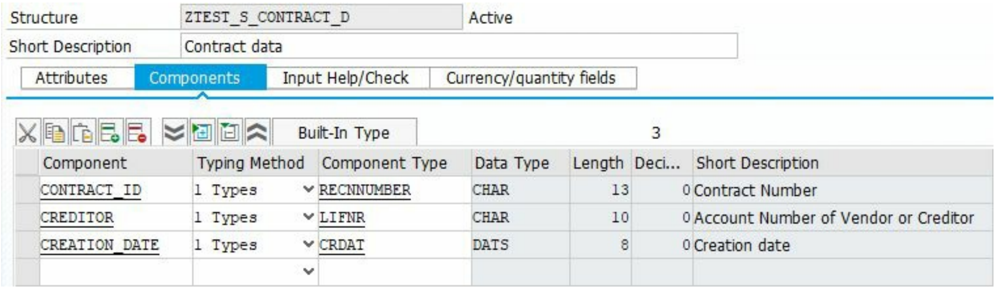
Query

□ Parameter Data Structure

Alternative Key

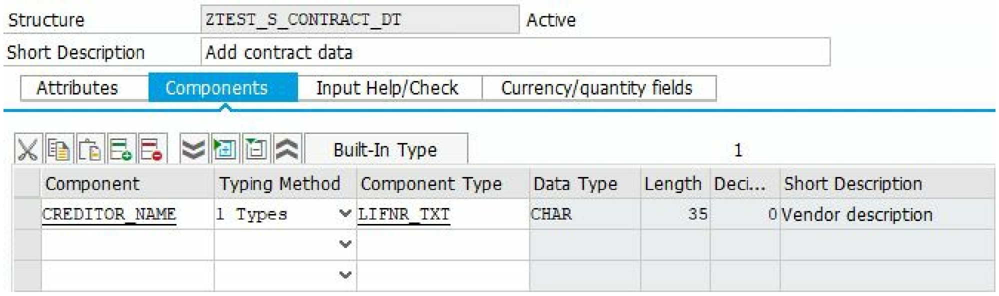
☑ Key Table Type

The checkboxes "combined structure" and "table type" should be activated, as well as the database. This will automatically create the last objects and we will not have any work left. The two structures and the finished database table can be found in the following pictures.



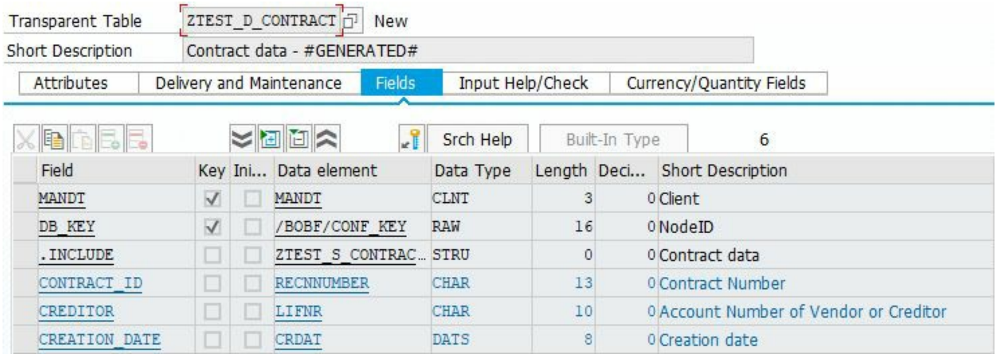
Structure: ZTEST_S_CONTRACT_D Active
Short Description: Contract data

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
CONTRACT_ID	1 Types	RECNNUMBER	CHAR	13	0	Contract Number
CREDITOR	1 Types	LIFNR	CHAR	10	0	Account Number of Vendor or Creditor
CREATION_DATE	1 Types	CRDAT	DATS	8	0	Creation date



Structure: ZTEST_S_CONTRACT_DT Active
Short Description: Add contract data

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
CREDITOR_NAME	1 Types	LIFNR_TXT	CHAR	35	0	Vendor description



Transparent Table: ZTEST_D_CONTRACT New
Short Description: Contract data - #GENERATED#

Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description
MANDT	☑	□	MANDT	CLNT	3	0	Client
DB_KEY	☑	□	/BOBF/CONF_KEY	RAW	16	0	NodeID
.INCLUDE	□	□	ZTEST_S_CONTRAC...	STRU	0	0	Contract data
CONTRACT_ID	□	□	RECNNUMBER	CHAR	13	0	Contract Number
CREDITOR	□	□	LIFNR	CHAR	10	0	Account Number of Vendor or Creditor
CREATION_DATE	□	□	CRDAT	DATS	8	0	Creation date

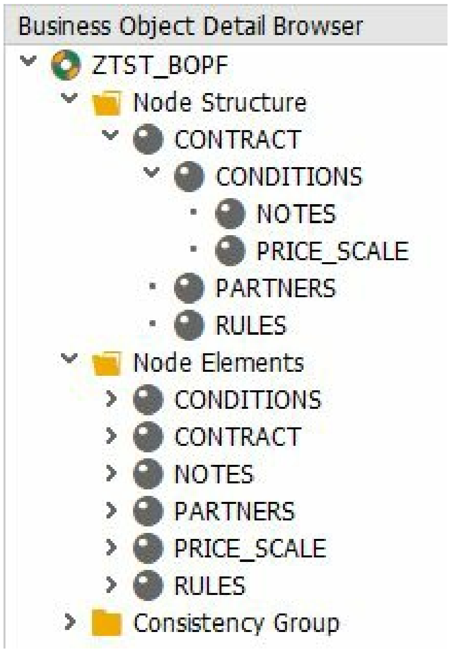
As you can see from the table, the client and the database key for the header table were automatically generated. The data structure taht has been created has no key in the initial state. In a later chapter we will create a real key for the data.

Data model

In the last chapter we defined a single node and also the different structures. Here, we would like to introduce you to the finished data model and how we defined the individual nodes.

Data model

The model consists of different levels of nodes and our root node is the contract that has different entities under it. Directly under the contract are the conditions, the partners and the rules/agreements. Under the various conditions there are still notes and discounts.

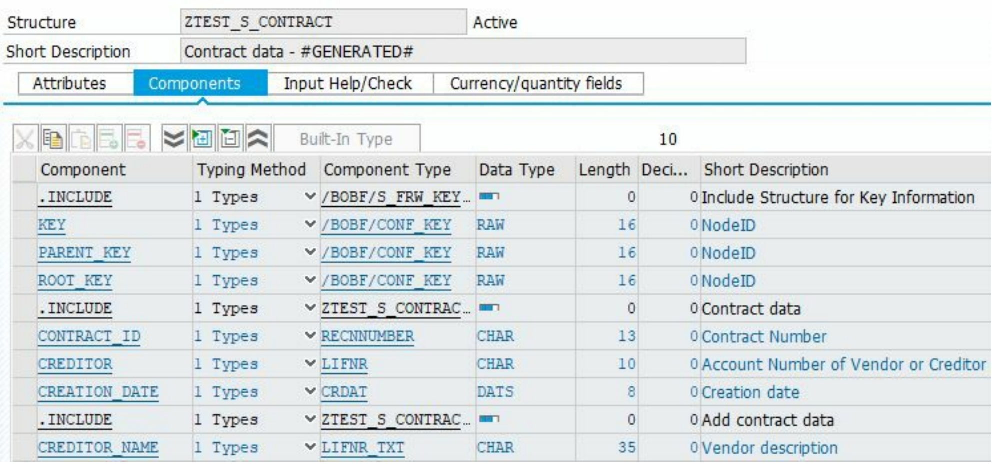


```
Business Object Detail Browser
⌄ ZTST_BOPF
  ⌄ Node Structure
    ⌄ CONTRACT
      ⌄ CONDITIONS
        · NOTES
        · PRICE_SCALE
      · PARTNERS
      · RULES
  ⌄ Node Elements
    › CONDITIONS
    › CONTRACT
    › NOTES
    › PARTNERS
    › PRICE_SCALE
    › RULES
  › Consistency Group
```

Tables

As you can already see in the description of the tables, they were automatically generated by the framework. The description can be adjusted at any time to meet your standard definitions. The first three fields of each table have the same key and other auxiliary fields that are included into the table through a structure.

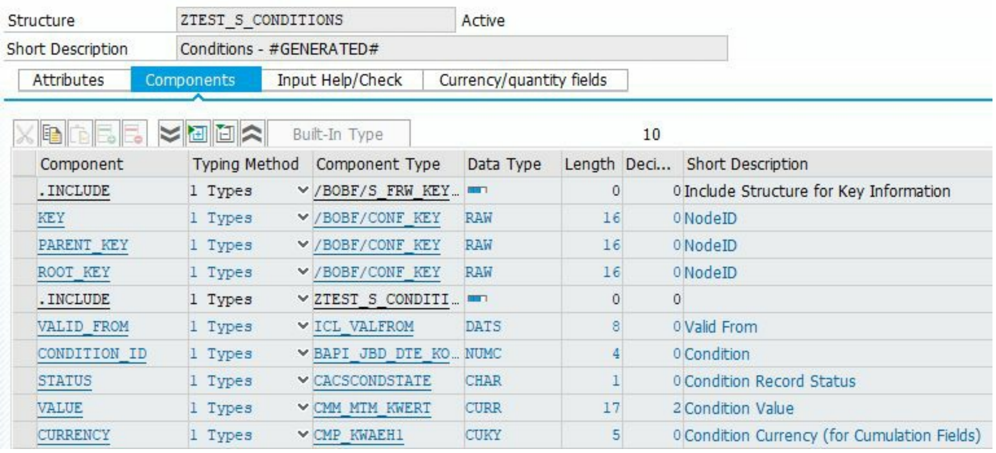
- KEY - Key of the entry in the current table
- PARENT_KEY - Key to the next higher level
- ROOT_KEY - Key of the root node



Structure ZTEST_S_CONTRACT Active

Short Description Contract data - #GENERATED#

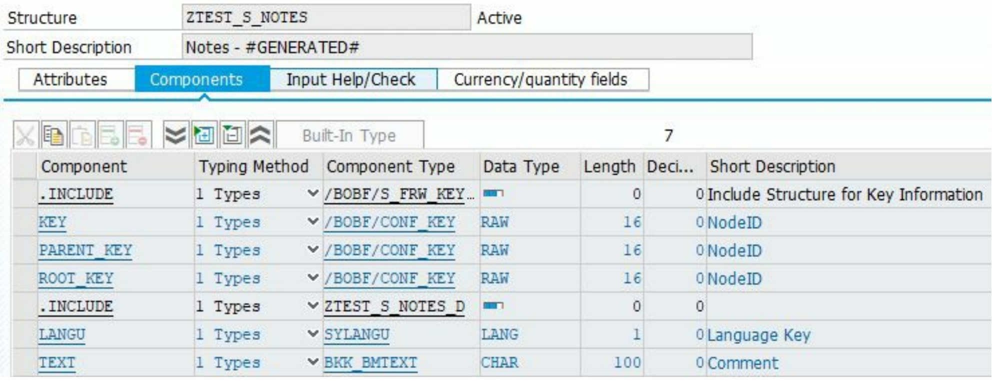
Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
.INCLUDE	1 Types	/BOBF/S_FRW_KEY...		0	0	Include Structure for Key Information
KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
PARENT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
ROOT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
.INCLUDE	1 Types	ZTEST_S_CONTRAC...		0	0	Contract data
CONTRACT_ID	1 Types	RECNNUMBER	CHAR	13	0	Contract Number
CREDITOR	1 Types	LIFNR	CHAR	10	0	Account Number of Vendor or Creditor
CREATION_DATE	1 Types	CRDAT	DATS	8	0	Creation date
.INCLUDE	1 Types	ZTEST_S_CONTRAC...		0	0	Add contract data
CREDITOR_NAME	1 Types	LIFNR_TXT	CHAR	35	0	Vendor description



Structure ZTEST_S_CONDITIONS Active

Short Description Conditions - #GENERATED#

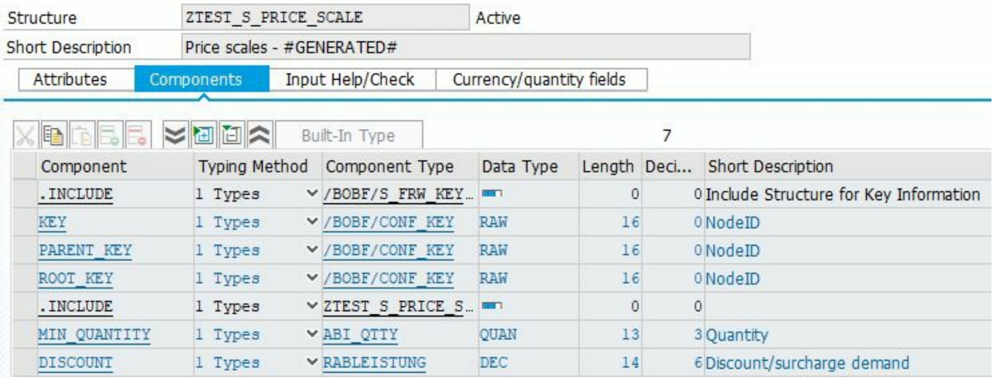
Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
.INCLUDE	1 Types	/BOBF/S_FRW_KEY...		0	0	Include Structure for Key Information
KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
PARENT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
ROOT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
.INCLUDE	1 Types	ZTEST_S_CONDITI...		0	0	
VALID_FROM	1 Types	ICL_VALFROM	DATS	8	0	Valid From
CONDITION_ID	1 Types	BAPI_JBD_DTE_KO...	NUMC	4	0	Condition
STATUS	1 Types	CACSCONDSTATE	CHAR	1	0	Condition Record Status
VALUE	1 Types	CMM_MTM_KWERT	CURR	17	2	Condition Value
CURRENCY	1 Types	CMP_KWAEH1	CUKY	5	0	Condition Currency (for Cumulation Fields)



Structure ZTEST_S_NOTES Active

Short Description Notes - #GENERATED#

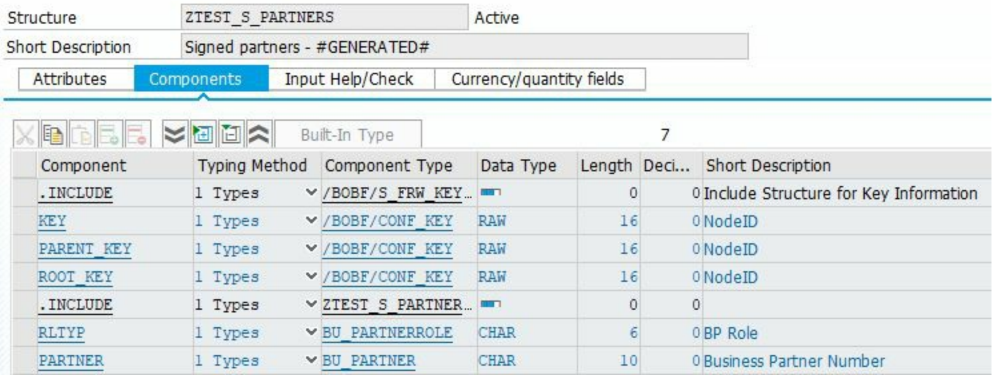
Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
.INCLUDE	1 Types	/BOBF/S_FRW_KEY...		0	0	Include Structure for Key Information
KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
PARENT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
ROOT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
.INCLUDE	1 Types	ZTEST_S_NOTES_D		0	0	
LANGU	1 Types	SYLANGU	LANG	1	0	Language Key
TEXT	1 Types	BKK_BMTEXT	CHAR	100	0	Comment



