



SAP - BOPF Developer Guide | CUSTOMER
%NW-ASABAP-LONG-7.54% FPS00
Document Version: 3.4 – 2019-09-20

SAP (On-Premise) - BOPF Developer Guide

Client Version 3.4

Content

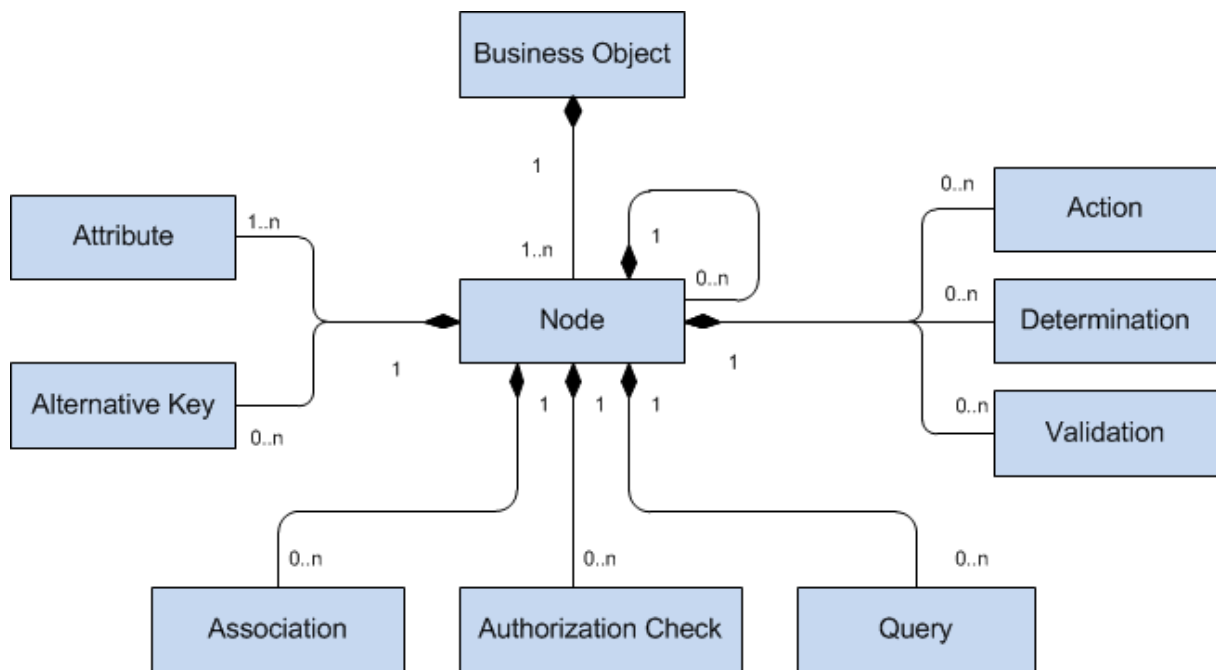
1	Business Object Metadata Model	4
1.1	Action.	5
1.2	Alternative Keys.	7
1.3	Association.	9
1.4	Authorization Checks.	12
1.5	Determination.	14
1.6	Nodes.	17
1.7	Query.	20
1.8	Validation.	54
1.9	Working with the BOPF API.	56
	Understanding the Action API.	59
	Understanding the Association API.	63
	Understanding the Authorization API.	67
	Understanding the Determination API.	73
	Understanding the Node API.	77
	Understanding the Query API.	81
	Understanding the Validation API.	86
2	BOPF - API Appendix.	90
2.1	Action Context IS_CTX.	90
2.2	Authorization Context IS_CTX.	91
2.3	Determination Context IS_CTX.	92
2.4	Reading Data Interface.	93
	Method CONVERT_ALTERN_KEY.	94
	Method GET_ROOT_KEY.	96
	Method RETRIEVE.	97
	Method RETRIEVE_BY_ASSOCIATION.	98
	Method COMPARE.	101
2.5	Changing Data Interface.	102
	Method CREATE.	103
	Method DELETE.	106
	Method DO_ACTION.	107
	Method DO_MODIFY.	108
	Method END_MODIFY.	108
	Method NOTIFY_CHANGE.	110
	Method NOTIFY_ASSOCIATION_CHANGE.	111
	Method NOTIFY_PROPERTY_CHANGE.	112

	Method UPDATE.	113
2.6	Changing Object Interface.	115
	Method GET.	116
	Method GET_CHANGES.	117
	Method HAS_CHANGES.	119
2.7	Message Object Interface.	120
	Method ADD.	121
	Method ADD_CM.	123
	Method ADD_CM_WITH_NEW_LOCATION.	124
	Method ADD_EXCEPTION.	125
	Method ADD_MESSAGE.	126
	Method CHECK.	127
	Method GET.	127
	Method GET_MESSAGES.	128
2.8	Semantics of Parameter IV_BEFORE_IMAGE.	129
3	Glossary.	131
4	References.	133

1 Business Object Metadata Model

In BOPF, a **business object** is represented as a **hierarchical** tree of nodes. A single node includes a set of semantically related attributes and the corresponding business logic. For each node, several types of entities can be defined to describe the specific business logic part of the business object.

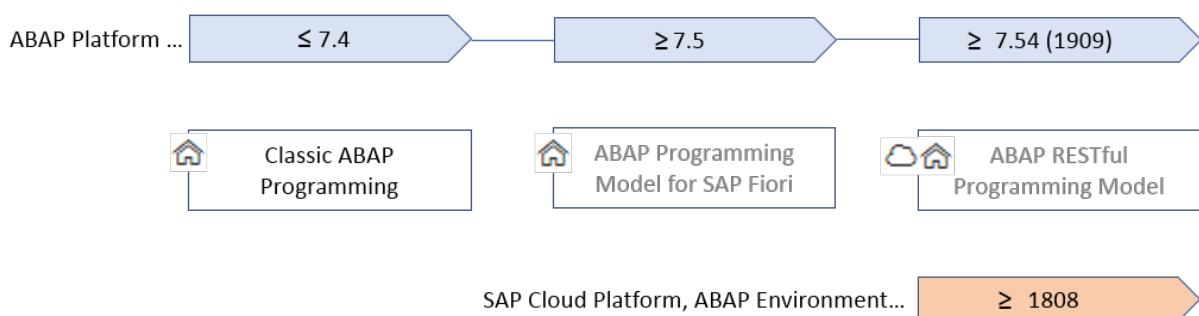
The following figure subsumes the elements of the business object metadata model and displays their relationship with each other.



Elements of the business object metadata model

Evolution of the Programming Model

Classification of BOPF within the Evolution of ABAP Programming Model



-
-

The *Classic ABAP Programming* is characterized by freestyle ABAP programming.

With the ABAP release 7.4, SAP started to heavily optimize the ABAP platform for its in-memory database SAP HANA. In this context ABAP Core Data Services (CDS) were introduced to offer the next generation data modeling infrastructure and data access. CDS-based data models are semantically rich data models that can be used for all application domains. Additionally, SAP Gateway became an integral part of the AS ABAP with release 7.4 and the **Business Object Processing Framework (BOPF)**. With release 7.5, BOPF also found its way into the ABAP platform.

i For more information about the evolution of the ABAP programming model, read this [blog](#) on the community portal.

Related Information

[Action \[page 5\]](#)

[Alternative Keys \[page 7\]](#)

[Association \[page 9\]](#)

[Authorization Checks \[page 12\]](#)

[Determination \[page 14\]](#)

[Nodes \[page 17\]](#)

[Validation \[page 54\]](#)

[Query \[page 20\]](#)

1.1 Action

An action is an entity assigned to an individual node of a business object that is used to implement a service (operation or behavior) of the business object.

Examples

You can use the **RELEASE** action to change the status of an invoice instance of the *CUSTOMER_INVOICE* business object. As a result of this action execution, the status of the invoice can be set to **RELEASED**. Other examples of an action

- **RELEASE ORDER**
- **COPY SHEET**
- **PUBLISH SALES_QUOTE**

Triggers for Actions

In order for an action to be executed, the corresponding triggers are required. Actions are called actively either by

- A service consumer, for example, by a User Interface
- Another business object
- Internally, for example, by another action or by a determination.

Categories of Actions

In general, BOPF provides the following categories for actions:

- **BO-Specific Actions**
BO-specific actions must be created explicitly for the node that requires a specific service implementation. Such actions are called by a service consumer or another business object.
- **Framework Actions**
Framework actions provide each BO node with core-services for which no implementation is required. They are automatically generated by BOPF when a node of a business object is created and are used internally by the framework itself. For each node, a framework action for create, update, delete, lock, unlock, validate and save is generated. Please note that these actions are not visible in the node tree of the design-time but some of them are offered actions to be prevented by an action validation.

Action Cardinality

You can perform an action on one or more node instances, or on no instance of the node. For this very purpose, you can assign the cardinality to an action. The action cardinality defines how many node instances the action can operate on during one action call. You can choose the following cardinality types for actions:

- **Multiple Node Instances**
Action operates on one or more node instances.
- **Single Node Instance**
Action operates on exactly one single node instance.
- **No Node Instances**
Action does not operate on any node instances and defines a static action.

Action Implementation

An action is implemented in a global ABAP class that implements the corresponding action interface **/BOBF/IF_FRW_ACTION**.

When calling an action, the consumer must provide the following specifications:

- The **key of node instances** on which the action operates. (This specification only applies when the action operates on one or more instances.) An action can only be performed with exactly that number of instances that is configured in the cardinality of the action.

i Note

To prevent an action from operating on certain specific instances, you can define an action validation.

- Any **input parameters** that the action requires.

i Note

This specification only applies when the action has a parameter structure in its definition.

In general, an action can have several importing parameters that the consumer passes to the implementation class during runtime. All parameters are included in the so-called parameter structure.

More on this:

[Understanding the Action API \[page 59\]](#)

Action Configuration

In addition, you can configure each action with an exporting type. Using the exporting type parameter, you can specify whether and if so what kind of data the action should be able to return to the caller during the action execution.

There are three options available:

- **None** - This default option indicates that the action does not return any data to the consumer
- **Type** - This option allows the action to return a data table defined by the row type maintained in the field [Exporting Parameter Structure](#) and the corresponding table type maintained in the field [Exporting Parameter Table Type](#).
- **Node** - This option allows the action to return instances of a foreign node that is defined by the field [Exporting Parameter Node](#) and [Exporting Parameter BO](#).

1.2 Alternative Keys

In BOPF, alternative keys are used to identify the node instances of business objects.

Alternative keys provide human-readable values for keys and are used, as the name implies, as an alternative to the technical, non-readable UUIDs of node instances. An alternative key is defined by an attribute or a set of attributes of a node that uniquely identifies the corresponding subordinated node instances.

→ Tip

In general, multiple alternative keys can be defined for a single business object node.

Example 1

Let us assume that our customer invoice application needs to be identified by a human-readable invoice number (1 – 100.000) for each single invoice. The key of the corresponding invoice node instance 0050562501281DDF97848577AF34366C obviously does not meet this requirement. Therefore, a node's attribute INVOICE_NUMBER that is intended to identify node instances perfectly represents an unstructured unique alternative key. Consumers can query the technical key of a node instance using a certain invoice number (for example, 4711).

Example 2

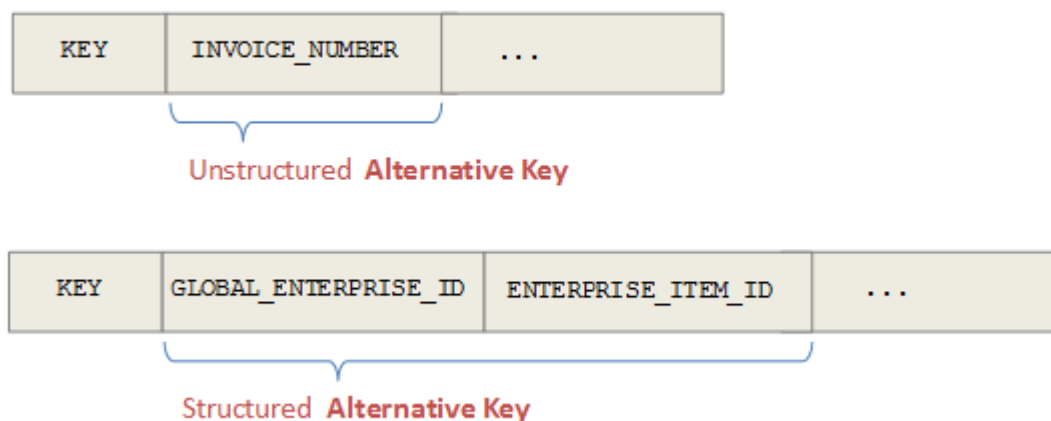
Assuming that a customer invoice instance has been created during the current transaction, you cannot look up the key of this instance using a query. In such a case, however, you can use an alternative key instead. Unlike queries, alternative key calls can be executed on BO data that have not been committed to the database yet.

Classification of Alternative Keys

You can classify alternative keys as:

Structured and Unstructured

Alternative keys can be related to a single attribute (unstructured alternative key) or to multiple node attributes (structured alternative key) of the same node.



Alternative Key

Uniqueness

Alternative keys can also be classified according to the uniqueness of their values:

- **Unique:** This classification ensures that the value of the alternative key is unique (for example: there can be only one invoice root node instance with the invoice number 4711).
- **Unique if attributes not initial:** This classification ensures that the value of the alternative key is unique if the value is not initial (not null). This means:
 - For unstructured alternative keys : each alternative key value that is not initial is always unique, but there can be some initial alternative keys, too.

- For structured alternative keys: alternative key values are unique if all of their fields are filled with (non-initial) values
- **Not unique:** This classification ensures that more than one node instance can have identical alternative key values.