Structure ZTEST_S_PRICE_SCALE Active

Short Description Price scales - #GENERATED#

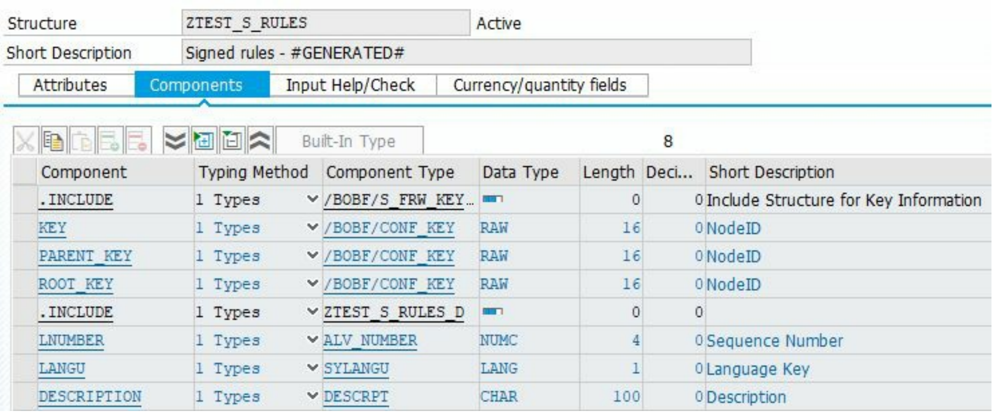
Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
.INCLUDE	1 Types	/BOBF/S_FRW_KEY...		0	0	Include Structure for Key Information
KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
PARENT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
ROOT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
.INCLUDE	1 Types	ZTEST_S_PRICE_S...		0	0	
MIN_QUANTITY	1 Types	ABI_QTTY	QUAN	13	3	Quantity
DISCOUNT	1 Types	RABLEISTUNG	DEC	14	6	Discount/surcharge demand



Structure ZTEST_S_PARTNERS Active

Short Description Signed partners - #GENERATED#

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
.INCLUDE	1 Types	/BOBF/S_FRW_KEY...		0	0	Include Structure for Key Information
KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
PARENT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
ROOT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
.INCLUDE	1 Types	ZTEST_S_PARTNER...		0	0	
RLTYP	1 Types	BU_PARTNERROLE	CHAR	6	0	BP Role
PARTNER	1 Types	BU_PARTNER	CHAR	10	0	Business Partner Number



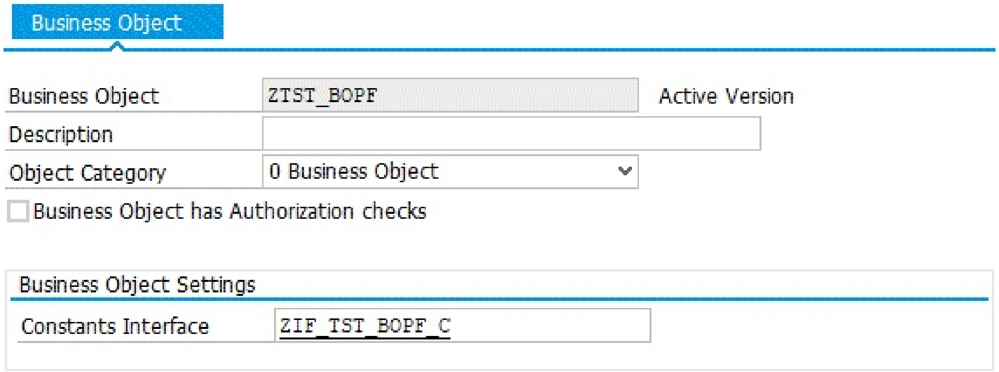
Structure: ZTEST_S_RULES Active

Short Description: Signed rules - #GENERATED#

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
.INCLUDE	1 Types	/BOBF/S_FRW_KEY...	▬	0	0	Include Structure for Key Information
KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
PARENT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
ROOT_KEY	1 Types	/BOBF/CONF_KEY	RAW	16	0	NodeID
.INCLUDE	1 Types	ZTEST_S_RULES_D	▬	0	0	
LNUMBER	1 Types	ALV_NUMBER	NUMC	4	0	Sequence Number
LANGU	1 Types	SYLANGU	LANG	1	0	Language Key
DESCRIPTION	1 Types	DESCRPT	CHAR	100	0	Description

Constants Interface

The interface ZIF_TST_BOPF_C contains all constants for access to the data model. The interface is regenerated every time the data model changes because it includes the fields as well as the structures and nodes as generated GUIDs. The interface is needed for all accesses to the data model and is therefore a central component of BOPF.



Business Object	ZTST_BOPF	Active Version
Description		
Object Category	0 Business Object	

□ Business Object has Authorization checks

Business Object Settings

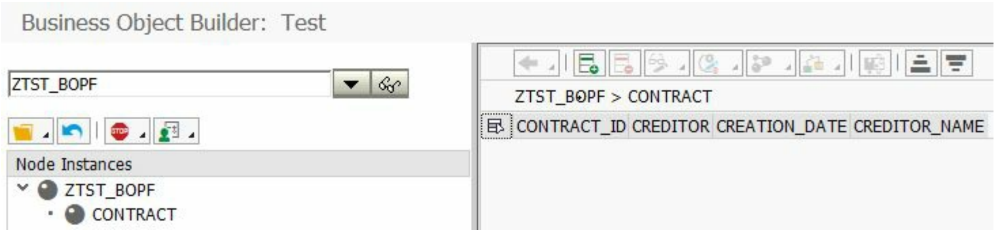
Constants Interface	ZIF_TST_BOPF_C

Test data

Creating, reading and deleting the data can easily be done via transaction BOBT. Getting the first records in the tables is no problem for you after this chapter.

BOBT

After calling the transaction, you usually get a blank screen and you have to choose your model first. With the arrow down you can select your object and you should get the following picture.



Business Object Builder: Test

ZTST_BOPF

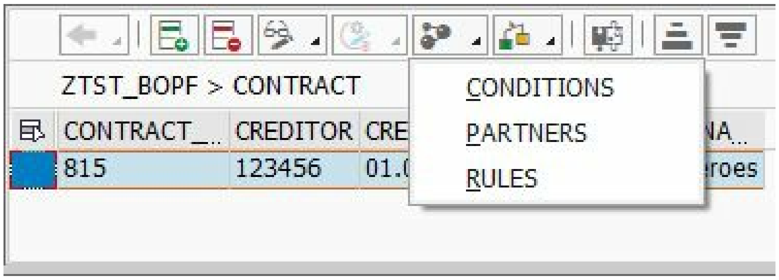
Node Instances

- ZTST_BOPF
  - CONTRACT

ZTST_BOPF > CONTRACT

CONTRACT_ID	CREDITOR	CREATION_DATE	CREDITOR_NAME

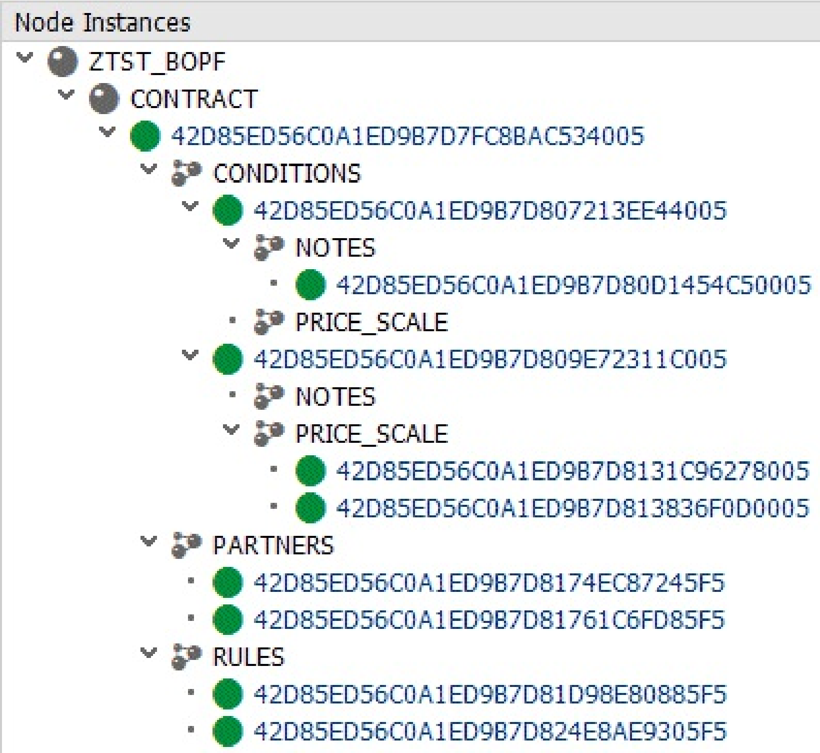
With a click on the white page button, you can create a new empty record and fill it with values. If you want to create an entity below this dataset, then you have to press the following button. Here are all nodes of the BOPF available, which are located under the current node. As soon as you have selected a node, you will get an empty table again, since no dataset has been created yet.



ZTST_BOPF > CONTRACT

CONTRACT_...	CREDITOR	CRE	NA...
815	123456	01.(	roes

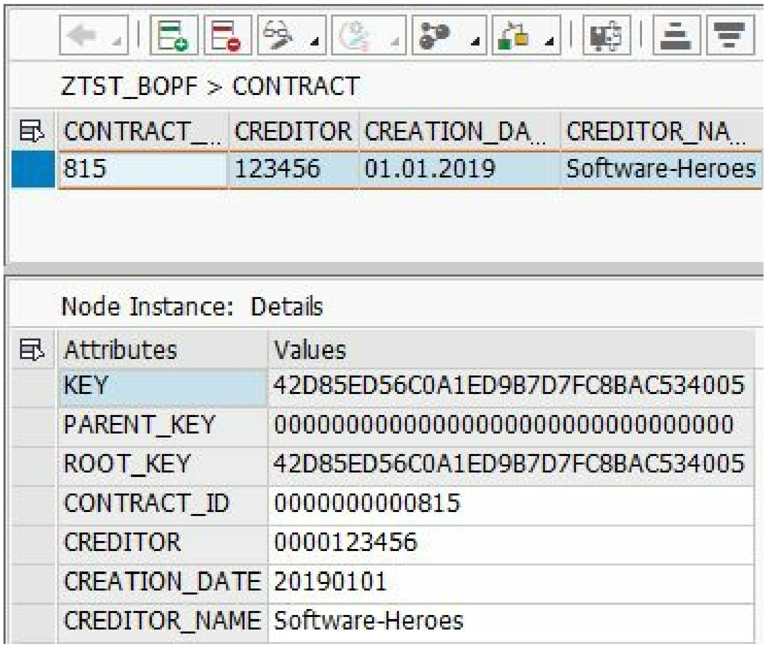
If you have created a contract, you can see the entire structure of nodes and sub nodes in the navigation tree.



```
Node Instances
ZTST_BOPF
  CONTRACT
    42D85ED56C0A1ED9B7D7FC8BAC534005
      CONDITIONS
        42D85ED56C0A1ED9B7D807213EE44005
          NOTES
            42D85ED56C0A1ED9B7D80D1454C50005
          PRICE_SCALE
        42D85ED56C0A1ED9B7D809E72311C005
          NOTES
          PRICE_SCALE
            42D85ED56C0A1ED9B7D8131C96278005
            42D85ED56C0A1ED9B7D813836F0D0005
      PARTNERS
        42D85ED56C0A1ED9B7D8174EC87245F5
        42D85ED56C0A1ED9B7D81761C6FD85F5
      RULES
        42D85ED56C0A1ED9B7D81D98E80885F5
        42D85ED56C0A1ED9B7D824E8AE9305F5
```

Data

As a summary of the entries made, we show you once again the test data we have created for the contract.

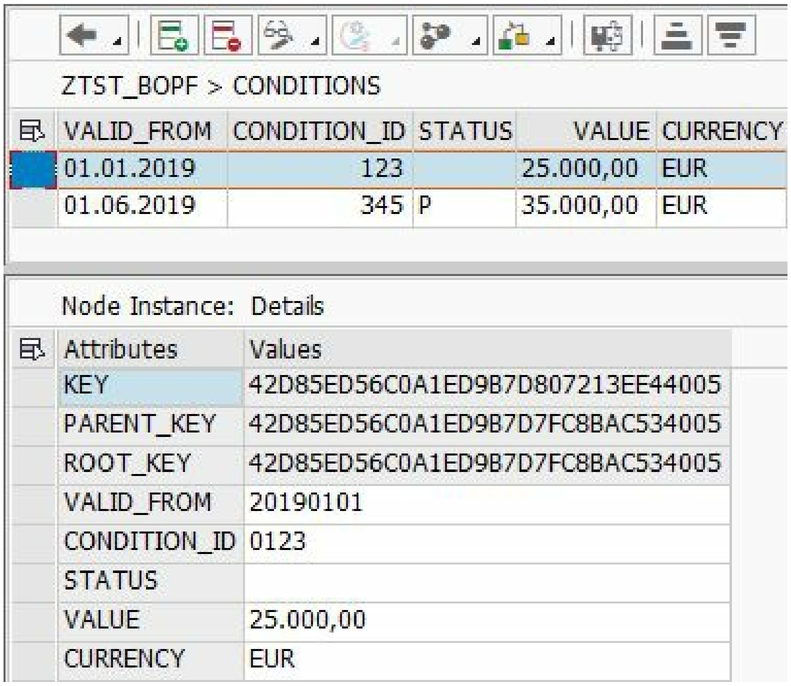


ZTST_BOPF > CONTRACT

CONTRACT_...	CREDITOR	CREATION_DA...	CREDITOR_NA...
815	123456	01.01.2019	Software-Heroes

Node Instance: Details

Attributes	Values
KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
PARENT_KEY	00000000000000000000000000000000
ROOT_KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
CONTRACT_ID	0000000000815
CREDITOR	0000123456
CREATION_DATE	20190101
CREDITOR_NAME	Software-Heroes

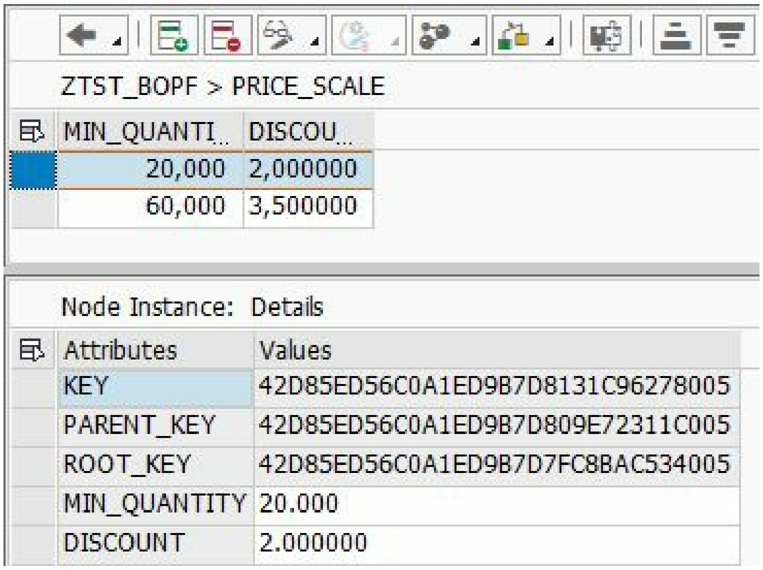


ZTST_BOPF > CONDITIONS

VALID_FROM	CONDITION_ID	STATUS	VALUE	CURRENCY
01.01.2019	123		25.000,00	EUR
01.06.2019	345	P	35.000,00	EUR

Node Instance: Details

Attributes	Values
KEY	42D85ED56C0A1ED9B7D807213EE44005
PARENT_KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
ROOT_KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
VALID_FROM	20190101
CONDITION_ID	0123
STATUS	
VALUE	25.000,00
CURRENCY	EUR