Getting Alternative Keys

Similar to queries, alternative keys provide an initial point of access for the consumer to look up business object node instances. Unlike queries, alternative key calls can be executed on the [Transactional Image \[page 132\]](#)

The following method call can be used by an **internal consumer** (for example, from within an action implementation) to retrieve alternative keys:

```
/BOBF/IF_FRW_READ->CONVERT_ALTERN_KEY ( )
```

In the following method call, the transaction service manager is used. This can be triggered by an **external consumer** - by a REPORT or a UI component, for example :

```
/BOBF/IF_TRA_SERVICE_MANAGER->CONVERT_ALTERN_KEY ( )
```

More on this: [Method CONVERT_ALTERN_KEY \[page 94\]](#)

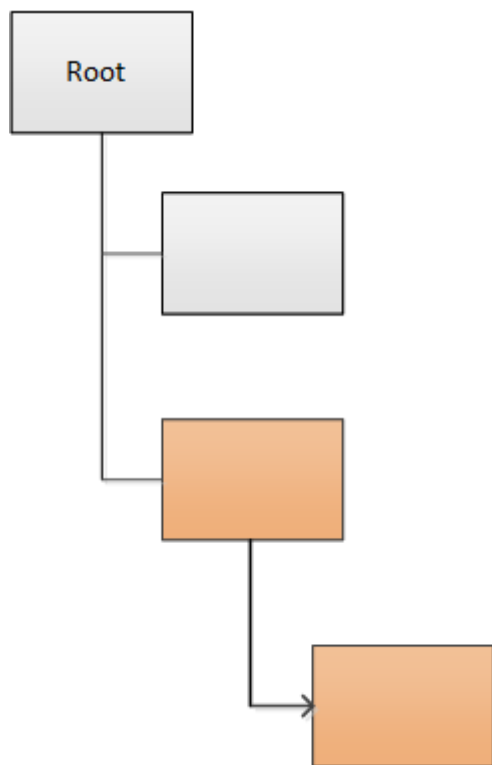
1.3 Association

In BOPF, an association is used to define a direct and unidirectional relationship between two nodes.

You can use an association to navigate from a **source node** to a related **target node**. You can define associated nodes in one or more business objects or business object enhancements. If you define associated nodes in more than one business object, you define cross-business-object associations. In this way, associations allow you to integrate BOs together in complex assemblies - according to the LEGO principle. In addition, you can use associations to create new node instances of subnode.

Association Type and Category

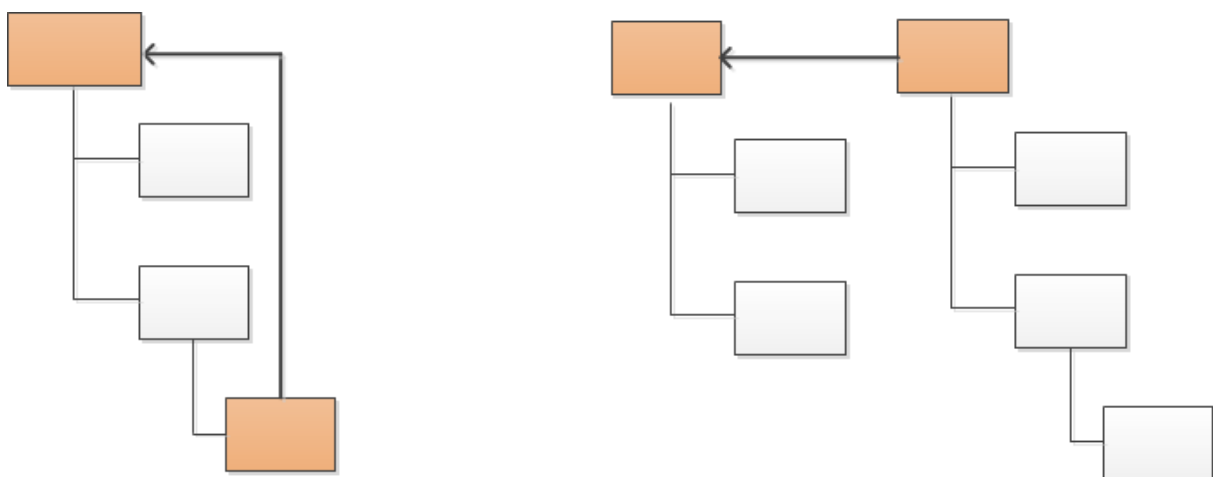
BOPF provides two different types of associations:



Composite Associations

Composite associations define the relationship from a parent node to a child node within the same tree node hierarchy. This type of association includes the following categories:

- Regular composition association
- Specialization
- Association with delegated node (dependent object)
- Framework associations (link to framework nodes)



General Associations

This type of association defines the relationship between a source node and a target node across the tree node hierarchy of one and the same BO or even between nodes of different BO trees (cross-BO association). The general associations include the following categories:

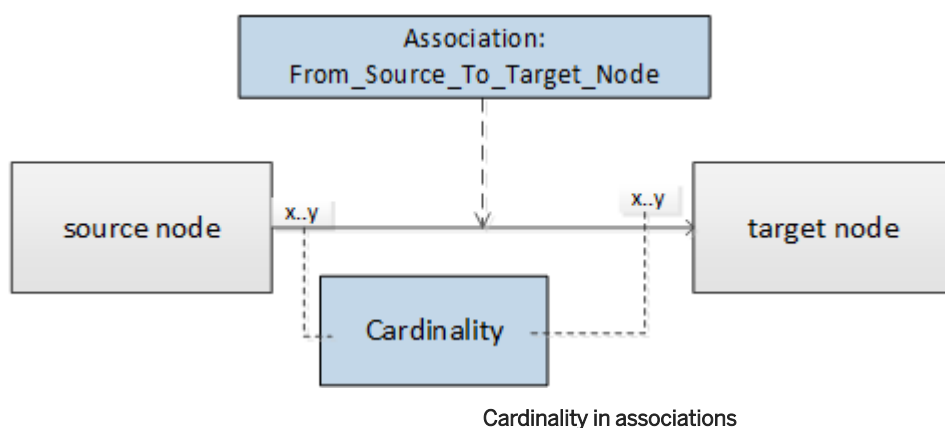
- Foreign key association / reverse foreign key association
- Reverse specialization
- Association “*To_Parent*”
- Association “*To_Root*”
- Cross-BO association
- implementation-specific association

Cardinality of Associations

The cardinality of an association expresses how the node instances may be involved in the relationship. It specifies the number of node instances that are connected to a single source instance and is expressed with a lower bound and an upper bound in the form: *x..y* (*lower_bound..upper_bound*).

You can assign the following values for an association's cardinality between the source and the target node:

- 1 : 0..1
- 1 : 1
- 1 : 0..n
- 1 : 1..n



Modeling or Implementing Associations

You can provide the association's logic in two different ways:

Using Association Binding (Recommended!)

The standard way to provide associations with business logic is to bind associations. This modelling approach is done by declaration steps without any implementation effort.

Implementation

In some cases, however, it can happen that very specific business logic is needed that is not covered by modelling capabilities. In such a case, the implementation of a specific association is necessary. The association is encapsulated within an ABAP class that implements the association interface *BOBF/IF_FRW_ASSOCIATION*.

More on this: [Understanding the Association API \[page 63\]](#)

Using the Association API

The method call can be used by an **internal consumer** (for example, out of an action implementation) to retrieve node keys and node data by following an association:

```
/BOBF/IF_FRW_READ->RETRIEVE_BY_ASSOCIATION( )
```

In the following call of the *RETRIEVE_BY_ASSOCIATION()* method, the service manager is used and can be triggered by an **external consumer** - by a REPORT or a UI component, for example:

```
/BOBF/IF_TRA_SERVICE_MANAGER->RETRIEVE_BY_ASSOCIATION( )
```

Implementing an Association Class

The association is encapsulated within an ABAP class that implements the association interface *BOBF/IF_FRW_ASSOCIATION*.

More on this: [Understanding the Association API \[page 63\]](#)

1.4 Authorization Checks

Business applications require an authorization concept for their data and for the operations on their data. Consequently, display and update activities are allowed for authorized users only.

In BOPF, an authorization check is introduced as an entity of a business object node that is used for managing authorizations on the node level. Using this authorization and role concept allows you to check whether a user is allowed to perform a certain activity on the node instances or to manipulate certain node data.

Examples

Let us assume that in a customer invoice application the invoices with `buyer_region AMERICAS` must be visible solely for authorized users. In addition, only authorized users should be able to change the status of invoices to `PAID` by executing the `INVOICE_PAID` action.

Generic Authorization Concept in BOPF

BOPF provides a generic authorization concept for your applications built on top of business objects so that each display and update activity can be protected by an authorization check.

Therefore, your application only has to...

1. Create an authorization object
2. Assign the authorization object to the appropriate business object node(s)
3. Map the authorization fields to the appropriate node data attributes

Note

In your application, you do not have to implement any specific authorization check coding as long as you accept the generic authorization concept.

What does the generic concept stand for ?

The generic authorization concept in BOPF is based on the well-known concept of authorization objects and the `AUTHORITY-CHECK` ABAP statement. However, the authorization objects that are going to be used in BOPF have to provide a specific field pattern:

Authorization objects that are assigned to the respective node have to meet the following requirements:

- The first field of the authorization object has to be `ACTVT`.
- The second field has to be `BO_SERVICE`.
- Further fields to be checked can be defined for the authorization object. All additional fields should be mapped to attributes of the respective BO node.

The screenshot shows the SAP authorization object configuration interface. The 'Object' field is set to 'Z_CI_AUT'. The 'Text' field contains 'Authorization object for BO Customer Invoices'. The 'Class' field is set to 'BOBF' with the description 'Authorizations for Business Objects'. The 'Author' field is empty. Below these fields is a table titled 'Authorization fields' with two columns: 'Authorization Field' and 'Short Description...'. The table contains three rows: 'ACTVT' with description 'Activity', 'BO_SERVICE' with description 'BO service name for authorization checks', and 'B_REGION' with description 'CI buyer region'. A fourth row is highlighted in yellow and contains an empty field and a small icon.

Object	Text	Class	Author
Z_CI_AUT	Authorization object for BO Customer Invoices	BOBF Authorizations for Business Objects	

Authorization fields	
Authorization Field	Short Description...
ACTVT	Activity
BO_SERVICE	BO service name for authorization checks
B_REGION	CI buyer region

Alternative Key

Basically, all BOPF core service requests (`RETRIEVE`, `RETRIEVE_BY_ASSOCIATION`, `CONVERT_ALTERN_KEY`, `MODIFY`, `DO_ACTION`, `QUERY`,...) are relevant for authorization checks and therefore can be checked for authorization through the implementation of generic authorization.

The generic authorization concept differentiates between static and the instance-based authorization checks:

- **Static checks:** Check whether the user has permission to perform a specific activity (for example: `DISPLAY`). The result of a static check is independent from specific data values.

- **Instance-based checks:** Evaluate the node data and check whether the user has permission to display or change data where an authorization-relevant attribute has a specific value. The result of this check depends on the node data.

Therefore, the first invoice example where `buyer_region` `AMERICAS` must be visible solely for authorized persons is handled by an instance-based check, whereas the second example (invoices can be changed to status `PAID` solely by authorized persons) is handled by a static check.

The static check utilizes the first field `ACTVT` and the second field `BO_SERVICE` of an authorization object. The instance-based check utilizes all authorization fields, especially the application-specific ones like `B_REGION`.

Propagation to Sub Nodes

Even if authorization objects and fields are assigned to a specific node within the BO hierarchy tree, at runtime the static and instance-based checks will be propagated along the composition tree. Authorizations of higher-level nodes apply implicitly to their compositions. For example, authorization checks that are configured for the `ROOT` node will be propagated to `ITEM` node. Therefore, the user is not allowed to display `ITEM` instances if he or she does not have display authorization for the `ROOT` node.

Runtime

The generic authorization runtime implementation consists of BOPF service manager and authorization handler, and the authorization check library class. The latter one can be substituted by an application-specific check class.

Implementing Application-Specific Authorization Checks

More on this: [Understanding the Authorization API \[page 67\]](#)

1.5 Determination

In BOPF, a determination is an entity of a business object node that is used to provide functions that are automatically executed as soon as a certain **trigger condition** is fulfilled. A determination is triggered internally on the basis of changes made to the node instance of a business object. The trigger conditions are checked by BOPF at different points during the transaction cycle, depending on the determination times and the changing operations on the relevant node instances. For each determination, it is necessary to specify both the points of time and the changes that form the trigger condition. Changes can include creating, updating, deleting, or loading node instances. You can use a determination primarily to compute data that is derived from the values of other attributes. The determined attributes and the determining attributes of the trigger condition either belong to the same node or to different nodes.

Example

If there is a change to the quantity of several products (items) in an invoice object instance, the amount attribute (the overall price) for all relevant products has to be calculated again. To calculate this amount

automatically, you have to extend the standard change process for the invoice business object instances with the help of determination. The determination can then react to the creation, the modification, the deletion, or the loading of an invoice item by deriving new values for the amount field. For such an example, the solution might look as follows:

After changing the quantity of invoice items, the trigger conditions of the *CALCULATE_ITEM_AMOUNT* determination and the *CALCULATE_TOTAL_AMOUNT* determination are both fulfilled. The *CALCULATE_ITEM_AMOUNT* determination calculates the amount of the changed item (price x quantity), whereas the *CALCULATE_TOTAL_AMOUNT* determination sums up the amounts of all items to the total amount of the invoice.

Determination Time and Triggers

Depending on the use case in question, BOPF checks the triggers of a determination at several points during the transaction.

A determination time defines at what time in the transaction cycle the trigger condition of that determination should be evaluated. For example, the re-calculation of the invoice amount should take place every time after a modification is performed (determination time: *After Modify*), but only if there are instances of the ITEM node that were updated (trigger: *Update*).

The following list shows which trigger operations are evaluated at which determination times:

The determination time and the triggers defines the trigger condition

Trigger >

Determination Time	Create	Update	Delete	Load	Determine
After Modification	X	X	X		
Calculate Transient Attributes	X	X	X	X	
During Check and Determine					X
Before Save	X	X	X		
Draw Number During Save	X	X	X		
During Save	X	X	X		
After Successful Save	X	X	X		
After Cleanup Transaction	X	X	X		
After Failed Save	X	X	X		

Determination Dependencies

If, in the example above, you do not define any dependencies between the determinations, the system might calculate the total amount before the item amount. Therefore, you must specify the

CALCULATE_ITEM_AMOUNT determination as a predecessor of the *CALCULATE_TOTAL_AMOUNT* determination.

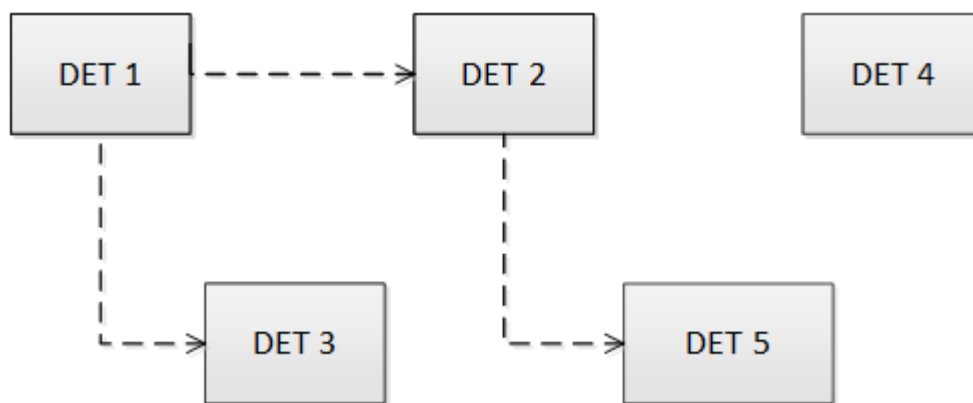
To specify whether a determination is a predecessor or a successor of another determination, you define a determination dependency. If there is more than one determination to be executed, the dependencies of the determinations are considered when BOPF checks the trigger conditions of determinations.

Example

The following dependency is defined in such a way that the item amount is calculated before the total amount.

Node: Determination	Dependency
ROOT: <i>CALCULATE_TOTAL_AMOUNT</i>	Determination depends on <i>CALCULATE_ITEM_AMOUNT</i>
ROOT > ITEM: <i>CALCULATE_ITEM_AMOUNT</i>	Determination must be executed before <i>CALCULATE_TOTAL_AMOUNT</i>

Dependency defines the execution order of determinations:



Dependencies between determinations

Examples of valid execution orders are:

4, 1, 3, 2, 5 (for example, the determination DET 2 is always executed before the execution of DET 5)

1, 2, 5, 4, 3 (for example, DET 1 is always executed before the execution of DET 2).

Determination Types

Depending on which category of nodes comes into play when executing determinations, there are two different determination types available:

- **Transient Determination** - Modifies only transient fields or nodes. For such determinations no locking is necessary.
- **Persistent Determination** - Modifies transient and persistent fields or nodes. Here locking is necessary.

Implementing a Custom Determination Class

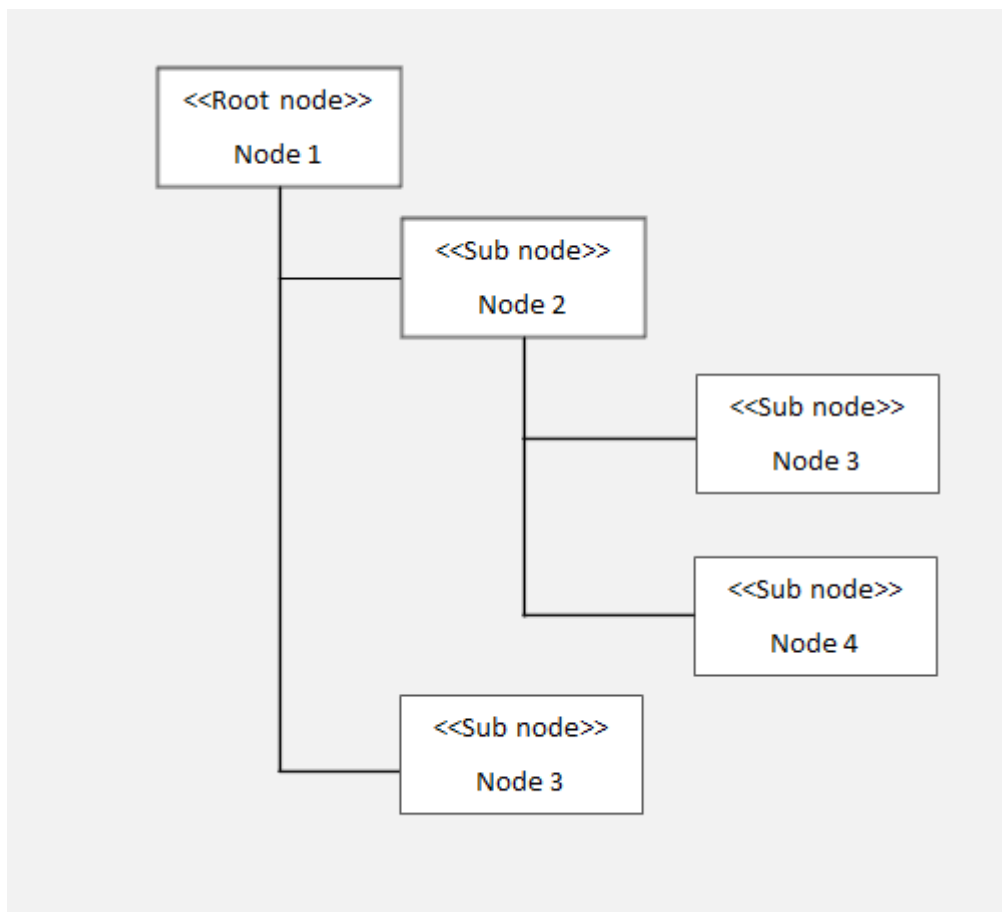
The determination logic is encapsulated within an ABAP class that implements the determination interface *BOBF/IF_FRW_DETERMINATION*.

More on this: [Understanding the Determination API \[page 73\]](#)

1.6 Nodes

A node is a data structure consisting of a set of attributes. Attributes are either persistent or transient. Persistent attributes are stored in a database table, whereas transient attributes are stored in the main memory of the system and are calculated during the user session inside a determination, or an action, for example.

Nodes are used to describe the data of a business object. A business object has exactly one root node with *n* subnodes. A subnode, in turn, can have *n* subnodes, and so on. The hierarchy of the nodes is described by composition associations. A node is identified by a [Technical Key \[page 132\]](#) and, where necessary, by further business identifiers.



Tree representation of a business object

Examples

A business object that represents a sales order consists of three nodes - one root node called ROOT with two subnodes called ITEM and NOTE:

- ROOT [SO_ID, BUYER_PARTY_ID, SO_STATUS, ...] - sales order header
 - ITEM [ITEM_POSITION, PRODUCT_ID, QUANTITY, ...] - sales order items
 - NOTE [NOTE] - sales order notes

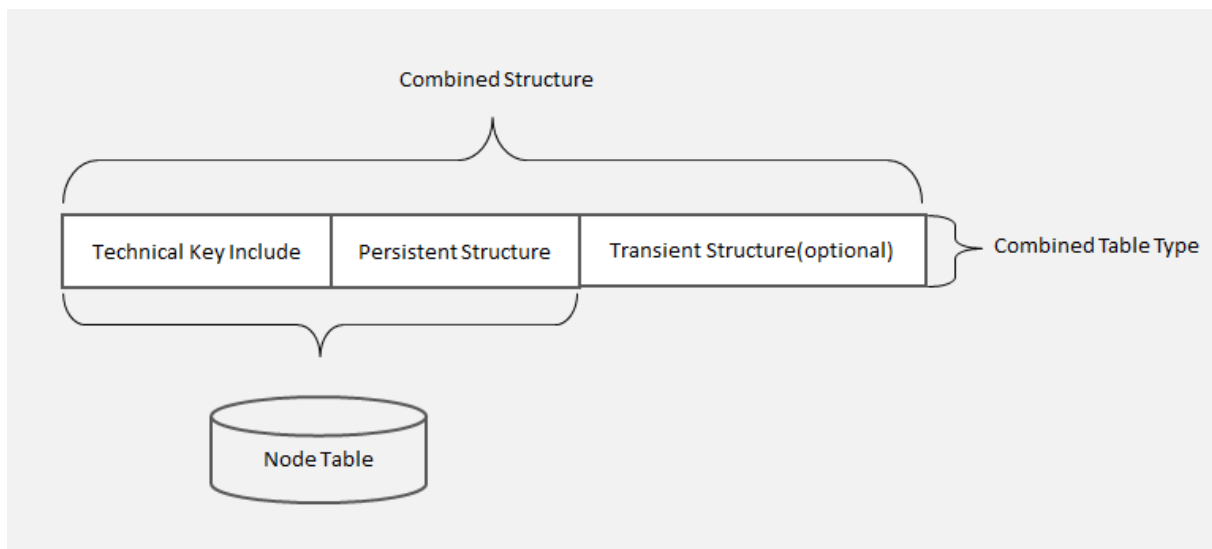
Node Artefacts

- **Persistent Structure:** Defines the business attributes that are stored on the database
- **Transient Structure:** Defines the business attributes that are calculated at runtime. The transient structure is optional.

Two further artefacts are generated automatically at design-time. They are relevant for framework processing and when BO services are consumed:

- **Combined Structure:** DDIC structure that includes a key structure (technical key), the persistent structure and the transient structure if one is defined.
- **Combined Table Type:** DDIC table type using the Combined Structure as the row type

Finally, a **database table** is generated and assigned to a node. The table stores the persistent attributes together with the corresponding technical keys and the client information.



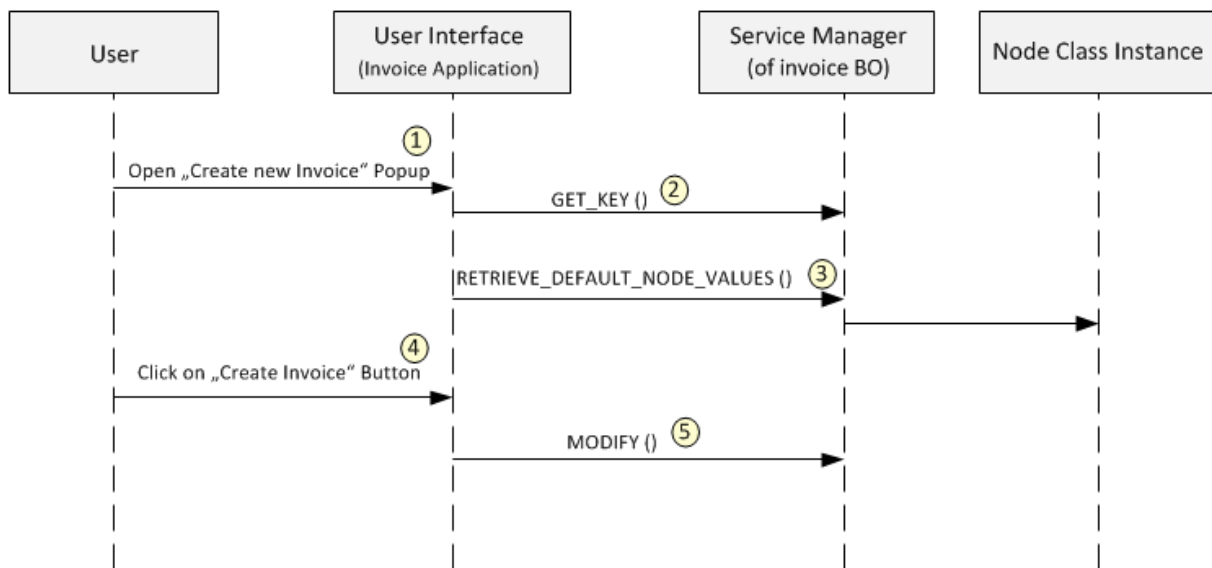
Node artefacts

Node Default Values

The default values of nodes can be important, especially in the UI context of BOPF-based applications. For example, the user should have access to predefined values for currencies in a form already when creating a new invoice.

Such defaulting allows the consumer to pre-fill the form that is related to a new node instance. To use node defaulting, the consumer must call the corresponding core services. The defaults can be consumed using the service manager. The service manager provides several services used to return default node values for nodes, but also for action parameters and association parameters. To calculate defaults of a certain node dynamically, a node class (class that is configured as *Node Class*) is used. The node class implements the `RETRIEVE_DEFAULT_VALUES` method of the node interface `/BOBF/IF_FRW_NODE`. Since the default node instances are only cached temporarily, the modification and then the save call is necessary to store the default values in the database.

Example



Sequence diagram for invoice creation based on node default values

Description

(1) The user of an invoice application opens a popup window to create an invoice. The corresponding UI provides a form for invoice creation.

(2) The service manager of the invoice BO provides the invoice application with a new unique key for the invoice node instance to be created.

(3) The invoice application retrieves the default values of this node instance key. The service manager calls the node class instance of the corresponding node and executes the method *RETRIEVE_DEFAULT_VALUES*. This method is implemented by the node class instance and is used to calculate the defaults of the related node instance.

(4) As a result of the last method execution, the export of the default data of the node instances to be defaulted is triggered. The default values of node attributes are returned and displayed on the form. The user populates the form with all values required for invoice creation and then (5) triggers a *MODIFY()* call to create the new instance. This modification call is necessary – otherwise the new node instance will not be saved at the end of the transaction.

More on this: [Understanding the Node API \[page 77\]](#)

1.7 Query

Queries provide the initial point of access to business objects. They are used by a consumer to look up business object instances that match certain search criteria.

In BOPF, a query is a business object entity that is always assigned to a certain node. Queries never modify any node instance data. The result of the query execution is a set of keys of all node instances that match the query criteria. In order to provide various types of search criteria, a suitable filter structure can be associated with a query.

Example

Let us assume that an application for editing customer invoices (BO: `CUSTOMER_INVOICE`) has to provide a screen for displaying all overdue invoices. This requirement however can be implemented by means of a query entity `GET_OVERDUE_INVOICES` located in the `CUSTOMER_INVOICE` business object that returns all overdue invoices in question.

Types of Queries

B.
a
i
l
d
w
iy
n
Q
u
e
r
io
e
s
u
p
t
h
e
n
o
d
e
i
n
s
t
a
n
c
e
s
w
i
t
h
t
h
e
h
e
l
p
o
-

f
t
h
e
a
t
t
r
i
b
u
t
e
v
a
l
u
e
s
.

T
h
e
r
e
a
r
e
t
w
o
b
u
i
l
t
-
i
n
q
u
e
r
i
e
s
a
-

available for nodes:

SELECT
—
ALL
-
This
is
query
every
y
is
a
u
t
o
m
a
t
—

i
c
a
l
l
y
g
e
n
e
r
a
t
e
d
f
o
r
a
n
o
d
e
b
y
t
h
e
B
O
P
F
f
r
a
m
e
w
o
r
k
w
h
e
n
t
h
e
—

n
o
d
e
i
s
c
r
e
a
t
e
d
.
I
t
r
e
t
u
r
n
s
a
l
l
n
o
d
e
i
n
s
t
a
n
c
e
s
o
f
t
h
e
n
o
d
e
—

t
o
w
h
i
c
h
t
h
e
e
q
u
e
r
y
i
s
a
s
s
i
g
n
e
d
.
F
o
r
t
h
i
s
b
u
i
l
d
-
i
n
q
u
e
r
y
,
t
-

her
e
r
e
i
s
n
o
fi
l
t
e
r
s
t
r
u
c
t
u
r
e
d
e
fi
n
e
d
f
o
r
s
e
l
e
c
t
i
o
n
.