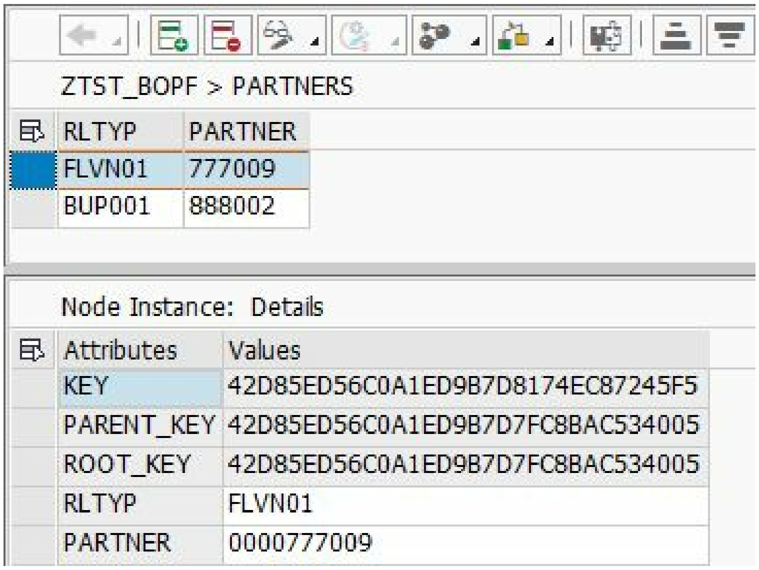


ZTST_BOPF > PRICE_SCALE

MIN_QUANTI…	DISCOU…
20,000	2,000000
60,000	3,500000

Node Instance: Details

Attributes	Values
KEY	42D85ED56C0A1ED9B7D8131C96278005
PARENT_KEY	42D85ED56C0A1ED9B7D809E72311C005
ROOT_KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
MIN_QUANTITY	20.000
DISCOUNT	2.000000

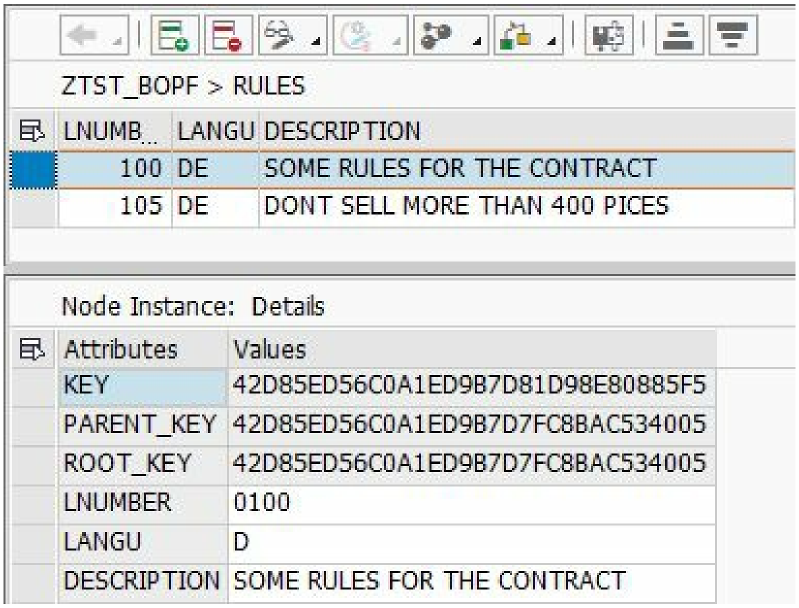


ZTST_BOPF > PARTNERS

RLTYP	PARTNER
FLVN01	777009
BUP001	888002

Node Instance: Details

Attributes	Values
KEY	42D85ED56C0A1ED9B7D8174EC87245F5
PARENT_KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
ROOT_KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
RLTYP	FLVN01
PARTNER	0000777009



ZTST_BOPF > RULES

LNUMB...	LANGU	DESCRIPTION
100	DE	SOME RULES FOR THE CONTRACT
105	DE	DONT SELL MORE THAN 400 PICES

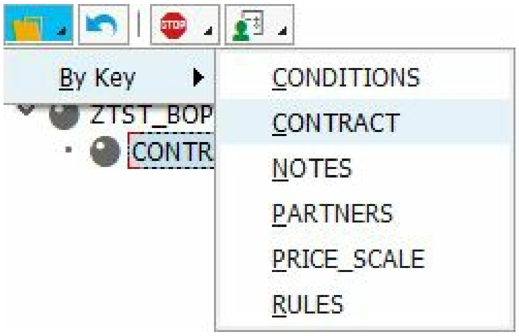
Node Instance: Details

Attributes	Values
KEY	42D85ED56C0A1ED9B7D81D98E80885F5
PARENT_KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
ROOT_KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
LNUMBER	0100
LANGU	D
DESCRIPTION	SOME RULES FOR THE CONTRACT

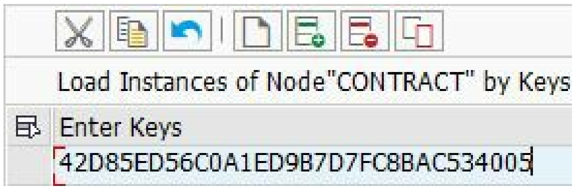
In the upper part are all data with external representation, the leading zeros are missing here for example. The lower part shows the data with the leading zeros, where you can always find the internal data. The lower part is also the detailed representation for the marked record, if there are several other records.

Read

For reading the datasets, only the standard function is available. This gives the possibility to read directly via the key of the data record. The function can be found under the folder icon on the left side of the screen.



It will open a popup, which is still empty at the beginning. With the button for a new element, we get a blank line in the list. After entering the non-speaking key (GUID) and confirm, the record is loaded.


Load Instances of Node"CONTRACT" by Keys

Enter Keys

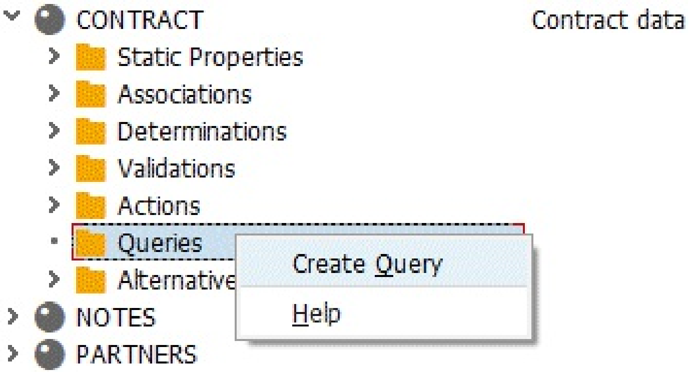
42D85ED56C0A1ED9B7D7FC8BAC534005

Queries

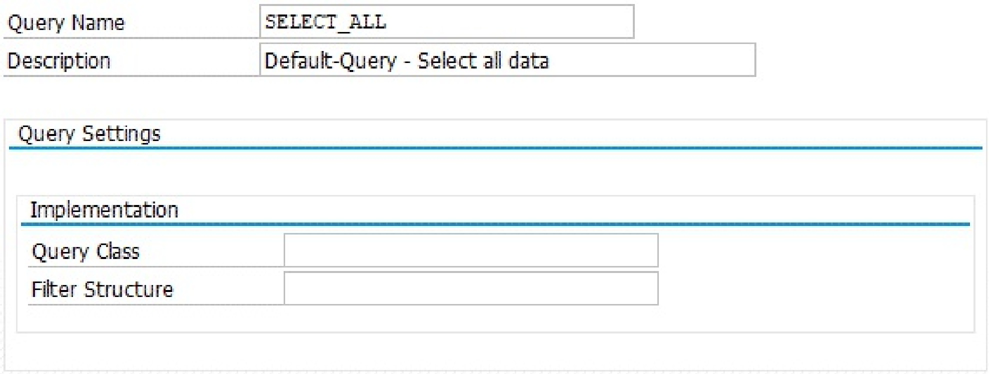
In this chapter, we'll look at how we can easily read data. Because we have already seen in the last chapter, there are only simple accesses via the key, but this must first be determined.

Query

There are queries that we can create and use. The definition of a query takes place in transaction BOBX, where we also create the model. With a right-click on queries, we can define a new query.



Without much effort, we can create two basic queries that the system already provides. SELECT_ALL finds all keys from the BOPF and returns them without restrictions. SELECT_BY_ELEMENT adopts the selection criteria and limits the number of keys. For this, the query requires a structure or field to be filtered.

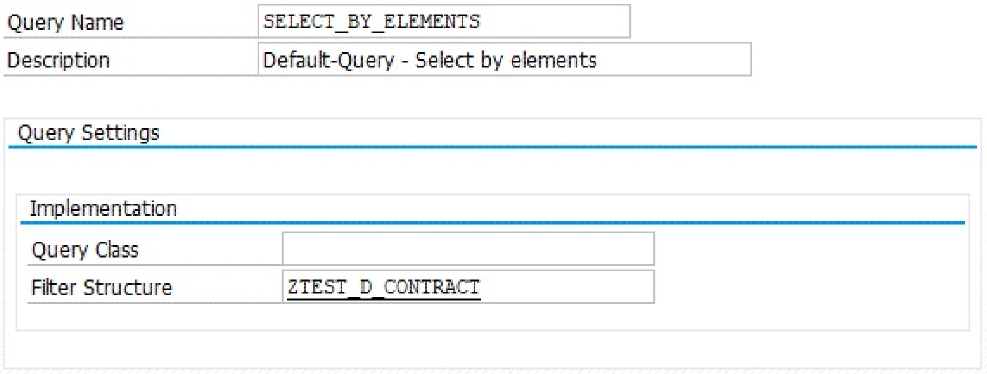


Query Name	SELECT_ALL
Description	Default-Query - Select all data

Query Settings

Implementation

Query Class	
Filter Structure	



Query Name	SELECT_BY_ELEMENTS
Description	Default-Query - Select by elements

Query Settings

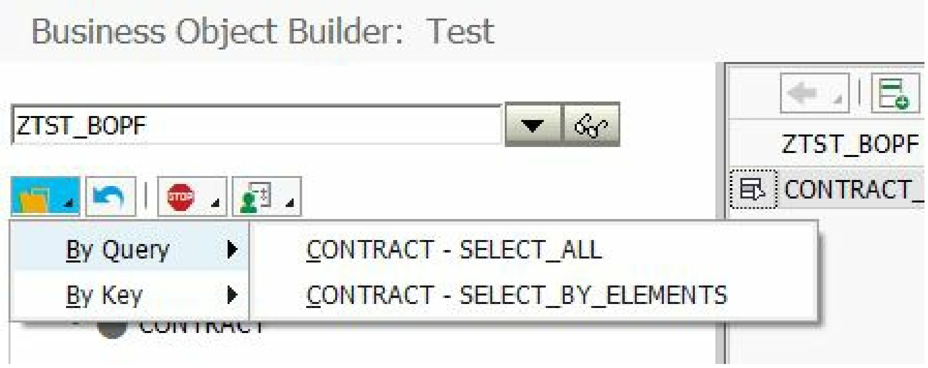
Implementation

Query Class	
Filter Structure	ZTEST_D_CONTRACT

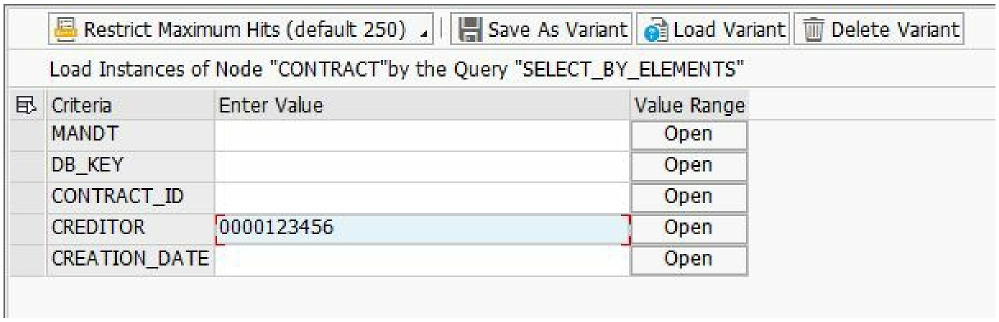
From experience, we can tell you that depositing the real table is the best way. Although the transient fields from the data are missing, the keys for PARENT can also be queried if, for example, you enter the search via a lower entity.

BOBT

After definition and activation in the model, the two new queries are also available in the test environment. These can now be used for the selection. If you just want to read all elements from the tables, the query SELECT_ALL is sufficient.



If you want to restrict the elements, you will get a popup again. This time, all fields from the structure are available for input, which should make it easier for you to narrow down the data. Now you can also restrict via the vendor.



Load Instances of Node "CONTRACT"by the Query "SELECT_BY_ELEMENTS"

Criteria	Enter Value	Value Range
MANDT		Open
DB_KEY		Open
CONTRACT_ID		Open
CREDITOR	0000123456	Open
CREATION_DATE		Open

Technical solution

Surely you are wondering how to perform the query technically? In this section, we'll show you a code example of how to apply the example above.

In the first step, you need an instance of the service manager to perform an action on the BOPF. Then you fill the selection table with the restrictions and in the last step transfer everything to the QUERY method of the service manager. You get back a message object that you should still check for errors, as well as the key table if entries are included. Since you only received the keys, you still need to request the data via the RETRIEVE method.

```
DATA:
  lt_sel  TYPE /bobf/t_frw_query_selparam,
  lt_data TYPE ztest_t_contract.

" Service Manager Instance
DATA(lo_smgr) = /bobf/cl_tra_serv_mgr_factory=>get_service_manager( zif_tst_bopf_c=>sc_bo_key
).

" Fill selection
APPEND VALUE #(
  attribute_name  = zif_tst_bopf_c=>sc_node_attribute-contract-creditor
  sign            = 'I'
  option          = 'EQ'
  low             = '0000123456'
) TO lt_sel.

" Get the keys
lo_smgr->query(
  EXPORTING
    iv_query_key            = zif_tst_bopf_c=>sc_query-contract-select_by_elements
    it_selection_parameters = lt_sel
  IMPORTING
    eo_message              = DATA(lo_msg_query)
    et_key                  = DATA(lt_key)
).

" Validate errors
IF lo_msg_query->check( ) = abap_true OR lt_key IS INITIAL.
  RETURN.
ENDIF.
```

```
" Read data
lo_smgr->retrieve(
  EXPORTING
    iv_node_key             = zif_tst_bopf_c=>sc_node-contract
    it_key                  = lt_key
  IMPORTING
    eo_message              = DATA(lo_msg)
    et_data                 = lt_data
).
```

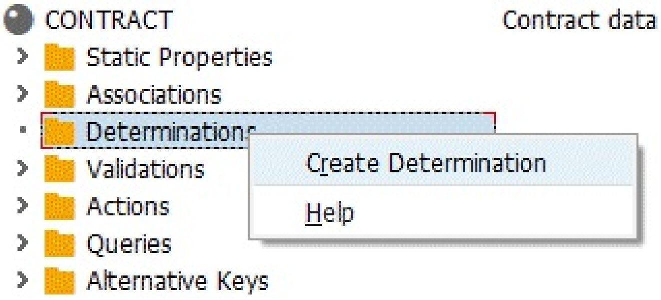
Hint: As you probably noticed, we use the constant interface of the BOPF model for the creation of the service manager, but also for the accesses. Never store the values as literals in these places, as these could change when the interface is regenerated.