S
E
L
E
C
T
—

—
B
Y

—
E
L
E
M
E
N
T
S
-
R
e
t
u
r
n
s
n
o
d
e
i
n
s
t
a
n
c
e
s
o
f
t
h
e
r
e
l
e
v
a
n
t
n
—

o
d
e
a
c
c
o
r
d
i
n
g
t
o
t
h
e
s
e
l
e
c
t
i
o
n
p
a
r
a
m
e
t
e
r
s
(
fi
l
t
e
r
s
t
r
u
c
t
-

u
r
e
)
.
E
a
c
h
a
t
t
r
i
b
u
t
e
o
f
t
h
e
n
o
d
e
i
n
q
u
e
s
t
i
o
n
c
a
n
b
e
u
s
e
d
a
s
—

a
s
e
l
e
c
t
i
o
n
p
a
r
a
m
e
t
e
r
.
T
h
e
r
e
f
o
r
e
,
y
o
u
c
a
n
u
s
e
t
h
i
s
k
i
n
d
-

o
f
b
u
i
l
t
-
i
n
q
u
e
r
y
w
h
e
n
e
v
e
r
y
o
u
o
n
l
y
n
e
e
d
t
o
s
e
a
r
c
h
f
o
r
B
O
n
-

o
d
e
s
b
y
t
h
e
i
r
a
t
t
r
i
b
u
t
e
s
(
f
o
r
e
x
a
m
p
l
e
,
f
o
r
a
n
o
d
e
l
D
=
,
1
2
3
—

4
5
,
)
.
—

C.
un
su
to
the
Q
u
e
re
ia
e
s
d
a
n
d
i
m
p
l
e
m
e
n
t
e
d
e
x
p
l
i
c
i
t
l
y
f
o
r
a
B
O
n
o
d
e

C
u
s
t
o
m
q
u
e
r
i
e
s
a
l
l
o
w
y
o
u
t
o
i
m
p
l
e
m
e
n
t
a
p
p
l
i
c
a
t
i
o
n
-
s
p
e

c
i
fi
c
q
u
e
r
i
e
s
f
o
r
a
n
y
u
s
e
c
a
s
e
.
Y
o
u
m
a
y
n
e
e
d
t
o
p
r
o
v
i
d
e
a
p
p
l

i
c
a
t
i
o
n
-
s
p
e
c
i
fi
c
q
u
e
r
i
e
s
i
f
t
h
e
l
o
g
i
c
o
f
t
h
e
b
u
i
l
d
-
i
n
q
u
e

r
i
e
s
i
s
n
o
t
s
u
ffi
c
i
e
n
t
.

**E
x
a
m
p
l
e**
:

G
E
T

—
O
V
E
R
R
D
U
E

—
I
N
V
O
I
C
E
S

F
o
r
c
u
s
t
o
m
q
u
e
r
i
e
s
,
y
o
u
m
u
s
t
c
o
d
e
t
h
e
e
n
t
i
r
e
q
u
e
r
y
l
o
g
i
c

w
i
t
h
i
n
a
g
l
o
b
a
l
A
B
A
P
c
l
a
s
s
t
h
a
t
i
m
p
l
e
m
e
n
t
s
t
h
e
i
n
t
e
r
f
a
c
e

/

B

O

B

F

/

I

F

—

F

R

W

—

Q

U

E

R

Y

.

I

n

a

d

d

i

t

i

o

n

,

y

o

u

h

a

v

e

t

h

e

o

p

t

i

o

n
t
o
a
s
s
i
g
n
a
n
a
r
b
i
t
r
a
r
y
fi
l
t
e
r
s
t
r
u
c
t
u
r
e
a
n
d
a
r
e
s
u
l
t
t
y
p
e

f
o
r
a
c
c
u
s
t
o
m
q
u
e
r
y
:

F
i
l
t
e
r
S
t
r
u
c
t
u
r
e
:
T
h
i
s
i
s
u
s
e
d
t
o
d
e
c

l
a
r
e
t
h
e
s
t
r
u
c
t
u
r
e
o
f
t
h
e
fi
l
t
e
r
c
r
i
t
e
r
i
a
u
s
e
d
i
n
a
q
u
e
r
y
.

**N
O
T
E**
:
T
h
i
s
s
t
r
u
c
t
u
r
e
c
a
n
b
e
u
s
e
d
f
o
r
t
h
e
r
a
n
g
e
v
a
l
u
e
s
e
l
e
c

t
i
o
n
b
y
a
g
e
n
e
r
i
c
U
l
t
o
d
i
s
p
l
a
y
a
n
i
n
p
u
t
f
o
r
m
t
h
a
t
i
s
b
a
s
e
d
o

n
a
r
a
n
g
e
t
a
b
l
e
.
A
s
s
o
o
n
a
s
t
h
e
q
u
e
r
y
i
s
e
x
e
c
u
t
e
d
a
n
d
t
h
e
f
o
r

m
i
s
fi
l
l
e
d
w
i
t
h
s
e
l
e
c
t
i
o
n
d
a
t
a
,
t
h
e
c
o
r
r
e
s
p
o
n
d
i
n
g
p
a
r
a
m
e

t
e
r
(

I
T

—
S
E
L
E
C
T
I
O
N

—
P
A
R
A
M
E
T
E
R
S
)
i
s
p
a
s
s
e
d
a
t
r
u
n
t
i
m
e

.
M
o
r
e
:
M
e
t
h
o
d
Q
U
E
R
Y
[
p
a
g
e
82
]
R
e
s
u
l
t
T
y
p
e
:
T
h
i
s
i
s
s
u
e
d
t
o

a
d
a
p
t
t
h
e
r
e
s
u
l
t
t
y
p
e
t
h
a
t
i
s
r
e
t
u
r
n
e
d
b
y
t
h
e
q
u
e
r
y
.

Using the Association API

The method call can be used by an external consumer (by a REPORT or a UI component, for example) to execute a query:

```
/BOBF/IF_TRA_SERVICE_MANAGER->QUERY ( ).
```

Implementing a Custom Queries

The query logic is encapsulated within an ABAP class that implements the query interface `/BOBF/IF_FRW_QUERY`.

More on this: [Understanding the Query API \[page 81\]](#)

1.8 Validation

In BOPF, a validation is an entity of a business object node that is triggered in certain situations to check various aspects of a given set of node instances. In particular, it is used to either validate whether a specific action can be executed on a given node instance (action validations) or whether a set of node instances is consistent (consistency validations).

Validations never modify any node instance data but return the messages and keys of failed (inconsistent) node instances.

Action Validations

Action validations check whether an action can be carried out.

Therefore, an action validation refers to an individual action and is used to determine whether or not this particular action can be executed on a set of node instances.

An action validation is executed when an action is called and before the action is executed. If a validation fails for certain instances, then the action will not be executed on all those failed instances. Action validations return only messages and the keys of node instances for which the action may not be executed.

Example

Using the RELEASE action of the *CUSTOMER_INVOICE* business object invoices can be set to the RELEASED status. However, before an invoice is released, it must be approved. If a consumer calls the RELEASE action, the *CHECK_RELEASE* action validation is automatically executed. It checks the node instances on which the RELEASE action has been called. Only instances that have the APPROVED status are processed by the RELEASE action.

Invoking Action Validations Explicitly

Since the BOPF framework invokes action validations automatically whenever actions are invoked by a consumer (report, UI), it is quite rare that you will have a need to invoke them directly. However, the `/BOBF/IF_TRA_SERVICE_MANAGER` interface provides the `CHECK_ACTION( )` method if you wish to do so:

Syntax:

```
/BOBF/IF_TRA_SERVICE_MANAGER->CHECK_ACTION( )
```

This method call checks whether an action can be executed or not.

Consistency Validations

Consistency validations check if a node instance is consistent regarding the consistency criteria imposed by the business requirements. Such validations are called at pre-defined points within the BOPF BO transaction cycle to ensure that BO nodes are persisted in a consistent state. These points can be configured for consistency validations. Each such validation configuration contains a trigger condition that is checked by BOPF at several time points during the transaction. If the trigger condition is fulfilled, the consistency validation is executed. Otherwise, if there are inconsistent node instances, a consistency validation behaves in one of the following ways:

- The validation sends messages to the consumer
- The validation sends messages to the consumer and prevents the transaction from being saved until the inconsistency is corrected
- The validation sends messages to the consumer and changes the consistency status.

Example

The `CUSTOMER_INVOICE` business object may contain several consistency validations to check if the items of an invoice are in a consistent state. For example, one consistency validation checks if the quantity and price of each item are maintained. Another consistency validation checks if the address of the buyer is valid. If it is not, the consistency validation prevents the transaction from being saved to ensure that only consistent invoice instances can be saved.

Invoking Consistency Validations Explicitly

The consistency validations are executed after each modification of node data, depending on the configured request nodes and when the consistency check is explicitly triggered (for example, by the UI).

To invoke a consistency validation explicitly, you can use the corresponding methods that are provided by the `/BOBF/IF_TRA_SERVICE_MANAGER` interface:

Syntax:

```
/BOBF/IF_TRA_SERVICE_MANAGER->CHECK_AND_DETERMINE( )
```

This method call performs a consistency check and also checks dependent determinations.

`/BOBF/IF_TRA_SERVICE_MANAGER->CHECK_CONSISTENCY( )`

This method performs the consistency check.

Implementing a Custom Validation Class

The validation logic is encapsulated within an ABAP class that implements the validation interface `/BOBF/IF_FRW_VALIDATION`.

More on this: [Understanding the Validation API \[page 86\]](#)

1.9 Working with the BOPF API

Although different types of BOPF entities need to implement different interfaces, they have common patterns and techniques for implementing business logic. These common patterns are described in this chapter.

Consuming Business Objects

The way for accessing business objects follows a service layer pattern. That means that a business object is abstracted by a set of common operations (such as `RETRIEVE`, `EXECUTE_ACTION`) that can be performed by its consumers. BOPF distinguishes between internal and external consumption and defines two kinds of APIs for this purpose:

- Internal Access API for BO internal access
- Service Manager API for accessing an external BO

Internal BO Access

- `IO_READ`: An object that provides reading access to the instances of the business object. The most common uses of this object are to read data using the `RETRIEVE` method or to navigate to an association using the `RETRIEVE_BY_ASSOCIATION` method.
More: [Reading Data Interface \[page 93\]](#)
- `IO_MODIFY`: An object that provides modifying access to the instances of the business object. This includes direct modifications by the methods `CREATE`, `UPDATE` and `DELETE` but also indirect modifications by the method `EXECUTE_ACTION`.
More: [Changing Data Interface \[page 102\]](#)

External BO Access

The so-called Service Manager API is used to consume services of an external business object.

Constants Interface

Since the APIs define common operations that are implemented by any business object, the signatures of these operations are typed generically and the consumer needs to specify the actual BO entity that is to be addressed by the actual operation. For example, in the signature of the RETRIEVE operation, the user needs to specify the actual node that is to be read. Or when calling EXECUTE_ACTION, the user needs to specify the action that is to be called and the corresponding node to which this action is assigned. A special constants interface plays a central role in specifying actual parts of a business object. The constants interface defines constants for all parts of a business object like nodes, actions, node attributes, and so on. Each single constant is represented by a GUID for the internal framework handling. Please note that it's not recommended to use the GUIDs directly in your business object code because GUIDs can change whenever the business object model is changed.

```
io_read->retrieve(  
    EXPORTING  
        iv_node = <ci>=>sc_node-root "Constant to identify the root node of your BO  
        ...  
    IMPORTING  
    ...).
```

Reading Data

Reading BO data is one of the main tasks of business logic. The corresponding operations (for example, RETRIEVE) have an exporting parameter called *ET_DATA* to receive the read data. Since this parameter is typed generically, it is important that the consumer is using the correctly typed variables in their implementation that fits the read data. Correctly typed means you have to use the Combined Table Type of the corresponding node.

```
DATA lt_root TYPE zcei_t_root. "Combined Table Type of the node  
io_read->retrieve(  
    EXPORTING  
        iv_node = <ci>=>sc_node-root  
        it_key  = it_key  
    IMPORTING  
        et_data = lt_root ).
```

Writing Data

Some of the BOPF entities - like actions and determinations - are able to change data. For a BO internal change the implementation would call the operations CREATE, UPDATE, or DELETE on the IAO IO_MODIFY. CHANGE

and UPDATE has a generically typed IS_DATA parameter to take over the new values of the node instance. You have to work with a data reference of the Combined Structure nodes in this case.

Messages

BOPF provides a new message container EO_MESSAGE for those entities that are meant to return messages. The most prominent entity is validation. The message concept is based on ABAP class-based messages. An entity that needs to throw a message instantiates a message object based on a class that inherits from /BOBF/CM_FRW and finally adds the message instance to the messages container EO_MESSAGE. BOPF takes care that the message is returned to the service consumer. Please note that you can't rely on EO_MESSAGE being already initialized when the entity is executed. Therefore you have to use the method /BOBF/CL_FRW_FACTORY=>GET_MESSAGE().

```
DATA lo_message type ref to zcm_my_messages "Class inherits from /BOBF/CM_FRW
IF problem found.
  CREATE OBJECT lo_message
  EXPORTING
    textid          = zcm_my_messages=>something_is_wrong
    severity        = zcm_my_messages=>co_severity_error
    ms_origin_location = ls_errr_location
    ... .
  IF eo_message IS NOT BOUND.
    eo_message = /bobf/cl_frw_factory=>get_message( ).
  ENDIF.
  eo_message->add_cm( lo_message )
  ENDIF.
```

When instantiating the message object, you have to specify the error text severity and an original location where you can provide specific information about the error location. For example, in the case of an inconsistent attribute value, the UI is able to point to the attribute that is inconsistent. BOPF message objects have some other parameters in the constructor that are described here.

Related Information

- [Understanding the Action API \[page 59\]](#)
- [Understanding the Association API \[page 63\]](#)
- [Understanding the Authorization API \[page 67\]](#)
- [Understanding the Determination API \[page 73\]](#)
- [Understanding the Node API \[page 77\]](#)
- [Understanding the Query API \[page 81\]](#)
- [Understanding the Validation API \[page 86\]](#)

1.9.1 Understanding the Action API

An action in BOPF is defined as a global ABAP class that implements the action interface `/BOBF/IF_FRW_ACTION`.

`/BOBF/IF_FRW_ACTION`

This interface defines methods for action execution in BOPF. Each ABAP class that represents an action in an application at runtime must implement this interface.

Method Summary

Deprecated

Do not implement the methods

- `RETRIEVE_DEFAULT_PARAM`
- `PREPARE`

Method	Description
<code>RETRIEVE_DEFAULT_PARAM</code>	<i>Deprecated!</i>
<code>PREPARE</code>	<i>Deprecated!</i>
<code>EXECUTE</code>	Carries out the action at runtime

Method Details

[Method RETRIEVE_DEFAULT_PARAM \[page 59\]](#)

[Method PREPARE \[page 60\]](#)

[Method EXECUTE \[page 60\]](#)

1.9.1.1 Method RETRIEVE_DEFAULT_PARAM

Retrieves the default parameters if a parameter structure is defined for the action

Deprecated

Do not use this method!

1.9.1.2 Method PREPARE

Allows you to enlarge the set of keys for which the action is executed

Deprecated

Do not use this method!

1.9.1.3 Method EXECUTE

Carries out the action at runtime

The execution of this **mandatory** method results in the possible modification of node data, both of the action's BOs and nodes that belong to other BOs.

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_ACT</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IO_READ</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_READ</i>
	<i>IO_MODIFY</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_MODIFY</i>
	<i>IS_PARAMETERS</i>	<i>type ref to</i>	<i>DATA</i>
<i>exporting</i>	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOPF/IF_FRW_MESSAGE</i>
	<i>ET_FAILED_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>ET_DATA</i>	<i>type</i>	<i>INDEX TABLE</i>
<i>raising</i>	<i>/BOBF/CX_FRW</i>		

Parameters

Parameter	Description
<i>IS_CTX</i>	Provides context information of the action More: Action Context IS_CTX [page 90]
<i>IT_KEY</i>	Set of keys of the node instances on which the action is to be performed

Parameter	Description
<i>IO_READ</i>	Provides read access to the business object More : Reading Data Interface [page 93]
<i>IO_MODIFY</i>	Provides write access to the business objects More : Changing Data Interface [page 102]
<i>IS_PARAMETERS</i>	Provides the parameters of the action call. These parameters are supplied by the service consumer.
<i>EO_MESSAGE</i>	Message container to return information, warning or error messages from the implementation
<i>ET_FAILED_KEY</i>	Set of node instance keys, for which the action execution failed i Note The parameter <i>ET_FAILED_KEY</i> must only be provided in the case of technical errors (for example of a not successful attempt to lock nodes).
<i>ET_DATA</i>	The exporting type parameter specifies whether and if so what kind of data the action should be able to return to the caller during the action execution.

Exceptions

Exception	Description
<i>/BOBF/CX_FRW</i>	Exception class that throws an exception when an error occurs It provides the user with BOPF-related message texts that are returned from T100.

Example

The following source code is used to implement the PUBLISH action of the business object *SALES_QUOTE*. A consumer can use this action to mark the sales quotation as published.

```
METHOD /bobf/if_frw_action~execute.
  DATA lr_root          TYPE REF TO zbo_s_root. " Combined structure of the BO
  DATA lt_changed_fields TYPE /bobf/t_frw_name.
  DATA ls_key           TYPE /bobf/s_frw_key.
  * To prepare update information, create the data reference for the root node
  CREATE DATA lr_root.

  * Set the ATTRIBUTE quote_status to new value
  lr_root->quote_status = 'P'. "Published

  APPEND zif_bo_sales_quote_c=>sc_node_attribute-root-quote_status TO
  lt_changed_fields.
  * Read the keys for all node instances that are processed by the action
  LOOP AT it_key INTO ls_key.

  * Use io_modify to update the node instance
  io_modify->update(
    EXPORTING
      iv_node      = zif_bo_sales_quote_c=>sc_node-root
      iv_key       = ls_key-key
      is_data      = lr_root
      it_changed_fields = lt_changed_fields ).
  ENLOOP.
ENDMETHOD.
```

Description

For the implementation of the PUBLISH action only the EXECUTE method is used here. As in most cases, the first step within the EXECUTE method is to retrieve relevant data of those node instances that are going to be changed. Remember, all attributes of the root node are represented by the combined structure's components.

For this purpose, a data reference for the root node *lr_root* is used, where *lr_root* refers to the combined structure *zbo_s_root*. This enables you to access the root node attribute *quote_status* and then set a new value 'P' that indicates that the node instance is set to status PUBLISHED.

In the second step, the internal table *lt_changed_fields* is populated with attributes that are going to be updated. Like all other entity names, the attribute names, too, are available through the Constants Interface *zif_bo_sales_quote_c*.

The keys of the node instances that need to be processed and thus need to be read are managed by the EXECUTE method's importing parameter *it_key*. It contains the keys of all node instances on which the action should be applied. For each relevant node instance, the update method of the access object *io_modify* is called.

1.9.2 Understanding the Association API

The association API interface is used by BOPF to resolve implemented associations and, if this association has the property “*create enabled*”, to create node instances. The association implementation itself belongs to the consumer application, which must implement the interface BOBF/IF_FRW_ASSOCIATION.

Both methods of this interface have to be implemented to prevent a runtime error. Each method implementation must include at least the METHOD/ENDMETHOD statements.

/BOBF/IF_FRW_ASSOCIATION

This interface defines the following methods for creating and resolving associations in BOPF.

Method Summary

Method	Description
CREATE	Creates an instance of a target node using an implemented association
RESOLVE	Resolves an association that has been implemented by the application

METHOD DETAILS

[Method CREATE \[page 63\]](#)

[Method RESOLVE \[page 65\]](#)

1.9.2.1 Method CREATE

Creates an instance of a target node using an implemented association

The execution of the method results in the creation of a node instance. When implementing the method, you also have the option to initialize certain fields of the node data with predefined values, such as indicator or type code fields.

Prerequisites:

This method is only called if the association has the property *create enabled*

Signature

importing	IS_CTX	type	/BOBF/S_FRW_CTX_ASSOC
	IT_MODIFICATION	type	/BOBF/T_FRW_MODIFICATION
	IO_READ	type ref to	/BOBF/IF_FRW_READ
	IO_MODIFY	type ref to	/BOBF/IF_FRW_MODIFY
exporting	EO_MESSAGE	type ref to	/BOPF/IF_FRW_MESSAGE
raising	/BOBF/CX_FRW		

Parameters

Parameter	Description
IS_CTX	Provides context information for associations
IT_MODIFICATION	Table type used for changes to be executed when creating an instance of a target node
IO_READ	Object that provides reading access to the instances of the business object More: Reading Data Interface [page 93]
IO_MODIFY	Object that provides writing access to the instances of the business object More: Changing Data Interface [page 102]
EO_MESSAGE	Message container to return information, warning, or error messages from the implementation

Exceptions

Exception	Description
<i>/BOBF/CX_FRW</i>	Exception class that throws an exception when an error occurs It provides the user with BOPF-related message texts that are returned from T100.

1.9.2.2 Method RESOLVE

Resolves an association that has been implemented by the application

The result of the method execution is a link table that assigns the target node instance keys to the source node instance keys.

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_ASSOC</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IO_READ</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_READ</i>
	<i>IS_PARAMETERS</i>	<i>type ref to</i>	<i>DATA</i>
	<i>IT_FILTERED_ATTRIBUTES</i>	<i>type</i>	<i>/BOPF/IF_FRW_NAME</i>
<i>exporting</i>	<i>ET_KEY_LINK</i>	<i>type</i>	<i>/BOPF/IF_FRW_KEY_LINK</i>
	<i>ET_FAILED_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOPF/IF_FRW_MESSAGE</i>
<i>raising</i>	<i>/BOBF/CX_FRW</i>		

Parameters

Parameter	Description
<i>IS_CTX</i>	Provides context information of the association

Parameter	Description
IT_KEY	Set of keys of the node instances for which the association is to be resolved
IO_READ	Object that provides reading access to the instances of the business object More: Reading Data Interface [page 93]
IS_PARAMETERS	Provides the parameters that are used when resolving the association between node instances These parameters are supplied by the service consumer. - <i>Optional</i> parameter
IT_FILTERED_ATTRIBUTES	List of field names that are relevant for association resolution - <i>Optional</i> parameter
ET_KEY_LINK	Link table that assigns the target node instance keys to the source node instance keys
ET_FAILED_KEY	Set of node instance keys (subset of <i>IT_KEY</i>) that do not match the criteria for association resolution
EO_MESSAGE	Message object used for returning information, warning, or error messages related to association resolution

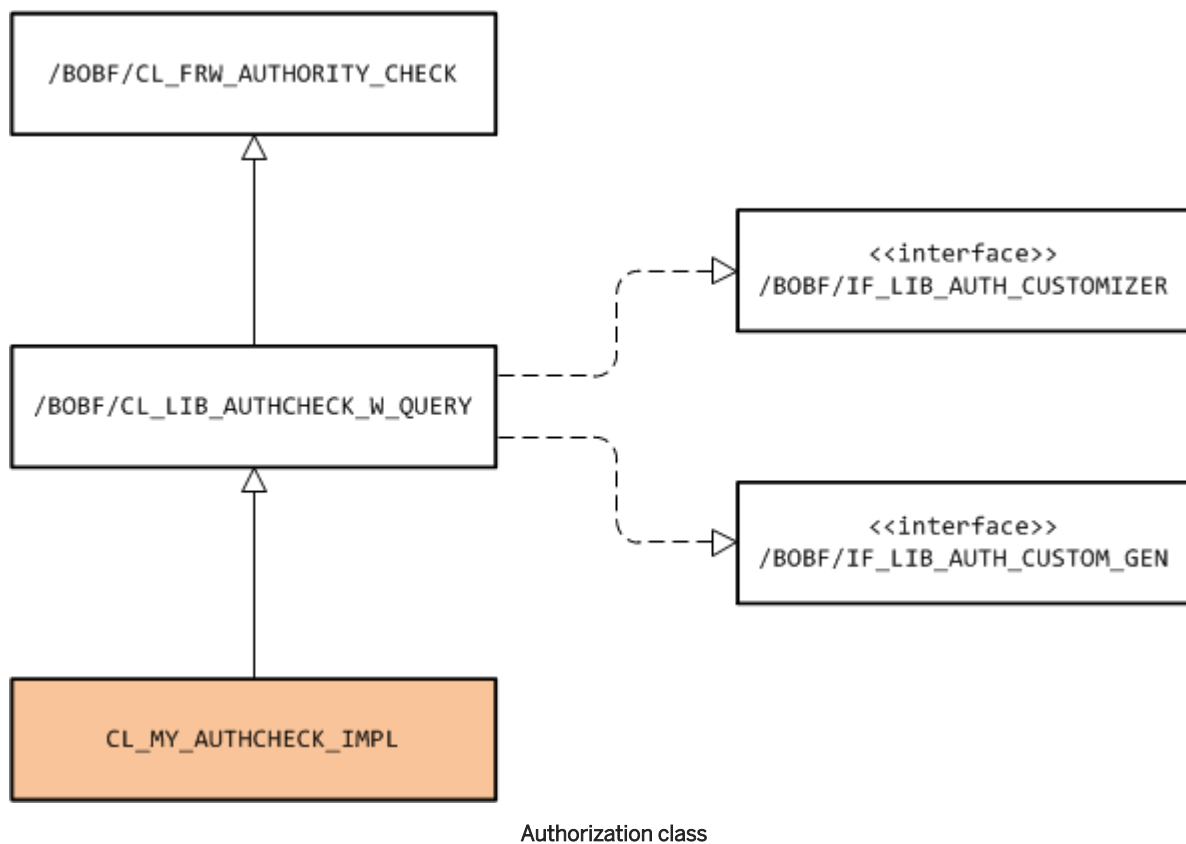
Exceptions

Exception	Description
/ BOBF/CX_FRW	Exception class that throws an exception when an error occurs It provides the user with BOPF-related message texts that are returned from T100.

1.9.3 Understanding the Authorization API

Extending the Generic Authorization Concept

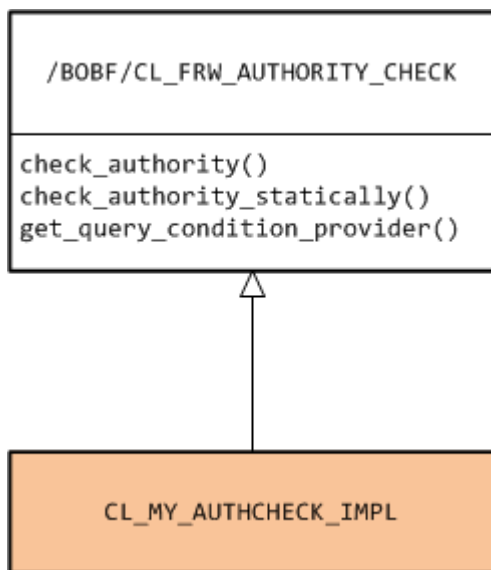
To redefine the behavior of the standard implementation of authorization checks, you can substitute the authorization check library class `/BOBF/CL_LIB_AUTHCHECK_W_QUERY` by an application-specific check class. In such a case, you must redefine the respective methods of the interfaces `/BOBF/IF_LIB_AUTH_CUSTOMIZER` and `/BOBF/IF_LIB_AUTH_CUSTOM_GEN`.



More on this : [Class /BOBF/CL_LIB_AUTHCHECK_W_QUERY \[page 68\]](#)

Implementing Own Authorization Checks

In rare cases, it might be necessary to implement a completely different authorization concept - for example, if the Access Control Lists are used instead of standard authorization objects. To achieve this, you can implement an application-specific class that inherits from the abstract authorization check superclass `/BOBF/CL_FRW_AUTHORITY_CHECK`.



Authorization checks

More on this: [Class /BOBF/CL_FRW_AUTHORITY_CHECK \[page 69\]](#)

1.9.3.1 Class /BOBF/CL_LIB_AUTHCHECK_W_QUERY

This class provides a default implementation of the authorization concept including queries. It inherits from the abstract class /BOBF/CL_FRW_AUTHORITY_CHECK.

Use of BO Metadata

The implementation takes the authorization-relevant metadata of the business object node into account. This includes both the assigned authorization objects and the authorization field mapping.

i Note

Authorization objects that are assigned to the respective node have to meet the following requirements:

- First field has to be ACTVT
- Second field has to be BO_SERVICE
- Additional fields can be defined; all additional fields should be mapped to attributes of the BO node.

Implementing Application-Specific Logic

/BOBF/CL_LIB_AUTHCHECK_W_QUERY is not defined as a final class. Therefore, subclasses can be derived to provide application-specific implementation. A new ABAP class that inherits from this must redefine the

respective methods of the interfaces `/BOBF/IF_LIB_AUTH_CUSTOM_GEN` and `/BOBF/IF_LIB_AUTH_CUSTOMIZER`.