Determination

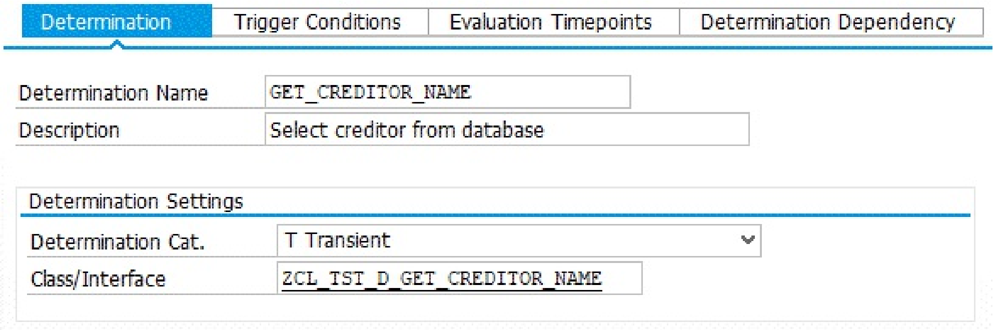
With data collection, you can read additional data at runtime when accessing BOPF. We had already presented this transient data with the data model and now we would like to show you an example of how the data can simply be read.

Create a determination

The determination is created on the respective node. For this, transaction BOBX must be called again. On the corresponding node, you now create a determination.



For this, the individual fields are filled, whereby the name of the processing class can be generated again via "Extras -> Propose Repository Names". As a category you choose Transient, because we want to read additional data for this node.

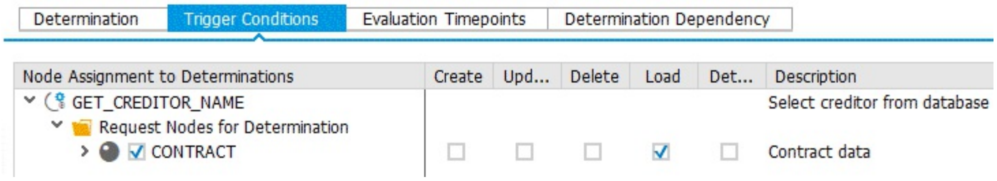


Determination Name	GET_CREDITOR_NAME
Description	Select creditor from database

Determination Settings

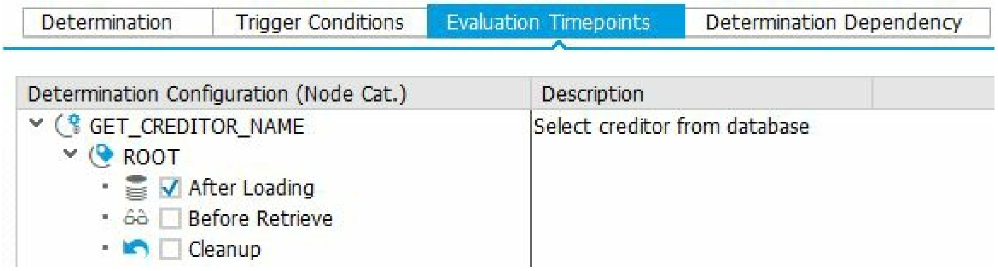
Determination Cat.	T Transient
Class/Interface	ZCL_TST_D_GET_CREDITOR_NAME

The next step is to set, when the determination should be made. For transient data in most cases the determination is sufficient for "load".



Node Assignment to Determinations	Create	Upd...	Delete	Load	Det...	Description
GET_CREDITOR_NAME						Select creditor from database
Request Nodes for Determination						
☑ CONTRACT	□	□	□	☑	□	Contract data

The evaluation timepoint is right here after loading the data from the database so that the data is complete. There are no dependencies for this node and determination.



Determination Configuration (Node Cat.)	Description
GET_CREDITOR_NAME	Select creditor from database
ROOT	
☑ After Loading	
□ Before Retrieve	
□ Cleanup	

Implementation

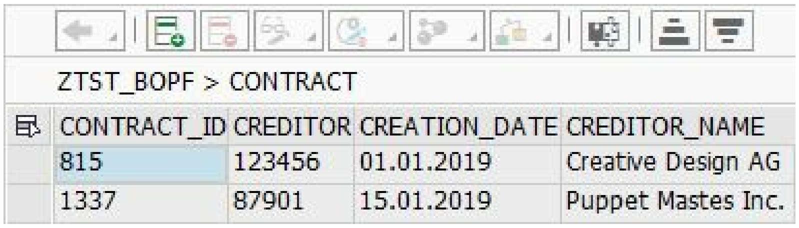
The implementation of the logic takes place in a separate class which implements the interface /BOBF/IF_FRW_DETERMINATION. In method EXECUTE, you receive all selected nodes, in this case only the table with keys. The implementation example is explained in the following code example:

```
DATA:
  lt_head TYPE ztest_t_contract.

io_read->retrieve(
  EXPORTING
    iv_node       = is_ctx-node_key
    it_key        = it_key
    iv_fill_data  = abap_true
  IMPORTING
    et_data       = lt_head
).

LOOP AT lt_head REFERENCE INTO DATA(lr_head).
  SELECT SINGLE name1
   FROM lfa1
   WHERE lifnr = @lr_head->creditor
   INTO @lr_head->creditor_name.

  IF sy-subrc = 0.
    io_modify->update(
      EXPORTING
        iv_node     = is_ctx-node_key
        iv_key      = lr_head->key
        iv_root_key = lr_head->root_key
        is_data     = lr_head
    ).
  ENDIF.
ENDLOOP.
```

Using the Retrieve method, you can read the corresponding data for the keys, using the combination table type that accepts the data. Then a loop about the header data by reference, reading the custom properties from the supplier's master record and updating the data with the update method, passing in the new data as a reference. The loop by reference has already solved this problem.

Test

When reading the node, the data is automatically read at the right time and the structure is enriched. The easiest way to check that is to try the test transaction BOBT. If you look at the sample data now, the field CREDITOR_NAME will be filled.



ZTST_BOPF > CONTRACT

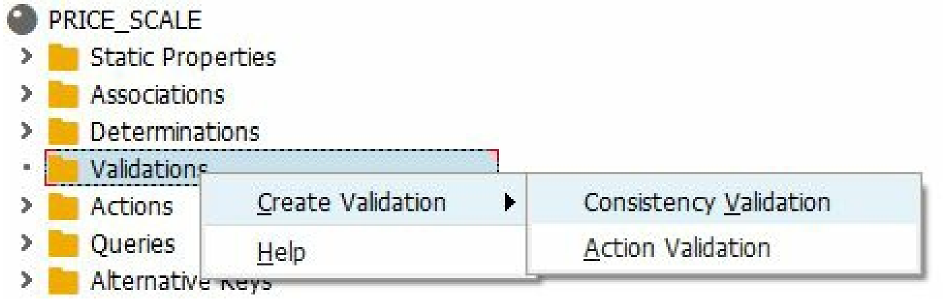
CONTRACT_ID	CREDITOR	CREATION_DATE	CREDITOR_NAME
815	123456	01.01.2019	Creative Design AG
1337	87901	15.01.2019	Puppet Mastes Inc.

Validation

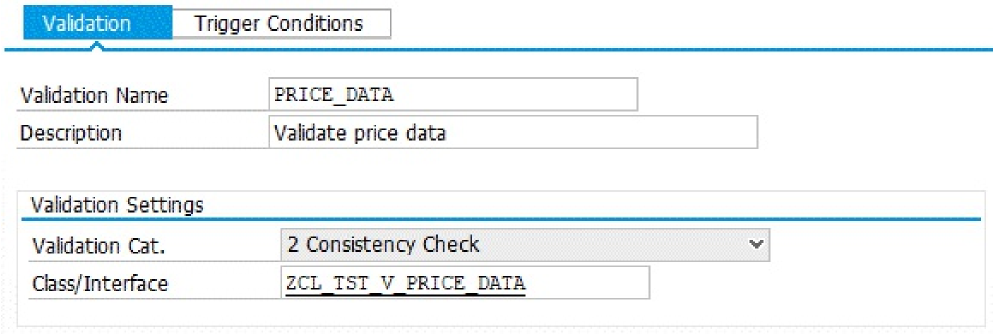
The topic of cleanliness of the data is also very important in the BOPF and provides with the validations a strong method of checking consistency. Similar to the determination, the validation in the data model is also created on the corresponding node. Here we'll show you how to define a validation and implement the appropriate logic in the class.

Create a validation

The validation, like the determination, is again defined on the corresponding node to which it applies. To do so, right-click and select "Consistency validation" to create a new empty object.



Fill in the name and description of the validation and then you can generate the class name again by suggestion of the system.

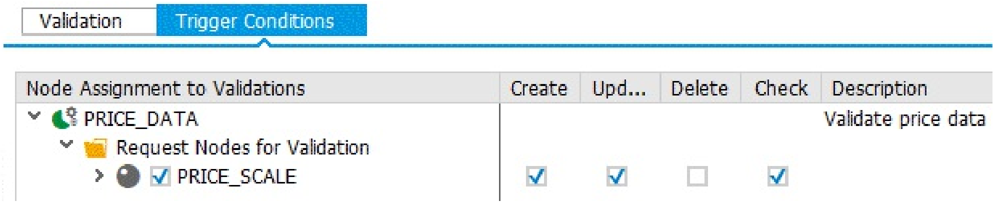


Validation Name	PRICE_DATA
Description	Validate price data

Validation Settings

Validation Cat.	2 Consistency Check
Class/Interface	ZCL_TST_V_PRICE_DATA

The condition must now be activated for certain actions (triggers). In this case, we activate it for all cases, except for deletion, since it makes no sense there. You can only go through validations during generation, if the data is then no longer changeable.



Node Assignment to Validations	Create	Upd...	Delete	Check	Description
PRICE_DATA					Validate price data
Request Nodes for Validation					
☑ PRICE_SCALE	☑	☑	□	☑	

Implementation

The logic is again implemented in the created class implementing the interface /BOBF/IF_FRW_VALIDATION. In the EXECUTE method, all checks are implemented on the data, with the following example:

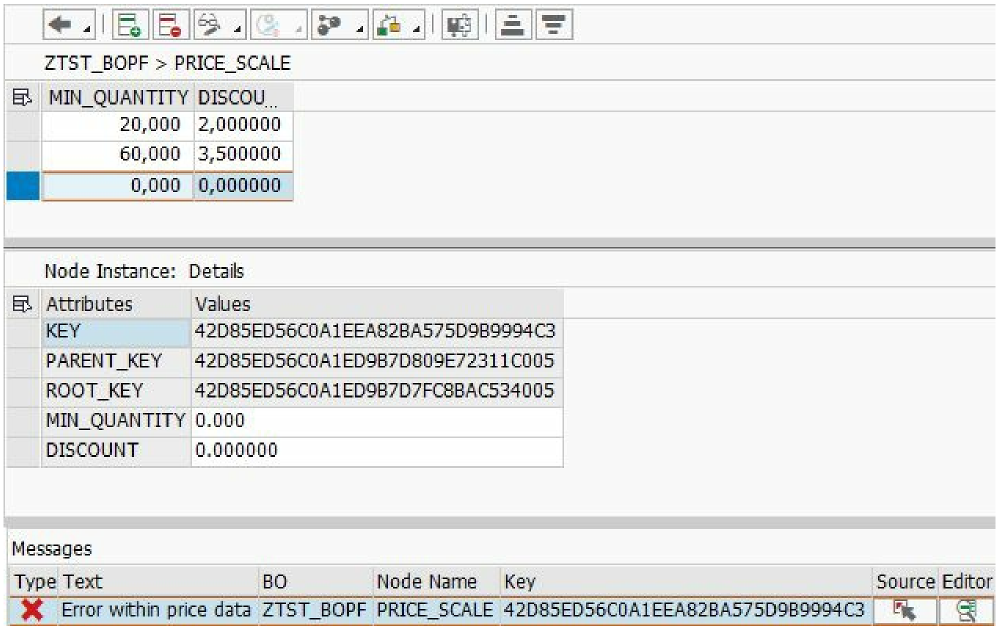
```
DATA:
  lt_price_data TYPE ztest_t_price_scale.

eo_message = /bobf/cl_frw_factory=>get_message( ).

io_read->retrieve(
  EXPORTING
    iv_node      = is_ctx-node_key
    it_key       = it_key
    iv_fill_data = abap_true
  IMPORTING
    et_data      = lt_price_data
).

LOOP AT lt_price_data REFERENCE INTO DATA(lr_price_data).
  DATA(ld_error) = abap_false.

  IF lr_price_data->min_quantity <= 0.
    ld_error = abap_true.
  ENDIF.

  IF lr_price_data->discount < 0.
    ld_error = abap_true.
  ENDIF.

  IF ld_error = abap_true.
    APPEND VALUE #( key = lr_price_data->key ) TO et_failed_key.

    eo_message->add_message(
      EXPORTING
        is_msg     = VALUE #( msgty = 'E'
                              msgid = 'ZTEST_MSG'
                              msgno = '001'
                     )
        iv_node    = is_ctx-node_key
        iv_key     = lr_price_data->key
    ).
  ENDIF.
```

```
ENDLOOP.
```

The data must first be read from the data model, because at this point only the keys are available. Afterwards, the data is checked for correctness. If the data has an error, then the key must be transferred to the failed keys table and a message can be generated. In any case, make sure that a new message instance has been created by the factory method before adding the message.

Test

When creating the test data, we will now receive a corresponding error message if the data is not filled correctly. The data can only be backed up if there are no more validation errors.