In this way, you can change the runtime behavior (compared with the generic authorization concept), for example, by ...

- Redefining both static fields `ACTVT` and `BO_SERVICE` of the authorization object
Example: For querying node data, use activity display instead of query.
- Suppressing authorization propagation
Example: For retrieving items, check authority only on item node instead of item and root node
- Sending application-specific messages in the case of authorization denied

1.9.3.2 Class `/BOBF/CL_FRW_AUTHORITY_CHECK`

This **abstract** class is the superclass of the authorization check classes and acts as an interface definition when you implement your own authorization concept in BOPF.

Each application-specific authorization check class can inherit directly from this class (requires complete implementation) or from one of the subclasses (allows redefinition of particular methods).

A default implementation is provided with the class `/BOBF/CL_LIB_AUTHCHECK_W_QUERY`.

This class defines all instance attributes, constants, and methods for authorization check execution.

Instance Attributes

Instance Attribute	Description
<code>MV_BO_KEY</code>	Represents the BO key for which the authorization is configured
<code>MV_NODE_KEY</code>	Represents the (authority) node for which the instance of the authorization check class has been created
<code>MO_FACTORY</code>	Provides access to instances of another authorization check class in order to propagate authorization checks from subnodes to superior nodes

Constants

Constant	Description
SC_ACTIVITY	Specifies all activity types (create, change, display, ...) that are used by the framework in order to check the authorization
SC_ACTIVITY_SUPPLEMENT	Supplement that specifies extended context information about the situation in which the authorization check is called For example, the value RBA_SOURCE_NODE is set as a supplement if the RETRIEVE_BY_ASSOCIATION service is executed and authority has to be checked for the source node.
SC_AUTHORITY_FIELD	Specifies the BOPF-specific authorization fields (ACTVT and BO_SERVICE)

Method Summary

Method	Description
CHECK_AUTHORITY	Executes the static and instance-based authorization check
CHECK_AUTHORITY_STATICALLY	Executes the static authorization check
GET_QUERY_CONDITION_PROVIDER	Returns the authority condition provider to execute query based on SADL

Method Details

[Method CHECK_AUTHORITY \[page 71\]](#)

[Method CHECK_AUTHORITY_STATICALLY \[page 72\]](#)

[Method GET_QUERY_CONDITION_PROVIDER \[page 72\]](#)

1.9.3.2.1 Method CHECK_AUTHORITY

Executes the static and instance-based authorization check

This method is usually called by the BOPF framework when a core service is executed and the static authorization check has already been passed successfully.

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_AUTH</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IO_READ</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_READ</i>
<i>exporting</i>	<i>ET_FAILED_KEY</i>	<i>type</i>	<i>/BOBF /T_FRW_KEY</i>
	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOBF /T_FRW_MESSAGE</i>

Parameters

Parameter	Description
IS_CTX	Provides model information of the authorization (authorization context) More: Authorization Context IS_CTX [page 91]
IT_KEY	Set of keys for node instances for which the authorization check is to be executed
IO_READ	Reference that provides read access to the data of the business object instances More: Reading Data Interface [page 93]
ET_FAILED _KEY	Set of keys for node instances (subset of IT_KEY) that do not match the criteria for authorization checks
EO_MESSAGE	Message object used for returning information, warning, or error messages related to the authorization check execution

1.9.3.2.2 Method CHECK_AUTHORITY_STATICALLY

Executes the static authorization check

This method is usually called before the execution of a BOPF core service in order to check whether the user has authorization to execute the service at all.

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_AUTH</i>
<i>exporting</i>	<i>EV_FAILED</i>	<i>type</i>	<i>BOOL_D</i>
	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOBF /T_FRW_MESSAGE</i>

Parameters

Parameter	Description
<i>IS_CTX</i>	Provides model information of the authorization (authorization context) More: Authorization Context IS_CTX [page 91]
<i>ET_FAILED</i>	Set of keys for node instances (subset of <i>IT_KEY</i>) that do not match the criteria for authorization checks
<i>EO_MESSAGE</i>	Message object used for returning information, warning, or error messages related to the authorization check execution

1.9.3.2.3 Method GET_QUERY_CONDITION_PROVIDER

Returns the authority condition provider for executing a query based on SADL

This method returns a SADL query condition provider, which adds the authorization conditions to a query in order to enable execution of authorization checks directly on the database level. The query condition provider is needed for execution of generic queries for which no implementation class is defined (usually *SELECT_ALL* or *SELECT_BY_ELEMENTS* queries).

i Note

As a consequence, queries always have to be implemented if it is not possible to return a query condition provider. Such a situation may occur if the authorization checks are not completely modeled. An example for this would be an authorization field mapping that uses an implemented association. The implemented association cannot be resolved by the database. The query implementation has to take care of authorization checks in this case.

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_AUTH</i>
<i>returning value</i>	<i>RO_PROVIDER</i>	<i>type ref to</i>	<i>IF_SADL_COND_PROVIDER_AUTHS</i>

Parameters

Parameter	Description
IS_CTX	Provides model information of the authorization (authorization context) More: Authorization Context IS_CTX [page 91]
RO_PROVIDER	Instance of the SADL query authorization condition provider, which adds the authorization conditions to a query

i Note

SADL supports the concept of authorization condition providers. Such providers are objects that are responsible for reading the metadata of business objects, evaluating the authorizations of the active users, and transforming the result into suitable restriction information for the database selection.

1.9.4 Understanding the Determination API

A determination in BOPF is defined as a global ABAP class that implements the action interface */BOBF/IF_FRW_DETERMINATION*.

/BOBF/IF_FRW_DETERMINATION

This interface defines methods for action execution in BOPF. Each ABAP class that represents an action in an application at runtime must implement this interface.

The determinations are executed after each modification of node data, depending on the configuration. **See also:** [Determination Time and Triggers \[page 15\]](#)

Method Summary

Deprecated

Do not implement the methods

- *CHECK_DELTA*
- *CHECK*

Method	Description
<i>CHECK_DELTA</i>	<i>Deprecated!</i>
<i>CHECK</i>	<i>Deprecated!</i>
<i>EXECUTE</i>	Carries out the validation at runtime The method execution can modify the data of node instances.

METHOD DETAILS

[Method CHECK_DELTA \[page 74\]](#)

[Method CHECK \[page 75\]](#)

[Method EXECUTE \[page 75\]](#)

1.9.4.1 Method CHECK_DELTA

Checks the relevance of node instances for the execution of a determination based on changes made since the last determination run

Deprecated

Do not use this method!

1.9.4.2 Method CHECK

Checks the relevance of node instances for the execution of a determination based on attribute values, independently of change independently of changes

Deprecated

Do not use this method!

1.9.4.3 Method EXECUTE

Carries out the determination at runtime

This mandatory method implements the main determination logic. It provides modification access to the node's instance data and returns the messages and the keys of failed node instances (instances for which the determination cannot be executed).

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_DET</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IO_READ</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_READ</i>
	<i>IO_MODIFY</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_MODIFY</i>
<i>exporting</i>	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOBF/T_FRW_MESSAGE</i>
	<i>ET_FAILED_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
<i>raising</i>	<i>/BOBF/CX_FRW</i>		

Parameters

Parameter	Description
<i>IS_CTX</i>	Provides model information of the determination (determination context) More: Determination Context IS_CTX [page 92]
<i>IT_KEY</i>	Set of node instance keys that are to be processed by the determination

Parameter	Description
<i>IO_READ</i>	Reference that provides read access to the data of the business object instances More: Reading Data Interface [page 93]
<i>IO_MODIFY</i>	Reference that provides write access to the data of the business object instances More: Changing Data Interface [page 102]
<i>EO_MESSAGE</i>	Message object used for returning information, warning, or error messages related to the determination execution More: Message Object Interface [page 120]
<i>ET_FAILED_KEY</i>	Set of node instance keys (subset of <i>IT_KEY</i>) that do not match the determination criteria i Note The parameter <i>ET_FAILED_KEY</i> must only be filled in case of technical errors (for example, a not successful attempt to lock nodes). It cannot be used to control the transactional behavior like the execution of succeeding determinations.

Exceptions

Exception	Description
<i>/BOBF/CX_FRW</i>	Exception class that throws an exception when an error occurs It provides the user with BOPF-related message texts that are returned from T100.

Example

```

METHOD /bobf/if_frw_determination~execute.
  DATA lt_item_data TYPE          zci_t_item.
  DATA lr_item_data TYPE REF TO zci_s_item.
  * For the customer invoice retrieve its ITEM nodes
  io_read->retrieve(
    EXPORTING
      iv_node = zif_ci_customer_invoice_c=>sc_node-item
      it_key  = it_key
  )

```

```

IMPORTING
  et_data = lt_item_data ).
* Calculate the amount (= price x quantity) for each item
  LOOP AT lt_item_data REFERENCE INTO lr_item_data.
    lr_item_data->amount = lr_item_data->price * lr_item_data->quantity.

* Update the customer invoice with calculated amount
  io_modify->update(
    EXPORTING
      iv_node = zif_ci_customer_invoice_c=>sc_node-item
      iv_key  = lr_item_data->key
      is_data = lr_item_data ).
  ENDLLOOP.
ENDMETHOD.

```

Description:

In this sample code, the `execute()` method implements a determination that is called when data of a customer invoice item is updated.

The reading access object `io_read` is used to retrieve the item instances of the customer invoice business object. Within the `LOOP` statement, the new `amount` value is calculated for each relevant item instance by applying the formula: `amount = price x quantity`. To update the data of the invoice items for each relevant node instance, the update method of the access object `io_modify` is called.

1.9.5 Understanding the Node API

A node of a business object in BOPF is defined as a global ABAP class that implements the node interface `/BOBF/IF_FRW_NODE`.

`/BOBF/IF_FRW_NODE`

This interface defines methods for operating with node instances in BOPF.

You can use this interface to calculate default values of node attributes at runtime. Providing default values of nodes can be important, especially in the UI context of BOPF-based applications.

Method Summary

Obsolete

Do not implement the `SET_NODE_CATEGORY` method in the context of ABAP Development Tools!

Method	Description
<i>RETRIEVE_DEFAULT_VALUES</i>	Returns default values of a node instance
<i>SET_NODE_CATEGORY</i>	Obsolete!

METHOD DETAILS

[Method RETRIEVE_DEFAULT_VALUES \[page 78\]](#)

[Method SET_NODE_CATEGORY \[page 81\]](#)

1.9.5.1 Method RETRIEVE_DEFAULT_VALUES

Returns the default value set of a node instance

If the node instance belongs to a subnode, the association and its parameters *IS_ASSOC_PARAMETERS*, and the list of field names *IT_ASSOC_FILTERED_ATTRIBUTES* can also be specified.

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_ASSOC</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IS_ASSOC_PARAMETERS</i>	<i>type ref to</i>	<i>DATA</i>
	<i>IT_ASSOC_FILTERED_ATTRIBUTES</i>	<i>type</i>	<i>/BOBF/T_FRW_NAME</i>
	<i>IV_SOURCE_NODE_KEY</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_SOURCE_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IO_READ</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_READ</i>
	<i>IO_MODIFY</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_MODIFY</i>
<i>exporting</i>	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOBF/T_FRW_MESSAGE</i>
<i>raising</i>	<i>/BOBF/CX_FRW</i>		

Parameters

Parameter	Description
IS_CTX	Provides context information for the default values. (Default values of a node instance are context-dependent.)
IT_KEY	Table with technical keys of the node instances for which the default value set is to be retrieved - <i>Optional</i> parameter
IS_ASSOC_PARAMETERS	Parameters of an association that are used when creating node instances with default values - <i>Optional</i> parameter
IT_ASSOC_FILTERED_ATTRIBUTES	List of field names that are relevant for association resolution - <i>Optional</i> parameter
IV_SOURCE_NODE_KEY	Source node of the association - <i>Optional</i> parameter
IV_SOURCE_KEY	Source node key of the association - <i>Optional</i> parameter
IO_READ	Provides read access to the data of the business object More: Reading Data Interface [page 93]
IO_MODIFY	Provides write access to the data of the relevant business object instances More: Changing Data Interface [page 102]
EO_MESSAGE	Message object used for returning information, warning, or error messages related to node value retrieval

Exceptions

Exception	Description
<code>/BOBF/CX_FRW</code>	Exception class that throws an exception when an error occurs It provides the user with BOPF-related message texts that are returned from T100.

Example

The following source code demonstrates how this method is used to set a default currency code (in our example: EUR) as a default value when creating a new invoice.

```
METHOD /BOBF/IF_FRW_NODE~RETRIEVE_DEFAULT_VALUES.  
  DATA ls_key TYPE /bobf/s_frw_key.  
  DATA lt_to_fill TYPE zbo_t_root.  
  DATA ls_to_fill TYPE zbo_s_root.  
  DATA lv_ref_data TYPE REF TO zbo_s_root. " Combined structure of the BO  
  io_read->retrieve(  
    EXPORTING  
      iv_node = zif_bo_currency_code_c=>sc_node-root  
      it_key = it_key  
    IMPORTING  
      et_data = lt_to_fill ).  
  * Data reference that is typed with node's Combined Structure  
  DATA ls_fill TYPE REF TO zbo_s_root.  
  LOOP AT lt_to_fill INTO ls_to_fill.  
    ls_to_fill-currency_code = 'EUR'.  
  * Node instance data for update  
  CREATE DATA ls_fill.  
  ls_fill->* = ls_to_fill.  
  io_modify->update(  
    EXPORTING  
      iv_node = zif_bo_currency_code_c=>sc_node-root  
      iv_key = ls_to_fill-key  
      is_data = ls_fill ).  
  ENDLOOP.  
ENDMETHOD.
```

Description

The BO access object *io_read* provides read access to the node instances of the invoice business object by calling the *retrieve()* method. This method exports the data of the root node instances to be defaulted from the buffer.

All attributes of the root node are represented by the combined structure's components. This enables you to access the root node attribute *currency_code* and set the value 'EUR' as the default value for the node instance.

For each relevant node instance, the `update( )` method of the access object *io_modify* is called.

As a result of the method call, the node instance data is updated in the buffer.

1.9.5.2 Method SET_NODE_CATEGORY

Deprecated / Obsolete

Do not use this method!

1.9.6 Understanding the Query API

A query in BOPF is defined as a global ABAP class that implements the query interface `/BOBF/IF_FRW_QUERY`.

`/BOBF/IF_FRW_QUERY`

This interface defines methods for action execution in BOPF.

Method Summary

Method	Description
QUERY	Executes the query logic
RETRIEVE_DEFAULT_PARAMS	Returns the default values for parameters of a query - Optional method

METHOD DETAILS

[Method QUERY \[page 82\]](#)

[Method RETRIEVE_DEFAULT_PARAMS \[page 85\]](#)

1.9.6.1 Method QUERY

Executes the query logic

The result of the method execution is a set of keys of the node instances that match the selection parameters.

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_QUERY</i>
	<i>IT_FILTER_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IT_SELECTION_PARAMETERS</i>	<i>type</i>	<i>/BOBF/T_FRW_QUERY_SELPARAM</i>
	<i>IO_QUERY_AUTHORITIES</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_AUTHORITY_QUERY</i>
	<i>IS_QUERY_OPTIONS</i>	<i>type</i>	<i>/BOBF/S_FRW_QUERY_OPTIONS</i>
	<i>IO_QUERY</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_QUERY</i>
	<i>IO_READ</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_READ</i>
	<i>IO_MIDIFY</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_MODIFY</i>
	<i>IV_FILL_DATA</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>IT_REQUESTED_ATTRIBUTES</i>	<i>type</i>	<i>/BOBF/T_FRW_NAME</i>
<i>exporting</i>	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_MESSAGE</i>
	<i>ET_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>ES_QUERY_INFO</i>	<i>type</i>	<i>/BOBF/S_FRW_QUERY_INFO</i>
	<i>ET_DATA</i>	<i>type</i>	<i>INDEX TABLE</i>
<i>raising</i>	<i>/BOBF/CX_FRW</i>		

Parameters

Parameter	Description
<i>IS_CTX</i>	Provides the context information of a query (BO key, key of root node, key of assigned node, ...)

Parameter	Description
IT_FILTER_KEY	Table with keys for a limited set of node instances that are to be affected by the query execution. When this optional parameter is used, the service consumer is able to specify a type of filtering. If this parameter is empty, all node instances (that are assigned to the query) are affected by the query. - <i>Optional parameter</i>
IT_SELECTION_PARAMETERS	Represents a range table of attributes of the query filter structure Using this parameter, you can specify (limit) a value range for dynamic query selections. - <i>Optional parameter</i>
IO_QUERY_AUTHORITYS	Represents the authority object that provides information for authority checks of the query - <i>Optional parameter</i>
IS_QUERY_OPTIONS	Structure that provides the query options. It includes the following components: • MAXIMUM_ROWS – Specifies the maximum number of node instances to be returned by the query. In the case of activated paging: the number of node instances of a single page. • SORTING_OPTIONS – You can use this option to specify the sorting of results. It consists of a table that allows you to specify the sort order (ascending/descending) for the attributes. • PAGING_OPTIONS – You can use this option to change the paging. For example, it allows you to define the top instance of the current page, either by a row (START_ROW) or by an instance key (START_KEY) - <i>Optional parameter</i>
IO_QUERY	Represents the query object that allows execution of the query
IO_READ	Provides read access to the data of business object instances More: Reading Data Interface [page 93]
IO_MODIFY	Provides write access to the data of business object instances More: Reading Data Interface [page 93] - <i>Optional parameter</i>

Parameter	Description
IV_FILL_DATA	Indicates that the node data is to be read. If true, ET_DATA holds the read node data - including all attributes. If false, only the attribute key is returned in table ET_DATA . - Default value = ABAP_FALSE
IT_REQUESTED_ATTRIBUTES	List of names (attribute names) - Optional parameter
EO_MESSAGE	Message object that contains all messages that are generated during query execution More: Reading Data Interface [page 93]
ET_KEY	Set of keys of the node instances for which the query has been executed (result of the query) Note: This parameter is not provided if IV_FILL_DATA is set to false.
ES_QUERY_INFO	Structure that provides query-related information. It comprises the following components: COUNT – Total number of results QUERY_HANDLE – Handle to identify the context
ET_DATA	Table of node data that is returned as a result of the query execution

Exceptions

Exception	Description
/BOBF/CX_FRW	Exception class that throws an exception when an error occurs It provides the user with BOPF-related message texts that are returned from T100.

Example

```
METHOD /bobf/if_frw_query~query.
  DATA: lt_sel_parameters TYPE /bobf/t_frw_query_selparam,
        ls_sel_parameters TYPE /bobf/s_frw_query_selparam.
  CLEAR: eo_message, et_key, es_query_info, et_data.
  * Prepare selection condition
  ls_sel_parameters-sign = 'I'.
  ls_sel_parameters-option = 'EQ'.
  * Payment status: '01' (not paid)
  ls_sel_parameters-attribute_name = <ci>=>sc_node_attribute-root-payment_status.
  ls_sel_parameters-low = '01'.
  APPEND ls_sel_parameters TO lt_sel_parameters.
  io_query->query(
    EXPORTING
      is_ctx                = is_ctx
      it_filter_key         = it_filter_key
      it_selection_parameters = lt_sel_parameters
      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          = 'X'
      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
  ).
ENDMETHOD.
```

Description:

In this sample code, the `query()` method implements the query `GET_OVERDUE_INVOICES` that is used to return all invoices of the `CUSTOMER_INVOICE` business object that are not yet paid.

The data object `ls_sel_parameters` is used to specify the parameters (value range) for query selection. The constant interface `<ci>` of the BO is used to retrieve the payment status of invoices. In this case, the payment status is defined as an attribute of the BO root node. All selection parameters are then appended to the internal table `lt_sel_parameters`, which, in turn, passes a value range for dynamic query selections to `query()` method. The query object `io_query` carries out the actual query.

1.9.6.2 Method RETRIEVE_DEFAULT_PARAM

Returns the default value set of a node instance

You can implement this optional method to provide the service consumer with default parameters for query selection.

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_QUERY</i>
<i>changing</i>	<i>CT_SELECTION_PARAMETERS</i>	<i>type</i>	<i>/BOBF/ T_FRW_QUERY_SELPARAM</i>
<i>raising</i>	<i>/BOBF/CX_FRW</i>		

Parameters

Parameter	Description
<i>IS_CTX</i>	Provides context (model) information for the query
<i>CT_SELECTION_PARAMETERS</i>	Parameters of a query selection that are predefined with default values

Exceptions

Exception	Description
<i>/BOBF/CX_FRW</i>	Exception class that throws an exception when an error occurs It provides the user with BOPF-related message texts that are returned from T100.

1.9.7 Understanding the Validation API

A validation in BOPF is defined as a global ABAP class that implements the validation interface */BOBF/IF_FRW_VALIDATION*.

/BOBF/IF_FRW_VALIDATION

This interface defines methods for validation execution in BOPF. Each ABAP class that represents a validation must implement this interface.

Method Summary

Deprecated

Do not implement the methods

- *CHECK_DELTA*
- *CHECK*

Method	Description
CHECK_DELTA	<i>Deprecated!</i>
CHECK	<i>Deprecated!</i>
EXECUTE	Carries out the validation at runtime

Method Details

[Method CHECK_DELTA \[page 87\]](#)

[Method CHECK \[page 87\]](#)

[Method EXECUTE \[page 88\]](#)

1.9.7.1 Method CHECK_DELTA

Deprecated

Do not use this method!

1.9.7.2 Method CHECK

Deprecated

Do not use this method!

1.9.7.3 Method EXECUTE

Carries out the validation at runtime

This mandatory method implements the main validation logic. It does not provide any modifying access to the node's instance data but returns the messages and the keys of failed node instances (instances for which the action may not be executed).

Signature

<i>importing</i>	<i>IS_CTX</i>	<i>type</i>	<i>/BOBF/S_FRW_CTX_VAL</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IO_READ</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_READ</i>
<i>exporting</i>	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOBF/T_FRW_MESSAGE</i>
	<i>ET_FAILED_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
<i>raising</i>	<i>/BOBF/CX_FRW</i>		

Parameters

Parameter	Description
IS_CTX	Provides context information of the validation More: Action Context IS_CTX [page 90]
IT_KEY	Set of node instance keys that should be processed by the validation
IO_READ	Provides read access to the data of the business object instances More: Reading Data Interface [page 93]
EO_MESSAGE	Message object used for returning information, or warning or error messages related to the validation

Parameter	Description
ET_FAILED_KEY	Set of node instance keys (subset of <i>IT_KEY</i>) that do not match the validation criteria

i Note

Whenever you implement an action validation, the corresponding action will not be executed on node instances that are contained in the parameter *ET_FAILED_KEY*.

Exceptions

Exception	Description
/ BOBF/CX_FRW	Exception class that throws an exception when an error occurs It provides the user with BOPF-related message texts that are returned from T100.

2 BOPF - API Appendix

Content

[Reading Data Interface \[page 93\]](#)

[Action Context IS_CTX \[page 90\]](#)

[Changing Data Interface \[page 102\]](#)

[Message Object Interface \[page 120\]](#)

[Semantics of Parameter IV_BEFORE_IMAGE \[page 129\]](#)

2.1 Action Context IS_CTX

/BOBF/S_FRW_CTX_ACT

This DDIC structure defines the action context in BOPF.

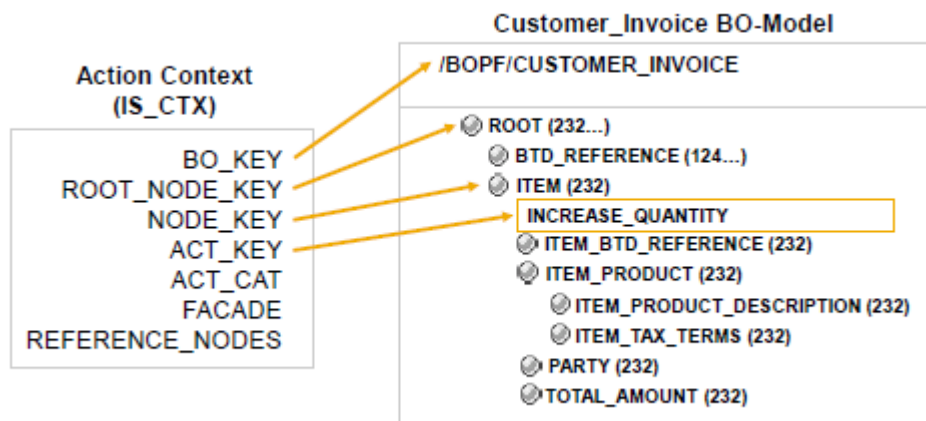
It provides **static** model information of the BO (no instance-related information) that is used as input parameter of all action classes.

Structure Components Summary

Structure Components	Description
<i>BO_KEY</i>	Key of the business object that the action is part of
<i>ROOT_NODE_KEY</i>	Key of the business object's root node
<i>NODE_KEY</i>	Key of the node, to which the action is assigned
<i>ACT_KEY</i>	Key of the action

Structure Components	Description
ACT_CAT	Category of the action Examples: /BOBF/IF_CONF_C=>SC_ACTION_LOCK, /BOBF/IF_CONF_C=>SC_ACTION_DELETE_STANDARD i Note The category is only relevant for framework actions. User-created actions have always the category /BOBF/IF_CONF_C=>SC_ACTION_STANDARD.

Example



Example of an action context

2.2 Authorization Context IS_CTX

/BOBF/S_FRW_CTX_AUTH

This DDIC structure defines the authorization context in BOPF.

The authorization context provides further information about the situation in which the authorization check is executed. It is used as an input parameter for all methods of the authorization check class and allows the applications to control the behavior in a fine granular way - depending on the situation.

Structure Components Summary

Structure Components	Description
NODE_KEY	Key of the BO node for which the authorization check will be executed The NODE_KEY is not necessarily the same node to which the authorization check is assigned. It can also be a subnode that has no own authorization check defined.
ACTIVITY	Activity type (create, change, display, ...) for which the authorization check will be executed
ACTIVITY_SUPPLEMENT	Extended context information about the situation (activity) in which the authorization check is called
BO_SERVICE	Name of the BO service (query or action) for which the authorization check will be executed

2.3 Determination Context IS_CTX

/BOBF/S_FRW_CTX_DET

This DDIC structure defines the determination context in BOPF.

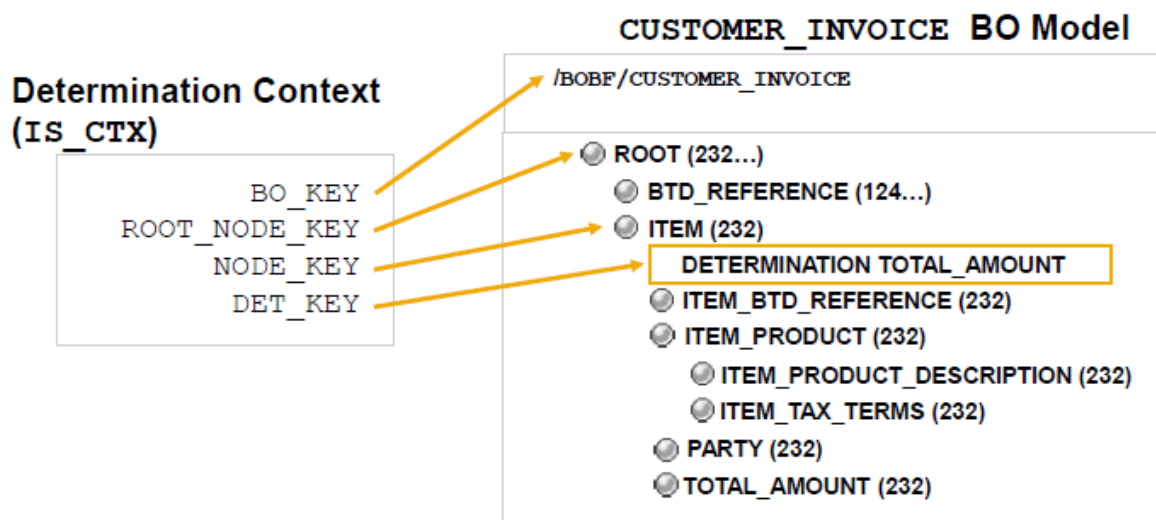
The determination context provides **static** model information of the BO (no instance-related information) and is used as the input parameter for all determination methods.

Structure Components Summary

Main Structure Components	Description
BO_KEY	Model Key [page 131] of the business object that the determination is part of
ROOT_NODE_KEY	Model key of the business object's root node
NODE_KEY	Model key of the node that is parent of the determination
DET_KEY	Model key of this determination
EXEETIME	One of the constants /BOBF/IF_CONF_C=>SC_TIME_* that can be used to identify the actual determination time
ASSOC_KEY	Model key of the association is set when EXEETIME is /BOBF/IF_CONF_C=>SC_TIME_BEFORE_RETRIEVE and when the loading is triggered by a RETRIEVE_BY_ASSOCIATION method.