ZTST_BOPF > PRICE_SCALE

MIN_QUANTITY	DISCOU...
20,000	2,000000
60,000	3,500000
0,000	0,000000

Node Instance: Details

Attributes	Values
KEY	42D85ED56C0A1EEA82BA575D9B9994C3
PARENT_KEY	42D85ED56C0A1ED9B7D809E72311C005
ROOT_KEY	42D85ED56C0A1ED9B7D7FC8BAC534005
MIN_QUANTITY	0.000
DISCOUNT	0.000000

Messages

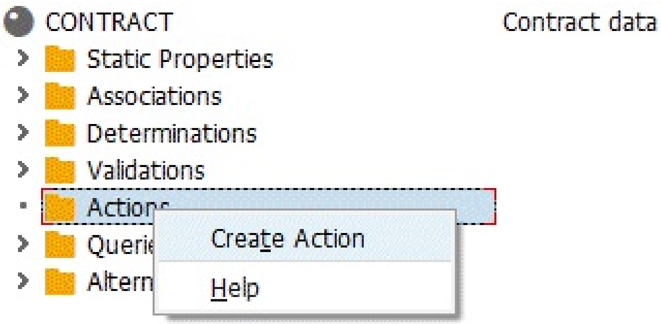
Type	Text	BO	Node Name	Key	Source	Editor
✗	Error within price data	ZTST_BOPF	PRICE_SCALE	42D85ED56C0A1EEA82BA575D9B9994C3		

Action

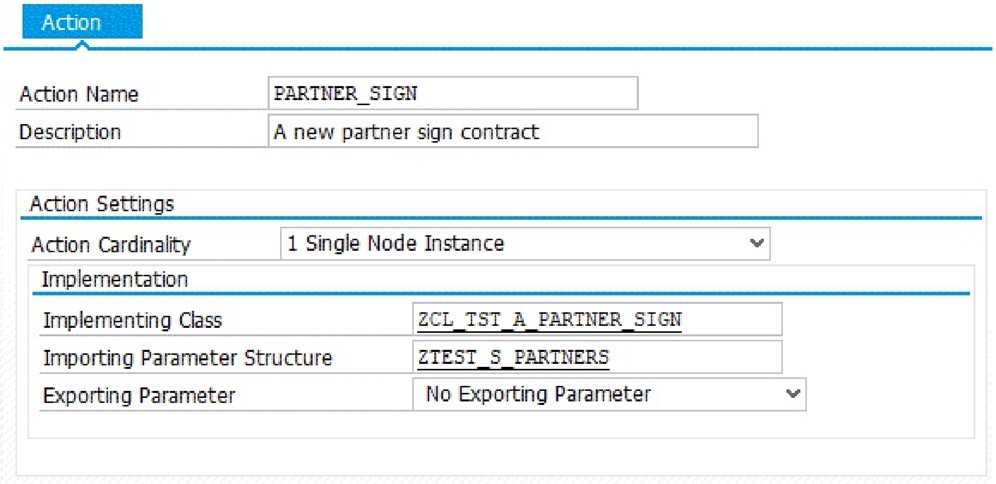
There should be different actions available on the data model that each user of the BOPF should execute? In this chapter, we'll show you what such an action can look like by "a partner is signing". The actions are externally published and collected changes to the data model, which are made available to the user.

Create an action

In the first step, we create the action at the top level, the contract. The reason is that the different partners in the real world also sign on the contract and not on the partner data.



Again, name and description must be entered to complete the data. The name of the class can be generated via the suggestion in the context menu (more about this in the investigation). The partner structure should serve as an import parameter because the new partner should sign a contract.



Action

Action Name	PARTNER_SIGN
Description	A new partner sign contract

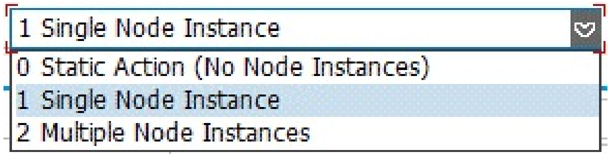
Action Settings

Action Cardinality	1 Single Node Instance

Implementation

Implementing Class	ZCL_TST_A_PARTNER_SIGN
Importing Parameter Structure	ZTEST_S_PARTNERS
Exporting Parameter	No Exporting Parameter

In the final step, we also set a cardinality of the change. There are three options that can be chosen: Work on one node, several nodes or no node. In our case, we only want to work with one contract.



1 Single Node Instance

0 Static Action (No Node Instances)
1 Single Node Instance
2 Multiple Node Instances

Implementation

We implement the logic in the new class which implements the interface /BOBF/IF_FRW_ACTION. In the EXECUTE method, the logic of the conversion is developed and all the addressed nodes are processed. A small example for the implementation:

```
DATA:
  lt_partners TYPE ztest_t_partners,
  ls_new      TYPE ztest_s_partners,
  ls_param    TYPE ztest_s_partners.

eo_message = /bobf/cl_frw_factory=>get_message( ).

IF lines( it_key ) <> 1.
  et_failed_key = it_key.
  eo_message->add_message(
    EXPORTING
      is_msg       = VALUE #( msgty = 'E'
                              msgid = 'ZTEST_MSG'
                              msgno = '003'
                    )
  ).
  RETURN.
ENDIF.

ASSIGN is_parameters->* TO FIELD-SYMBOL(<ls_param>).
IF sy-subrc = 0.
  ls_param = CORRESPONDING #( <ls_param> ).
ELSE.
  et_failed_key = it_key.
  eo_message->add_message(
    EXPORTING
      is_msg       = VALUE #( msgty = 'E'
                              msgid = 'ZTEST_MSG'
                              msgno = '004'
                    )
  ).
  RETURN.
ENDIF.
```

The first part of the coding we build the message instance, because here we validate further data and check whether data has been transferred. If more than one key has been handed over or the partner data is missing, we will trigger a corresponding error.

```
io_read->retrieve_by_association(
  EXPORTING
    iv_node         = zif_tst_bopf_c=>sc_node-contract
    it_key          = it_key
    iv_association  = zif_tst_bopf_c=>sc_association-contract-partners
    iv_fill_data    = abap_true
  IMPORTING
    et_data         = lt_partners
).

READ TABLE lt_partners
WITH KEY rltyp   = ls_param-rltyp
          partner = ls_param-partner
TRANSPORTING NO FIELDS.
IF sy-subrc = 0.
  et_failed_key = it_key.
  eo_message->add_message(
    EXPORTING
      is_msg       = VALUE #( msgty = 'E'
                              msgid = 'ZTEST_MSG'
                              msgno = '002'
                     )
  ).

  RETURN.
ENDIF.

ls_new-partner = ls_param-partner.
ls_new-rltyp = ls_param-rltyp.

io_modify->create(
  EXPORTING
    iv_node            = zif_tst_bopf_c=>sc_node-partners
    is_data            = REF #( ls_new )
    iv_assoc_key       = zif_tst_bopf_c=>sc_association-partners-to_parent
    iv_source_node_key = zif_tst_bopf_c=>sc_node-contract
    iv_source_key      = it_key[ 1 ]-key
    iv_root_key        = it_key[ 1 ]-key
  IMPORTING
    ev_key             = DATA(ld_key)
).
```

In the second part, we read out all the partner data for the association agreement and validate the current partners against the new one. If all goes well, we will add the new record to the database and sign the contract.

Report

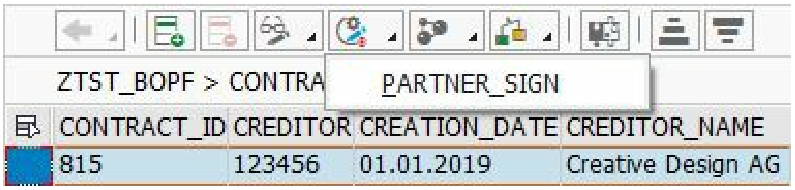
For the test of the action we want to show you an implementation in a program that does not need BOBT and what it would probably look like in the real world:

```
DATA(lo_tmgr) = /bobf/cl_tra_trans_mgr_factory=>get_transaction_manager( ).
DATA(lo_smgr) = /bobf/cl_tra_serv_mgr_factory=>get_service_manager( zif_tst_bopf_c=>sc_bo_key
).

DATA(ls_parameters) = VALUE ztest_s_partners(
  rltyp     = 'BUP001'
  partner   = '0000888002'
).

lo_smgr->do_action(
  EXPORTING
    iv_act_key          = zif_tst_bopf_c=>sc_action-contract-partner_sign
    it_key              = VALUE #( ( key = '42D85ED56C0A1ED9B7D7FC8BAC534005' ) )
    is_parameters       = REF #( ls_parameters )
  IMPORTING
    et_failed_key       = DATA(lt_failed)
    eo_message          = DATA(lo_msg)
).

IF lt_failed IS INITIAL.
  lo_tmgr->save(
    IMPORTING
      ev_rejected        = DATA(ld_reject)
      et_rejecting_bo_key = DATA(lt_keys)
  ).
ENDIF.

IF ld_reject = abap_true.
  lo_tmgr->cleanup( ).
ENDIF.
```

In our example, the data and the entry to be changed are hard-coded, here a dynamic determination should be performed and read via the contract. If the action was successful and the table lt_failed is empty, the data can be persisted to the database.

Test

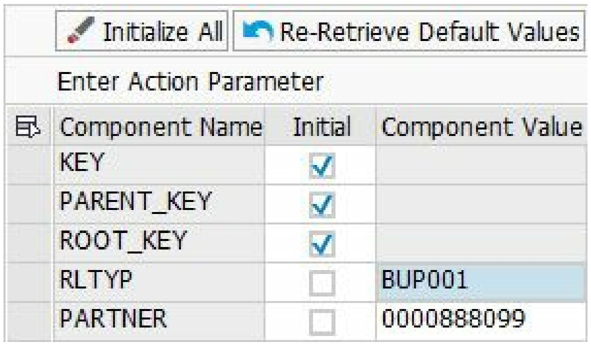
In transaction BOBT, after loading a contract, the action can be found in the upper context menu, where all available actions are listed.



ZTST_BOPF > CONTRA

CONTRACT_ID	CREDITOR	CREATION_DATE	CREDITOR_NAME
815	123456	01.01.2019	Creative Design AG

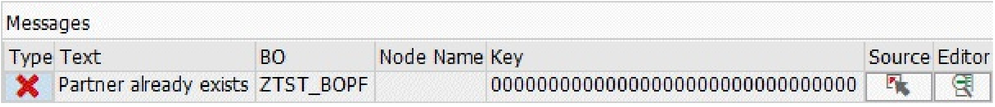
After execution of the action, a pop-up appears with the expected import data that can be filled or left empty for testing. After confirming this popup, the action will be executed and iterated through your code.



Enter Action Parameter

Component Name	Initial	Component Value
KEY	☑	
PARENT_KEY	☑	
ROOT_KEY	☑	
RLTYP	□	BUP001
PARTNER	□	0000888099

If, for example, a partner record is already included, an error message occurs as with the validations. The buttons behind the message can also be used to navigate to the source code where the message was generated. Since we left nodes and keys blank when generating the message, the information is missing in the display.



Messages

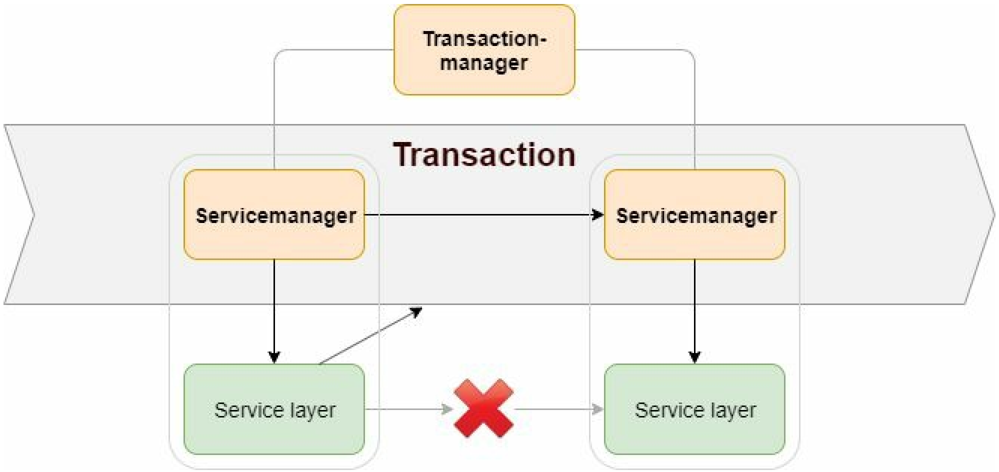
Type	Text	BO	Node Name	Key	Source	Editor
✗	Partner already exists	ZTST_BOPF		00000000000000000000000000000000		

Manager

As you've seen in recent chapters, we've already used a few managers for outside access who have done various things for us. Now we want to show you what it is all about and for what you need which manager.

Functionality

The managers are used to manage the object and the data streams within the processing in the BOPF. For a brief explanation, the following graphic illustrates the scenario:



The transaction includes all the actions and changes that are to be performed on a BOPF model and can be thought of as a flow, in which individual actions are repeatedly carried out. For this the managers are needed.

Transaction manager

Monitors and collects all changes to the model and forms a LUW to keep the data consistent. The manager takes care of the entire transaction and at the end writes the data back to the database. That's why he's the staple on everything.

Service manager

Always created for a specific BOPF model so the manager knows what data he manages. Provides actions to change the data, perform actions on the model, read and validate data. The actions of the manager are always on their own.

Service layer

Only affected by the service manager and unable to communicate with another service layer of another manager. After the action has been taken, processing returns to the transaction layer.

Example

In this specific case, you can read data only with the help of a service manager, but you need the transaction manager as soon as the data is to be changed concretely.

In the variant without transaction manager, the data is read by filter and via the QUERY method. As a result, you get a table with the affected keys and can then use the RETRIEVE method to read in the data in the appropriate structure:

```
DATA:
  lt_sel       TYPE /bobf/t_frw_query_selparam,
  lt_contracts TYPE ztest_t_contract.

DATA(lo_smgr) = /bobf/cl_tra_serv_mgr_factory=>get_service_manager( zif_tst_bopf_c=>sc_bo_key
).

INSERT VALUE #(
  attribute_name  = 'CREDITOR'
  sign            = 'I'
  option          = 'EQ'
  low             = '0000123456'
) INTO TABLE lt_sel.

lo_smgr->query(
  EXPORTING
    iv_query_key           = zif_tst_bopf_c=>sc_query-contract-select_by_elements
    it_selection_parameters = lt_sel
  IMPORTING
    et_key                 = DATA(lt_key)
).

lo_smgr->retrieve(
  EXPORTING
    iv_node_key            = zif_tst_bopf_c=>sc_node-contract
    it_key                 = lt_key
  IMPORTING
    et_data                = lt_contracts
).
```

In the variant with transaction manager, a table with changes is created (a contract with a rule) and changed by the MODIFY method. If there are no problems and errors, the transaction manager can take over the data and create it in the tables. In the case of an error, the CLEANUP method is called, which rolls back the steps since the last save.

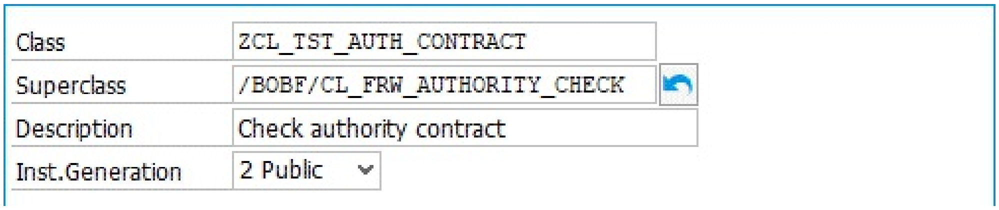
```
DATA:
  lt_mod TYPE /bobf/t_frw_modification.

DATA(ld_key) = /bobf/cl_frw_factory=>get_new_key( ).

" Contract data
DATA(ls_contract) = VALUE ztest_s_contract(
  contract_id   = '910'
  creditor      = '0000123456'
  creation_date = '20191101'
).

INSERT VALUE #(
  node        = zif_tst_bopf_c=>sc_node-contract
  change_mode = /bobf/if_frw_c=>sc_modify_create
  data        = REF #( ls_contract )
  key         = ld_key
) INTO TABLE lt_mod.

" Rules
DATA(ls_rule) = VALUE ztest_s_rules(
  lnumber     = 1
  langu       = 'D'
  description = 'One rule for all'
).

INSERT VALUE #(
  node        = zif_tst_bopf_c=>sc_node-rules
  change_mode = /bobf/if_frw_c=>sc_modify_create
  data        = REF #( ls_rule )
  association = zif_tst_bopf_c=>sc_association-rules-to_parent
  source_node = zif_tst_bopf_c=>sc_node-contract
  source_key  = ld_key
) INTO TABLE lt_mod.

DATA(lo_tmgr) = /bobf/cl_tra_trans_mgr_factory=>get_transaction_manager( ).
DATA(lo_smgr) = /bobf/cl_tra_serv_mgr_factory=>get_service_manager( zif_tst_bopf_c=>sc_bo_key
).

lo_smgr->modify(
  EXPORTING
    it_modification = lt_mod
  IMPORTING
    eo_change       = DATA(lo_change)
    eo_message      = DATA(lo_msg)
).

IF lo_msg->check( ) = abap_false.
  lo_tmgr->save(
    IMPORTING
      ev_rejected         = DATA(ld_reject)
      et_rejecting_bo_key = DATA(lt_keys)
  ).
ENDIF.

IF ld_reject = abap_true.
  lo_tmgr->cleanup( ).
ENDIF.
```

Authorization

The developer help in the BOPF describes two ways to create your own authorization check, first with the class /BOBF/CL_LIB_AUTHCHECK_W_QUERY or with completely new implementation of your own logic with the abstract class /BOBF/CL_FRW_AUTHORITY_CHECK. Now we show you the complete route via the abstract class, in which you implement your own authorization checks.

Preparation

The class for checking the authorizations must be created ourselves, since we are working with inheritance and have to do some rework on the new class. Name, description and parent class must be entered, the rest can remain on the standard settings.



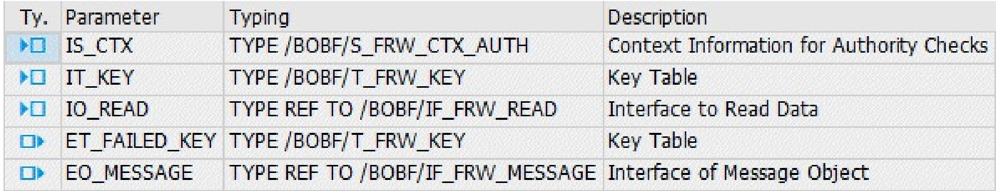
Class	ZCL_TST_AUTH_CONTRACT
Superclass	/BOBF/CL_FRW_AUTHORITY_CHECK
Description	Check authority contract
Inst.Generation	2 Public

You should redefine the methods CHECK_AUTHORITY and CHECK_AUTHORITY_STATICALLY in the first step, but you can leave them empty. The method GET_QUERY_CONDITION_PROVIDER should also be redefined and requires implementation. In the following coding you will get a sample coding of how such an implementation can look like. The configuration is read there and a provider for the node and objects is created. Without the implementation of the method, an error occurs when performing operations on the business object.

```
DATA(lo_conf) = /bobf/cl_frw_factory=>get_configuration( zif_tst_bopf_c=>sc_bo_key ).

lo_conf->get_node(
  EXPORTING
    iv_node_key = is_ctx-node_key
  IMPORTING
    es_node     = DATA(ls_node_conf)
).

ro_provider = /bobf/cl_sadl_auth_cond_provid=>get_instance(
  EXPORTING
    iv_anchor_entity = /bobf/cl_sadl_entity=>get_entity_id_by_bo_node_name(
                         EXPORTING
                           iv_bo_name   = CONV #( lo_conf->ms_obj-bo_name )
                           iv_node_name = CONV #( ls_node_conf-node_name )
                       )
    is_customizing_context = is_ctx
).
```

Implementation

During the implementation, we use the dynamic check on values of the instance, i.e. we implement the CHECK_AUTHORITY method. The method has the following interface, which we can use:



Ty.	Parameter	Typing	Description
▸□	IS_CTX	TYPE /BOBF/S_FRW_CTX_AUTH	Context Information for Authority Checks
▸□	IT_KEY	TYPE /BOBF/T_FRW_KEY	Key Table
▸□	IO_READ	TYPE REF TO /BOBF/IF_FRW_READ	Interface to Read Data
□▸	ET_FAILED_KEY	TYPE /BOBF/T_FRW_KEY	Key Table
□▸	EO_MESSAGE	TYPE REF TO /BOBF/IF_FRW_MESSAGE	Interface of Message Object

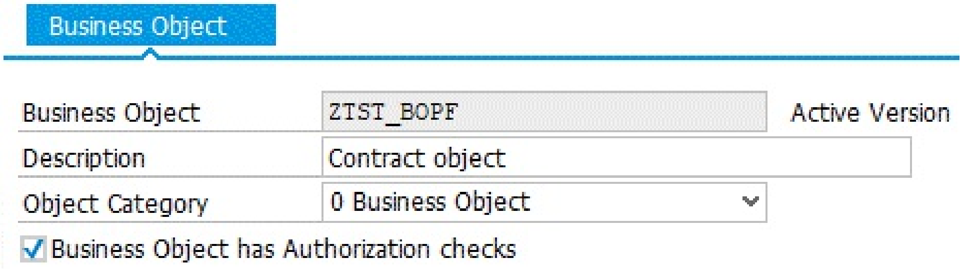
Accordingly, the context of the request is available to us, with which we first check whether the correct node has been selected. BOPF checks the authorizations for all nodes, but our check should only work specifically for the head entry. If this is successful, we read in the data for the node and check the data against an authorization object. Faulty nodes are taken over by the return table ET_FAILED_KEY so that these data records are filtered out. An implementation could therefore look like this:

```
DATA:
  lt_head  TYPE ztest_t_contract.

IF is_ctx-node_key <> zif_tst_bopf_c=>sc_node-contract.
  RETURN.
ENDIF.

eo_message = /bobf/cl_frw_factory=>get_message( ).

io_read->retrieve(
  EXPORTING
    iv_node       = is_ctx-node_key
    it_key        = it_key
    iv_fill_data  = abap_true
  IMPORTING
    et_data       = lt_head
).

LOOP AT lt_head REFERENCE INTO DATA(lr_head).
```

```
  " Do Authority Check

  IF sy-subrc <> 0.
    INSERT VALUE #( key = lr_head->key ) INTO TABLE et_failed_key.
  ENDIF.
ENDLOOP.
```

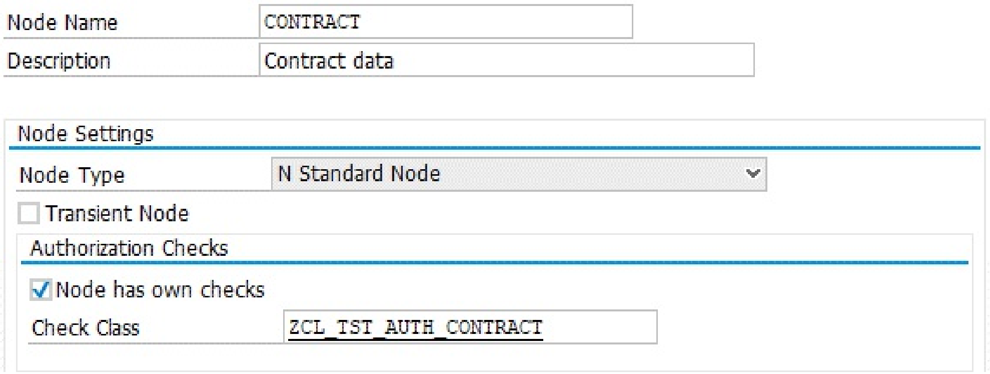
Activation

However, your own authorization check must first be activated before it run automatically. For this, the checkbox "Business Object has authorization checks" must be activated in the header of the business object.



Business Object: ZTST_BOPF Active Version
Description: Contract object
Object Category: 0 Business Object
☑ Business Object has Authorization checks

This unlocks additional options and fields on the nodes. The checkbox "Node has own checks" and the deposit of the class then, activate the checks. Then the authorization for the respective node is checked with each access.


Node Name: CONTRACT
Description: Contract data

Node Settings
Node Type: N Standard Node
□ Transient Node

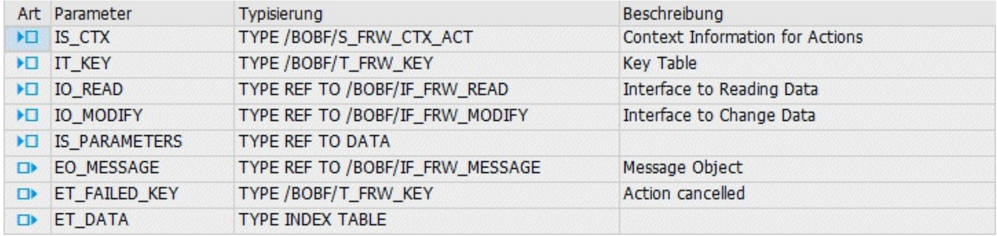
Authorization Checks
☑ Node has own checks
Check Class: ZCL_TST_AUTH_CONTRACT

Interface

When working with BOPF, you will inevitably come across the interfaces that are available and use them to read or change data. Here we are going to familiarize you with the names and uses so that you can use them efficiently and instinctively.

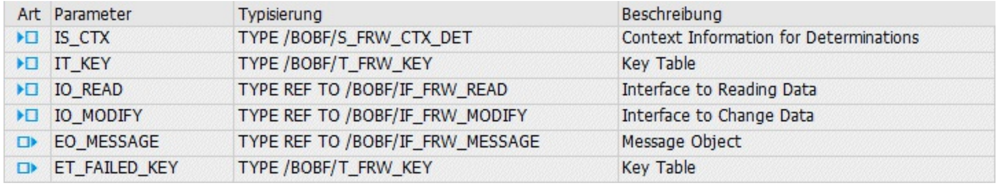
Usage

As usage, we want to introduce two interfaces that you will probably see more often in the end during the BOPF development. The first image relates to the EXECUTE method of the action. Whenever you implement an action, you will get this interface:



Art	Parameter	Typisierung	Beschreibung
▸□	IS_CTX	TYPE /BOBF/S_FRW_CTX_ACT	Context Information for Actions
▸□	IT_KEY	TYPE /BOBF/T_FRW_KEY	Key Table
▸□	IO_READ	TYPE REF TO /BOBF/IF_FRW_READ	Interface to Reading Data
▸□	IO_MODIFY	TYPE REF TO /BOBF/IF_FRW_MODIFY	Interface to Change Data
▸□	IS_PARAMETERS	TYPE REF TO DATA	
□▸	EO_MESSAGE	TYPE REF TO /BOBF/IF_FRW_MESSAGE	Message Object
□▸	ET_FAILED_KEY	TYPE /BOBF/T_FRW_KEY	Action cancelled
□▸	ET_DATA	TYPE INDEX TABLE	

The second picture relates to a determination. No parameters structure is passed and no return data are filled. Here you should only work with the data that is available in the BOPF and field contents are adjusted.



Art	Parameter	Typisierung	Beschreibung
▸□	IS_CTX	TYPE /BOBF/S_FRW_CTX_DET	Context Information for Determinations
▸□	IT_KEY	TYPE /BOBF/T_FRW_KEY	Key Table
▸□	IO_READ	TYPE REF TO /BOBF/IF_FRW_READ	Interface to Reading Data
▸□	IO_MODIFY	TYPE REF TO /BOBF/IF_FRW_MODIFY	Interface to Change Data
□▸	EO_MESSAGE	TYPE REF TO /BOBF/IF_FRW_MESSAGE	Message Object
□▸	ET_FAILED_KEY	TYPE /BOBF/T_FRW_KEY	Key Table

Variables

When using BOPF you will always come across the same variables and interfaces, in this section we want to go into the typical variables and names.

IS_CTX

- Structure
- Information to the business objekt, node and relationship

IT_KEY

- Table
- Selected keys (without data)

IO_READ

- Object
- Used to read data
- Method RETRIEVE - Read data
- Method RETRIEVE_BY_ASSOCIATION - Read above association

IO_MODIFY

- Object
- Used to change data
- Method DO_ACTION - Trigger an action
- Method DO_MODIFY - Change some data

IO_QUERY

- Object
- Used to query some data

IV_FILL_DATA

- Field
- Indicator whether return data should only be the keys or also data
- in most cases the keys are enough (performance)

EO_MESSAGE

- Object
- Created messages

ET_FAILED_KEY

- Table
- Bad keys
- e.g. from authorization check or when accessing

ET_DATA

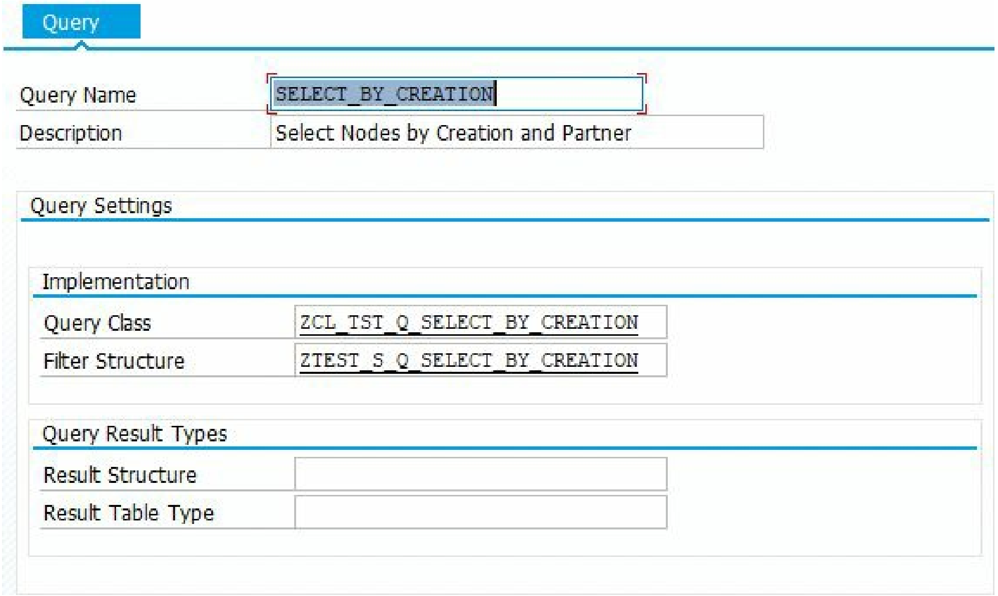
- Table
- Data from BOPF
- Type depends on the current node

Custom Query

In some cases, you would like to announce interfaces to the outside and give your user the opportunity to perform frequently used queries again and again? There are custom queries that you can define in the BOPF. Now we show you an example of how you can implement something like this.

BOBX

It all starts with defining the query in the data model. To do this, go back to transaction BOBX to expand the model and create your own query. You store a name and optionally a description and you can have the dictionary object names generated again.



Query

Query Name	SELECT_BY_CREATION
Description	Select Nodes by Creation and Partner

Query Settings

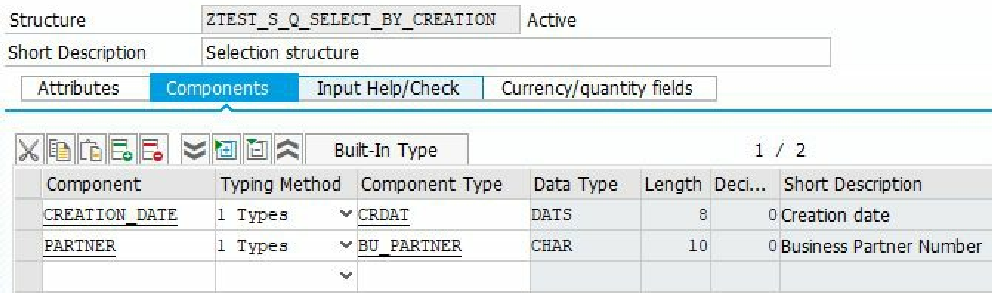
Implementation

Query Class	ZCL_TST_Q_SELECT_BY_CREATION
Filter Structure	ZTEST_S_Q_SELECT_BY_CREATION

Query Result Types

Result Structure	
Result Table Type	

We would like to enable the user to perform a custom and unique query and therefore provide some fields from our data model in a structure, the name of the structure we can also have generated by the system. Before we create the class, we need the complete structure.

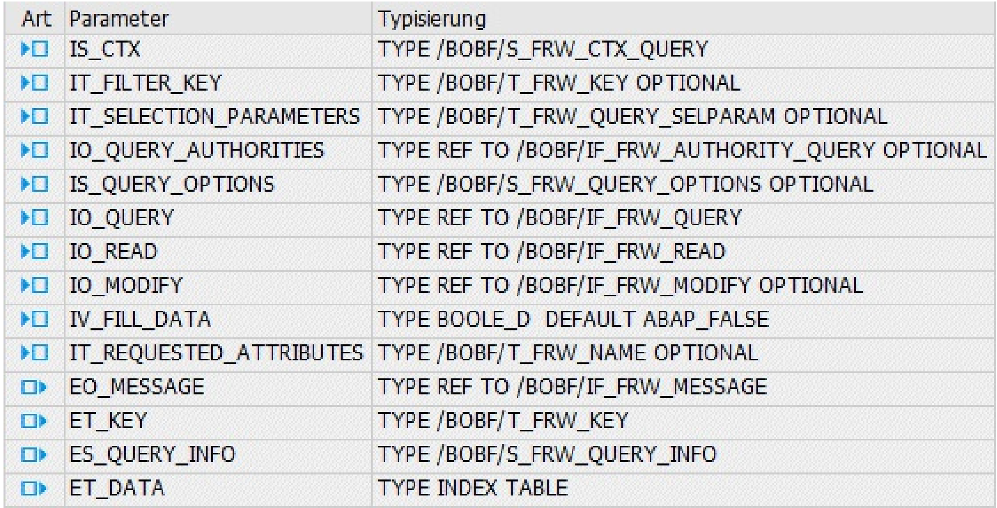


Structure	ZTEST_S_Q_SELECT_BY_CREATION	Active
Short Description	Selection structure	

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
CREATION_DATE	1 Types	CRDAT	DATS	8	0	Creation date
PARTNER	1 Types	BU_PARTNER	CHAR	10	0	Business Partner Number

Data interface

Let’s take another look at the data interface of the method before we start implementing it. As you will recognize from some chapters before, we already know many parameters. There is again the configuration data of the current node and objects for reading and changing data.



Art	Parameter	Typisierung
▸□	IS_CTX	TYPE /BOBF/S_FRW_CTX_QUERY
▸□	IT_FILTER_KEY	TYPE /BOBF/T_FRW_KEY OPTIONAL
▸□	IT_SELECTION_PARAMETERS	TYPE /BOBF/T_FRW_QUERY_SELPARAM OPTIONAL
▸□	IO_QUERY_AUTHORITIES	TYPE REF TO /BOBF/IF_FRW_AUTHORITY_QUERY OPTIONAL
▸□	IS_QUERY_OPTIONS	TYPE /BOBF/S_FRW_QUERY_OPTIONS OPTIONAL
▸□	IO_QUERY	TYPE REF TO /BOBF/IF_FRW_QUERY
▸□	IO_READ	TYPE REF TO /BOBF/IF_FRW_READ
▸□	IO_MODIFY	TYPE REF TO /BOBF/IF_FRW_MODIFY OPTIONAL
▸□	IV_FILL_DATA	TYPE BOOLE_D DEFAULT ABAP_FALSE
▸□	IT_REQUESTED_ATTRIBUTES	TYPE /BOBF/T_FRW_NAME OPTIONAL
□▸	EO_MESSAGE	TYPE REF TO /BOBF/IF_FRW_MESSAGE
□▸	ET_KEY	TYPE /BOBF/T_FRW_KEY
□▸	ES_QUERY_INFO	TYPE /BOBF/S_FRW_QUERY_INFO
□▸	ET_DATA	TYPE INDEX TABLE

For your own query, the selection (IT_SELECTION_PARAMETERS), which contains the value restrictions for the individual fields, the query object (IO_QUERY) with which we can carry out standard queries, and the return fields ET_KEY and ET_DATA are particularly relevant. If something goes wrong, the message object (EO_MESSAGE) should of course be filled.

Implementation

You now have to process the following steps: Break down the selection into the individual ranges and do a select on the database. From a performance perspective, you can of course also use a database view or CDS view, which would be a bit more powerful than the JOIN.

In the next step you transfer the data to the query object, which will now select and return the correct data based on the settings and keys.

```
DATA:
  lt_key       TYPE /bobf/t_frw_key,
  lt_r_create  TYPE RANGE OF crdat,
  lt_r_partner TYPE RANGE OF bu_partner.

eo_message = /bobf/cl_frw_factory=>get_message( ).
CLEAR: et_key, et_data.

LOOP AT it_selection_parameters REFERENCE INTO DATA(lr_param).
  CASE lr_param->attribute_name.
    WHEN 'CREATION_DATE'.
      INSERT VALUE #(
        sign    = lr_param->sign
        option  = lr_param->option
        low     = lr_param->low
        high    = lr_param->high
      ) INTO TABLE lt_r_create.
    WHEN 'PARTNER'.
      INSERT VALUE #(
        sign    = lr_param->sign
        option  = lr_param->option
        low     = lr_param->low
        high    = lr_param->high
      ) INTO TABLE lt_r_partner.
  ENDCASE.
ENDLOOP.

SELECT c~db_key
FROM ztest_d_contract AS c
INNER JOIN ztest_d_partner AS p
  ON p~parent_key = c~db_key
WHERE c~creation_date IN @lt_r_create
   AND p~partner IN @lt_r_partner
INTO TABLE @lt_key.
```

```
IF sy-subrc = 0.
  io_query->query(
    EXPORTING
      is_ctx                   = is_ctx
      it_filter_key            = lt_key
      io_query_authorities    = io_query_authorities
      is_query_options        = is_query_options
      io_query                 = io_query
      io_read                  = io_read
      io_modify                = io_modify
      iv_fill_data             = iv_fill_data
      it_requested_attributes = it_requested_attributes
    IMPORTING
      eo_message               = eo_message
      et_key                   = et_key
      es_query_info            = es_query_info
      et_data                  = et_data
  ).
ENDIF.
```

Helper methods

In this chapter we want to show you briefly what methods and helps from BOPF are available that support you in development and save you time when you reuse them. In some places you should also use them, otherwise problems or errors can occur.

Framework Factory

You get some help methods available via the framework's factory class /BOBF/CL_FRW_FACTORY. These are general methods for handling the data.

GET_MESSAGE

Return of a new message object for the transfer of error messages in the event of problems with data determination or processing. The current error status can also be determined via the object.

GET_NEW_KEY

Generation of a new unique key to save the data in the database. Usually a key is generated if none is handed over. However, if you want to create data with associations, you already need the key of the previous dataset for the association.

GET_NEW_TRANSIENT_KEY

Creates a new transient key and returns it. These keys can be created for disposable records if, for example, entire nodes in the model are transient but need a key for identification.

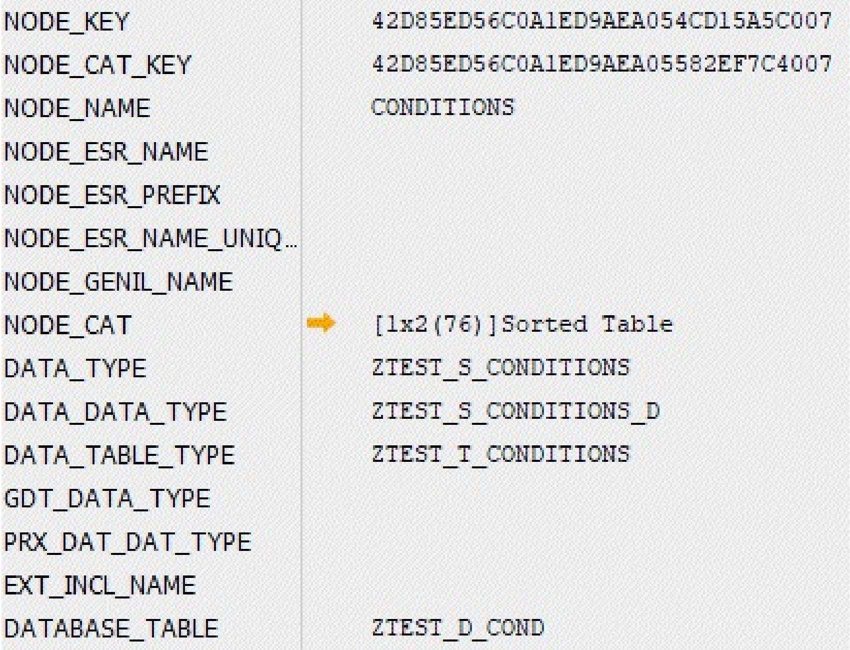
GET_CONFIGURATION

Reading in the configuration for a BOPF model (key of the data model from the constant interface) and returning an object of the type /BOBF/CL_CONFRT_COMPLETE.

```
DATA(lo_config) = /bobf/cl_frw_factory=>get_configuration( zif_tst_bopf_c=>sc_bo_key ).

lo_config->get_node(
  EXPORTING
    iv_node_key = zif_tst_bopf_c=>sc_node-conditions
  IMPORTING
```

```
    es_node      = DATA(ls_node)
).
```

The structure of LS_NODE then contains information about the selected node in the example above. This makes it easy to find the data types or tables for the selected node.



NODE_KEY	42D85ED56C0A1ED9AEA054CD15A5C007
NODE_CAT_KEY	42D85ED56C0A1ED9AEA05582EF7C4007
NODE_NAME	CONDITIONS
NODE_ESR_NAME	
NODE_ESR_PREFIX	
NODE_ESR_NAME_UNIQ…	
NODE_GENIL_NAME	
NODE_CAT	➡ [1x2(76)]Sorted Table
DATA_TYPE	ZTEST_S_CONDITIONS
DATA_DATA_TYPE	ZTEST_S_CONDITIONS_D
DATA_TABLE_TYPE	ZTEST_T_CONDITIONS
GDT_DATA_TYPE	
PRX_DAT_DAT_TYPE	
EXT_INCL_NAME	
DATABASE_TABLE	ZTEST_D_COND

Constant Interface

The generated constant interface is an important access point to the generated keys and nodes and connects them with meaningful names. As you have already seen in the example above, you can read the configuration for the data model in a readable form without having to remember the technical identifier.

Hint: With new nodes or changes in the model, the constant interface is regenerated, so it is best practice to only access the constants, as these do not change as long as the node names do not change.

As an example, an extract from the interface for our data model:

```
INTERFACE zif_tst_bopf_c
  PUBLIC .

  INTERFACES /bobf/if_lib_constants .

  CONSTANTS:
    BEGIN OF sc_action,
      BEGIN OF conditions,
        create_conditions   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0558383898007',
        delete_conditions   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA055838416C007',
        save_conditions     TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA055838493C007',
        update_conditions   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0558383D70007',
        validate_conditions TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0558384568007',
      END OF conditions,
      BEGIN OF contract,
        create_contract   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0525CBC3E8006',
        delete_contract   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0525CBF46C006',
        lock_contract     TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0525B61750006',
        partner_sign      TYPE /bobf/act_key VALUE '42D85ED56C0A1EEA82BB08B1DA099376',
        save_contract     TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0525CBFCF0006',
        unlock_contract   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0525B89214006',
        update_contract   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0525CBC8C0006',
        validate_contract TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0525CBF8E0006',
      END OF contract,
      BEGIN OF notes,
        create_notes   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA058A5C783400A',
        delete_notes   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA058A5C85CC00A',
        save_notes     TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA058A5C8DD800A',
        update_notes   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA058A5C81BC00A',
        validate_notes TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA058A5C89F000A',
      END OF notes,
      BEGIN OF partners,
        create_partners   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0576054AC800A',
        delete_partners   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA057605583800A',
        save_partners     TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA057605638C00A',
        update_partners   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA057605527000A',
        validate_partners TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA0576055DD800A',
      END OF partners,
      BEGIN OF price_scale,
        create_price_scale   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA05A0EE35D800A',
        delete_price_scale   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA05A0EE453C00A',
        save_price_scale     TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA05A0EE511C00A',
        update_price_scale   TYPE /bobf/act_key VALUE
'42D85ED56C0A1ED9AEA05A0EE3F6000A',
        validate_price_scale TYPE /bobf/act_key VALUE
'42D85ED56C0A1ED9AEA05A0EE4B6800A',
      END OF price_scale,
      BEGIN OF rules,
        create_rules   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA057BD6EDBC00A',
        delete_rules   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA057BD6F94C00A',
        save_rules     TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA057BD7014400A',
        update_rules   TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA057BD6F2E400A',
        validate_rules TYPE /bobf/act_key VALUE '42D85ED56C0A1ED9AEA057BD6FD5C00A',
      END OF rules,
    END OF sc_action .
...
```

Performance

You have probably already wondered what the performance of this framework shows in the end when we access the individual data? Here we want to talk about this topic and give you some suggestions.

Amount of data

In principle, the framework is very fast when accessing individual components if you subtract the loading time of the framework and the settings. You don't have to worry about direct access to a small amount of data or reading sub-items, since these accesses are sufficiently fast enough.

However, as soon as you use the BOPF model in a gateway service, for example, and the first "count query" goes on the total data, you will not be able to avoid a manually SELECT. With a large amount of data, it can already be worth implementing a manual access.

Performance black hole

Normal access to the data is quick, but there are some mechanisms in the BOPF that slow down performance. These include:

- Authority checks
- Determinations

Authority check

The data is protected by an authorization check and each record must be checked accordingly? This can have a strong impact on performance, especially if you have to read all or a lot of data. Here it is advisable to work with a buffer if you have already carried out a test for a combination. The check requires more performance than access to a sorted table.

Determination

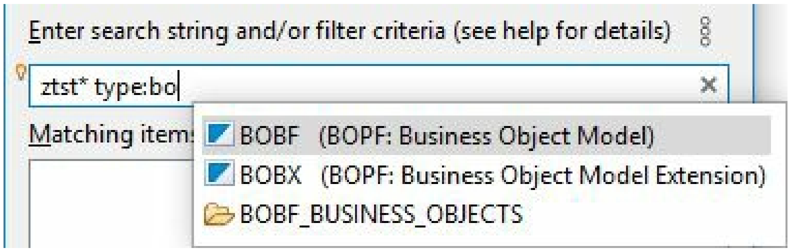
If data is also derived, it is usually read from other tables, which means additional access. A determination cannot be deactivated for only special access. Here is our recommendation to you to set manual access directly to the data table.

Eclipse

The working environment for ABAP is to become Eclipse in the future, says SAP and is bringing more and more features for ABAP development into Eclipse. Now we want to show you what is working so far in relation to BOPF in Eclipse.

Search

In order to edit a BOPF model, it has to be found first. To do this, you can simply use the general search in Eclipse, which you can access via shortcurt CTRL + SHIFT + A. Here you can work with a type filter to filter by BOBF or BOBX and only get relevant objects.



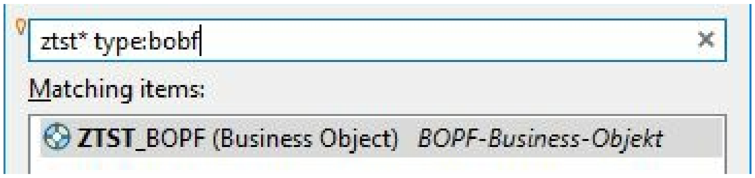
Enter search string and/or filter criteria (see help for details)

ztst* type:bo

Matching item:

BOBF (BOPF: Business Object Model)
BOBX (BOPF: Business Object Model Extension)
BOBF_BUSINESS_OBJECTS

In this example we wanted to have all ZTST objects of type BOBF. Since we only have one such model in the system, the corresponding result is displayed.

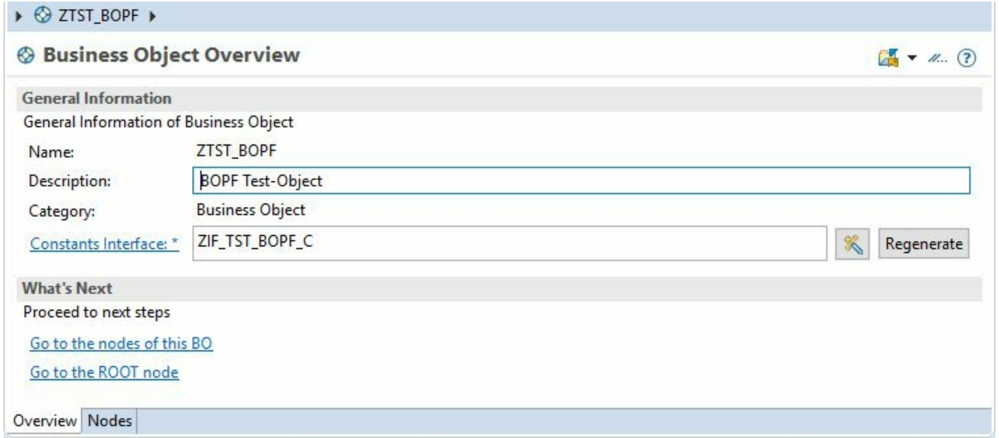


ztst* type:bobf

Matching items:

ZTST_BOPF (Business Object) *BOPF-Business-Objekt*

After selecting the object, you will land in the overview of the BOPF. From here you can control the different areas and objects.



ZTST_BOPF

Business Object Overview

General Information

General Information of Business Object

Name: ZTST_BOPF

Description: BOPF Test-Object

Category: Business Object

Constants Interface: * ZIF_TST_BOPF_C

What's Next

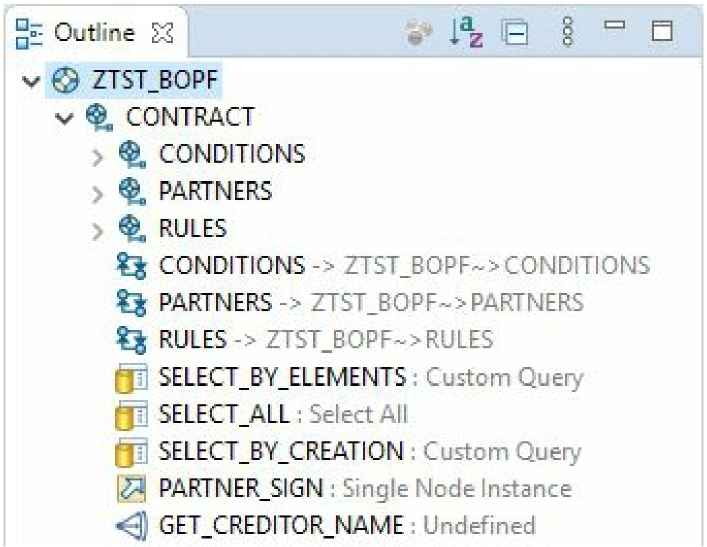
Proceed to next steps

Go to the nodes of this BO

Go to the ROOT node

Navigation

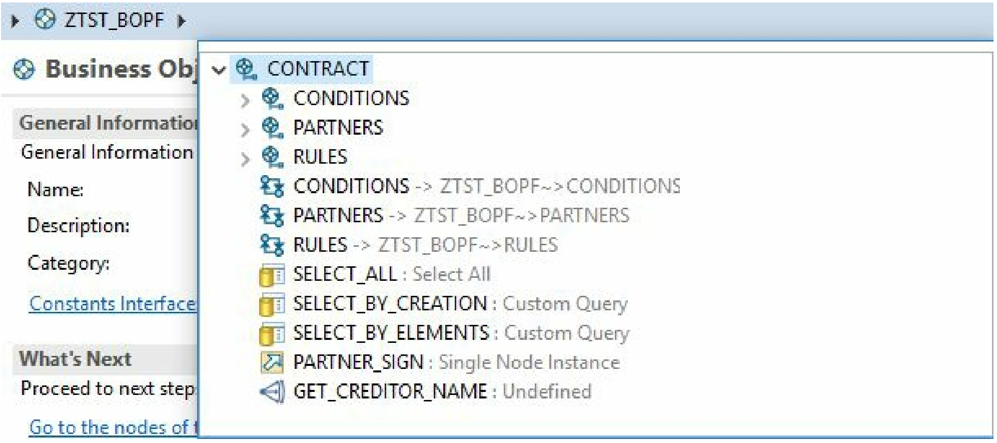
There are several ways that the ADTs offer you navigation and overview in the BOPF. On the one hand, the outlines show you an overview of the entire property and you can easily navigate to the individual sections.



Outline

ZTST_BOPF
CONTRACT
CONDITIONS
PARTNERS
RULES
CONDITIONS -> ZTST_BOPF~>CONDITIONS
PARTNERS -> ZTST_BOPF~>PARTNERS
RULES -> ZTST_BOPF~>RULES
SELECT_BY_ELEMENTS : Custom Query
SELECT_ALL : Select All
SELECT_BY_CREATION : Custom Query
PARTNER_SIGN : Single Node Instance
GET_CREDITOR_NAME : Undefined

On the other hand, you can also access the subsequent sections of an object using the arrow in the upper area. With one click, the menu for navigation opens.



ZTST_BOPF

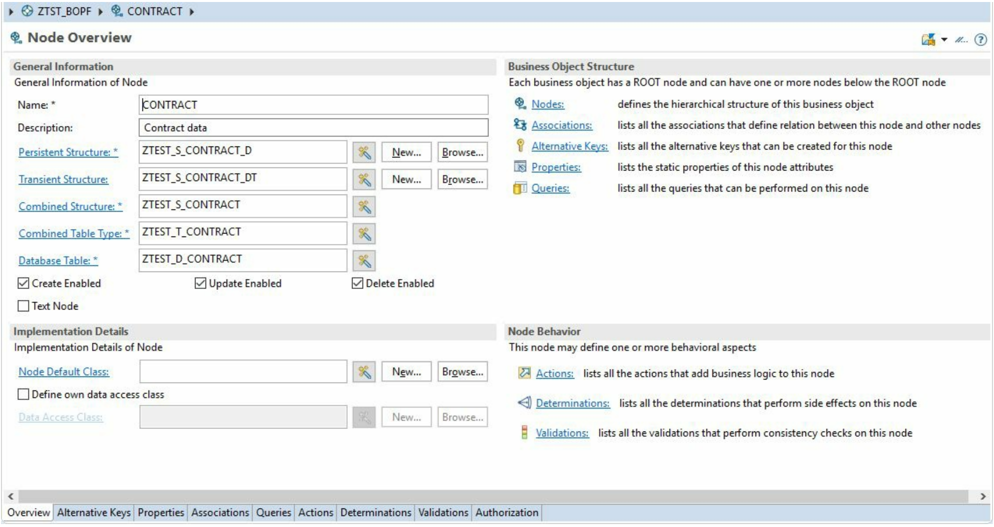
Business Obj
General Informatio
General Information
Name:
Description:
Category:
Constants Interface
What's Next
Proceed to next step
Go to the nodes of t

CONTRACT
CONDITIONS
PARTNERS
RULES
CONDITIONS -> ZTST_BOPF~>CONDITIONS
PARTNERS -> ZTST_BOPF~>PARTNERS
RULES -> ZTST_BOPF~>RULES
SELECT_ALL : Select All
SELECT_BY_CREATION : Custom Query
SELECT_BY_ELEMENTS : Custom Query
PARTNER_SIGN : Single Node Instance
GET_CREDITOR_NAME : Undefined

Nodes

The overview of a node works similar to the SAP GUI, you get all information about the structures and tables at a glance. What you might notice are the different ways of navigating and calling the functions within the node.

- Navigation to the object works via the link in front of it
- Suggestions for the name via the button with the magic wand
- New objects are created using the "New" button
- Suggestions for other functions are divided into different categories



Node Overview

General Information

General Information of Node

Name: * CONTRACT

Description: Contract data

Persistent Structure: * ZTEST_S_CONTRACT_D

Transient Structure: ZTEST_S_CONTRACT_DT

Combined Structure: * ZTEST_S_CONTRACT

Combined Table Type: * ZTEST_T_CONTRACT

Database Table: * ZTEST_D_CONTRACT

☑ Create Enabled ☑ Update Enabled ☑ Delete Enabled

☐ Text Node

Implementation Details

Implementation Details of Node

Node Default Class:

☐ Define own data access class

Data Access Class:

Business Object Structure

Each business object has a ROOT node and can have one or more nodes below the ROOT node

Nodes: defines the hierarchical structure of this business object

Associations: lists all the associations that define relation between this node and other nodes

Alternative Keys: lists all the alternative keys that can be created for this node

Properties: lists the static properties of this node attributes

Queries: lists all the queries that can be performed on this node

Node Behavior

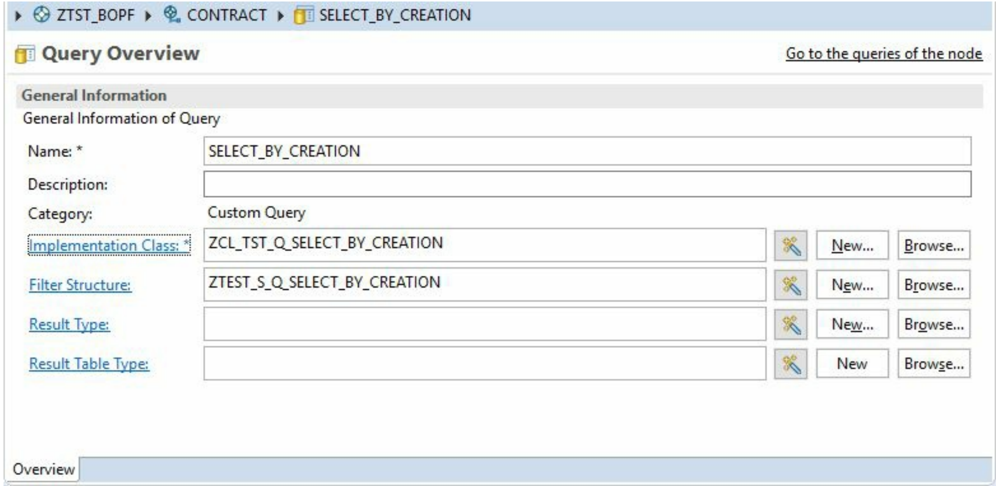
This node may define one or more behavioral aspects

Actions: lists all the actions that add business logic to this node

Determinations: lists all the determinations that perform side effects on this node

Validations: lists all the validations that perform consistency checks on this node

Let's take a look at our self-defined query, which was created directly under the contract. Here, as in the BOBX, we find information about the query, the structure used and our implemented class.



ZTST_BOPF ▸ CONTRACT ▸ SELECT_BY_CREATION
Go to the queries of the node
New...
Browse...
New
Overview

Query Overview

General Information

General Information of Query

Name: * SELECT_BY_CREATION

Description:

Category: Custom Query

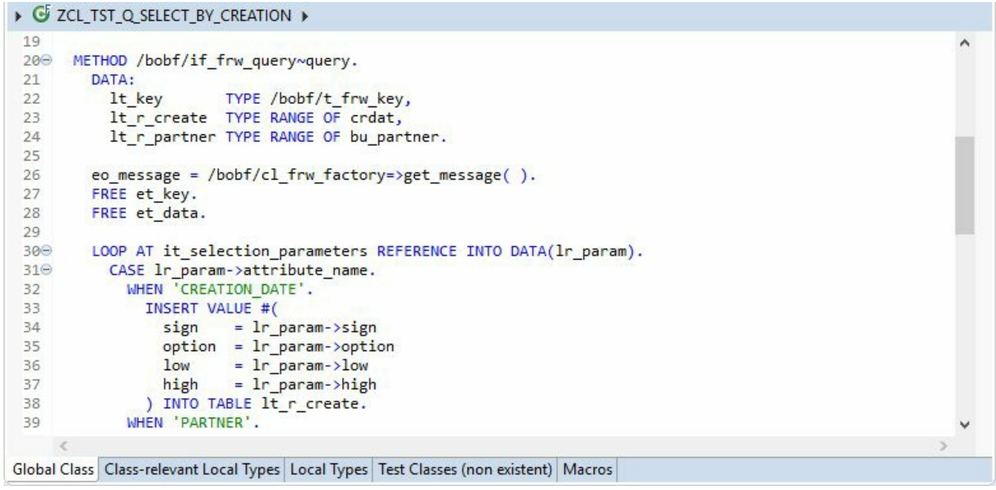
Implementation Class: * ZCL_TST_Q_SELECT_BY_CREATION

Filter Structure: ZTEST_S_Q_SELECT_BY_CREATION

Result Type:

Result Table Type:

Another click on the link in front of the class opens another tab and we are right in the class and could now adjust the methods and definitions.



ZCL_TST_Q_SELECT_BY_CREATION ▸
Global Class
Class-relevant Local Types
Local Types
Test Classes (non existent)
Macros

```

  METHOD /bobf/if_frw_query~query.
    DATA:
      lt_key       TYPE /bobf/t_frw_key,
      lt_r_create  TYPE RANGE OF crdat,
      lt_r_partner TYPE RANGE OF bu_partner.

    eo_message = /bobf/cl_frw_factory=>get_message( ).
    FREE et_key.
    FREE et_data.

    LOOP AT it_selection_parameters REFERENCE INTO DATA(lr_param).
      CASE lr_param->attribute_name.
        WHEN 'CREATION_DATE'.
          INSERT VALUE #(
            sign    = lr_param->sign
            option  = lr_param->option
            low     = lr_param->low
            high    = lr_param->high
          ) INTO TABLE lt_r_create.
        WHEN 'PARTNER'.
```

Ending

You have learned a lot in the last few chapters and already followed the example? Then you are ready for the “real” world out there. For the implementation of your data model, it is best to always use the environment that you can handle the best.

Would you like to learn more about SAP and ABAP development? Then take a look at our website www.software-heroes.com and follow our weekly blog to that topic.