Main Structure Components	Description
<i>ASSOC_PARAMETERS</i>	List of the association parameters is set when <i>EXEETIME</i> is <i>/BOBF/</i> <i>IF_CONF_C=>SC_TIME_BEFORE_RETRIEVE</i> and when the loading is triggered by a <i>RETRIEVE_BY_ASSOCIATION</i> method.
<i>FILTERED_ATTRIBUTES</i>	List of attributes is set when <i>EXEETIME</i> is <i>/BOBF/</i> <i>IF_CONF_C=>SC_TIME_BEFORE_RETRIEVE</i> and when the loading is triggered by a <i>RETRIEVE_BY_ASSOCIATION</i> method.
<i>CLEANUP_MODE</i>	One of the constants <i>/BOBF/IF_CONF_C=>SC_CLEANUP_*</i> that can be used when <i>EXEETIME</i> is <i>/BOBF/IF_CONF_C=>SC_TIME_CLEANUP</i>
<i>KEEP_CACHE</i>	Is only set when <i>EXEETIME</i> is <i>/BOBF/IF_CONF_C=>SC_TIME_CLEANUP</i> . Evaluating this component enables the application to discard any cached data when it potentially becomes out-dated.

Example



Example of a determination context

2.4 Reading Data Interface

/BOBF/IF_FRW_READ

Defines various methods that allow the business objects to retrieve the node data.

An instance of this interface is passed to the methods of business content classes that can read node data (associations, actions, determinations, queries, validations) when these methods are executed. This interface is implemented by BOPF and directs the calls to the buffering classes.

Method Summary

Method	Description
<i>CONVERT_ALTERN_KEY</i>	Converts alternative keys to BOPF keys (RAW 16 format)
<i>GET_ROOT_KEY</i>	Retrieves the root keys of the specified node keys
<i>RETRIEVE</i>	Retrieves node instance data
<i>RETRIEVE_BY_ASSOCIATION</i>	Retrieves node keys and data by following an association
<i>COMPARE</i>	Compares the current with the previous version of data (data image prior the last change More: Semantics of Parameter IV_BEFORE_IMAGE [page 129]

Method Details

[Method CONVERT_ALTERN_KEY \[page 94\]](#)

[Method GET_ROOT_KEY \[page 96\]](#)

[Method RETRIEVE \[page 97\]](#)

[Method RETRIEVE_BY_ASSOCIATION \[page 98\]](#)

[Method COMPARE \[page 101\]](#)

2.4.1 Method CONVERT_ALTERN_KEY

This method converts alternative keys to BOBF keys (RAW 16 format) .

Signature

<i>importing</i>	<i>IV_NODE_KEY</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_ALTKEY_KEY</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_TARGET_ALTKEY_KEY</i>	<i>type</i>	<i>/BOBF/OBM_ALTKEY_KEY</i>
	<i>IT_KEY</i>	<i>type</i>	<i>INDEX TABLE</i>
	<i>IV_BEFORE_IMAGE</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>IV_INVALIDATE_CACHE</i>	<i>type</i>	<i>BOOLE_D</i>
<i>exporting</i>	<i>ET_KEY</i>	<i>type</i>	<i>INDEX TABLE</i>

ET_RESULT

type

/BOBF/T_FRW_KEYINDEX

Parameters

Parameter	Description
IV_NODE_KEY	Model Key [page 131] of the node where the conversion is to be carried out. The source and target alternative keys belong to this node.
IV_ALTKEY_KEY	Model key of the source alternative key
IV_TARGET_ALTKEY_KEY	Model key of the target alternative key - <i>Optional parameter</i> - <i>Default value</i> = technical key of IV_NODE_KEY
IT_KEY	Table of source alternative keys for which the target alternative keys is be retrieved
IV_BEFORE_IMAGE	Indicates whether the conversion should run on the before image or not. The before image can be different - depending on the context where this method is used. More: Semantics of Parameter IV_BEFORE_IMAGE [page 129] - <i>Optional parameter</i> - <i>Default value</i> = false
IV_INVALIDATE_CACHE	<i>Deprecated</i>
ET_KEY	Resulting target alternative keys of the conversion. Typed by the table type of the target alternative key. i Note Do not use this parameter if you have a non-unique alternative key of the node instance key.
ET_RESULT	Table of technical keys pointing to a line in IT_KEY. ET_RESULT can have multiple entries pointing to the same source alternative key in IT_KEY if the source alternative key is not unique.

i Note

You can also use the [Technical Key \[page 132\]](#) as the source or the target of a conversion. To specify the KEY attribute in IV_ALTKEY_KEY or IV_TARGET_ALTKEY_KEY , access the constant /BOBF/IF_FRW_C=>SC_ALTERNATIVE_KEY_KEY .

2.4.2 Method GET_ROOT_KEY

The method retrieves the root keys of the specified node keys. The result of the method call is stored in the table of node instance pairs where the node keys are assigned to its root keys.

Signature

<i>importing</i>	<i>IV_NODE</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IV_BEFORE_IMAGE</i>	<i>type</i>	<i>BOOLE_D</i>
<i>exporting</i>	<i>ET_TARGET_KEY</i>	<i>type</i>	<i>BOPF/T_FRW_KEY</i>
	<i>ET_KEY_LINK</i>	<i>type</i>	<i>BOPF/T_FRW_KEY_LINK</i>
	<i>ET_FAILED_KEY</i>	<i>type</i>	<i>/BOBF / T_FRW_KEY</i>

Parameters

Parameter	Description
<i>IV_NODE</i>	Model Key [page 131] of the node where the technical keys of the ROOT node are to be retrieved :
<i>IT_KEY</i>	Table of Node Instance [page 131] for which the technical keys of their ROOT node are to be read
<i>IV_BEFORE_IMAGE</i>	Parameter that indicates whether the retrieval runs on the before image. The semantics of the before image can be different, depending on the context where this method is used. More: Semantics of Parameter IV_BEFORE_IMAGE [page 129] - <i>Optional parameter</i> - <i>Default value = false</i> i Note A before image is not always available. Depending on the content category (action, association, determination, query, validation) that is called, and on the execution time (if the content is a determination), there may be no valid before image. In such a case, the returned values are those of the current image.
<i>ET_TARGET_KEY</i>	Table of technical ROOT node keys that are retrieved for the node instances in <i>IT_KEY</i>

Parameter	Description
<i>ET_KEY_LINK</i>	A table of node instance pairs (Source node instance, ROOT node instance)
<i>ET_FAILED_KEY</i>	Table of node instance keys that could not be read (for example, if the node instances do not exist). It is equal to or a subset of <i>IT_KEY</i> .

2.4.3 Method RETRIEVE

Retrieves node instance data

The method exports the data of the requested node instances.

If a node instance does not exist, or it does not exist in the requested image (before or current image), the key is returned in the parameter *ET_FAILED_KEY*.

Signature

<i>importing</i>	<i>IV_NODE</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IV_BEFORE_IMAGE</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>IV_FILL_DATA</i>	<i>type</i>	<i>BOOLE_D</i>
<i>exporting</i>	<i>IT_REQUESTED_ATTRIBUTES</i>	<i>type</i>	<i>/BOBF/T_FRW_NAME</i>
	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_MESSAGE</i>
	<i>ET_DATA</i>	<i>type</i>	<i>INDEX TABLE</i>
	<i>ET_FAILED_KEY</i>	<i>type</i>	<i>/BOBF / T_FRW_KEY</i>
	<i>ET_NODE_CAT</i>	<i>type</i>	<i>/BOBF/T_FRW_NODE_CAT</i>

Parameters

Deprecated

Do not use the parameters

1. *IT_REQUESTED_ATTRIBUTES*
2. *ET_NODE_CAT*

Parameter	Description
<i>IV_NODE</i>	Model Key [page 131] of the node that should be read
<i>IT_KEY</i>	Table of Node Instance [page 131] that should be read, represented by its technical keys

Parameter	Description
<i>IV_BEFORE_IMAGE</i>	Indicates that the before image is to be read. The before image can be different, depending on the context where this method is used. More: Semantics of Parameter IV_BEFORE_IMAGE [page 129] - <i>Optional</i> parameter - <i>Default value</i> = false
<i>IV_FILL_DATA</i>	Indicates that the node data is to be read. If true, <i>ET_DATA</i> holds the read node data - including all attributes. If false, only the attribute key is returned in table <i>ET_DATA</i>. - <i>Optional</i> parameter - <i>Default value</i> = true i Note This parameter might be set to false if just an existence check has to be performed.
<i>IT_REQUESTED_ATTRIBUTES</i>	<i>Deprecated!</i>
<i>EO_MESSAGE</i>	Message container that contains all messages that are created by the <i>RETRIEVE</i> call
<i>ET_DATA</i>	Table of node data typed with the <i>Combined Table</i> type of the read node i Note This parameter is not provided if <i>IV_FILL_DATA</i> is set to false.
<i>ET_FAILED_KEY</i>	Table of node instance keys that could not be read (for example, if the node instances do not exist). It is equal to or a subset of <i>IT_KEY</i> .
<i>ET_NODE_CAT</i>	<i>Deprecated!</i>

2.4.4 Method RETRIEVE_BY_ASSOCIATION

Retrieves node keys and data by following an association The method returns the data of the target nodes - if requested. It also returns a key link table that assigns the source nodes to the target nodes and a table of the target keys.

Signature

importing	IV_NODE	type	/BOBF/OBM_NODE_KEY
	IT_KEY	type	/BOBF/T_FRW_KEY
	IV_ASSOCIATION	type	/BOBF/OBM_ASSOC_KEY
	IS_PARAMETERS	type ref to	DATA
	IT_FILTERED_ATTRIBUTES	type	BOBF/T_FRW_NAME
	IV_FILL_DATA	type	BOOLE_D
	IV_BEFORE_IMAGE	type	BOOLE_D
	IT_REQUESTED_ATTRIBUTES	type	/BOBF/T_FRW_NAME
exporting	EO_MESSAGE	type ref to	/BOBF/IF_FRW_MESSAGE
	ET_DATA	type	INDEX TABLE
	ET_KEY_LINK	type	/BOBF/T_FRW_KEY_LINK
	ET_TARGET_KEY	type	/BOBF/T_FRW_KEY
	ET_FAILED_KEY	type	/BOBF /T_FRW_KEY

Parameters

Deprecated

Do not use the parameter *IT_REQUESTED_ATTRIBUTES*!

Parameter	Description
IV_NODE	Model Key [page 131] of the source node of the association
IT_KEY	Table of technical keys of source Node Instance [page 131] for which the target node instances are to be retrieved More:
IV_ASSOCIATION	Model key of the association on the source node that is to be used
IS_PARAMETERS	Indicates whether the filter has values for a filtered association or not - <i>Optional</i> parameter

Parameter	Description
<i>IT_FILTERED_ATTRIBUTES</i>	Specifies which attributes in the filter structure are to be used for the evaluation of the association - <i>Optional</i> parameter
<i>IV_FILL_DATA</i>	Indicates that the data of the target node instances is to be read. If set to true, the parameter <i>ET_DATA</i> will have the data of the target node instances. - <i>Optional</i> parameter - <i>Default value</i> = false
<i>IV_BEFORE_IMAGE</i>	Indicates that the before image is to be read. The before image can be different, depending on the context where this method is used. More :Semantics of Parameter IV_BEFORE_IMAGE [page 129] - <i>Optional</i> parameter - <i>Default value</i> = false
<i>IT_REQUESTED_ATTRIBUTES</i>	<i>Deprecated!</i>
<i>EO_MESSAGE</i>	Message container that comprises the messages that are raised by the <i>RETRIEVE_BY_ASSOCIATION</i> call
<i>ET_DATA</i>	Table of target node data, typed with the <i>Combined Table</i> type of the read target node. Note that this is only provided if <i>IV_FILL_DATA</i> is set to true.
<i>ET_KEY_LINK</i>	A table of node instance pairs (source node instance, target node instance) that is returned by this association
<i>ET_TARGET_KEY</i>	Table of the target node instances that is returned by this association
<i>ET_FAILED_KEY</i>	Table of node instance keys that could not be read (for example, if the source node instances do not exist). It is equal to or a subset of <i>IT_KEY</i> .

2.4.5 Method COMPARE

Compares the current with the previous version of data (data image prior the last change)

You can use this method to compare the current and before image of node instances *IT_KEY* of the node *IV_NODE* .

The result of the method call is passed as the change object *EO_CHANGE* . If required (*IV_FILL_ATTRIBUTES* = true), the attributes of the node instances are compared too, and the result is also written into the change object. The scope defines whether only the node specified or the complete subtree is compared.

Signature

importing	<i>IV_NODE_KEY</i>	type	/BOBF/OBM_NODE_KEY
	<i>IT_KEY</i>	type	/BOBF/T_FRW_KEY
	<i>IV_FILL_ATTRIBUTES</i>	type	BOOLE_D
	<i>IV_SCOPE</i>	type	BOBF/FRW_SCOPE
exporting	<i>EO_CHANGE</i>	type ref to	/BOBF/IF_FRW_CHANGE

Parameters

Parameter	Description
<i>IV_NODE_KEY</i>	Model Key [page 131] for the node where the data versions is to be compared
<i>IT_KEY</i>	Technical Key [page 132] of node instances that should be compared
<i>IV_FILL_ATTRIBUTES</i>	If true, the comparison is carried out on attribute level. Changed attributes are part of the change container content. - <i>Optional</i> parameter - <i>Default value</i> = false

Parameter	Description
<code>IV_SCOPE</code>	The scope defines whether only the specified node or the complete subtree is compared. - <i>Optional</i> parameter - <i>Default value</i> = local i Note The call <code>/BOBF/IF_FRW_C=>SC_SCOPE_LOCAL</code> compares exactly the node indicated in <code>IV_NODE_KEY</code>, whereas <code>/BOBF/IF_FRW_C=>SC_SCOPE_SUBSTRUCTURE</code> also compares the sub nodes of <code>IV_NODE_KEY</code>. Note that this scope is supported only for the ROOT node of a business object.
<code>EO_CHANGE</code>	Change container that comprises the changes between the transactional image and the Before Image [page 131].

2.5 Changing Data Interface

`/BOBF/IF_FRW_MODIFY` defines various callback methods that allow the business objects to change the node data.

An instance of this interface is passed to the methods the business content classes that can change node data (CREATE method of associations, EXECUTE method of actions and determinations) when these methods are executed. This interface is implemented by BOBF and directs the calls to the buffering classes.

Method Summary

Method	Description
<code>CREATE</code>	Creates a new node instance
<code>DELETE</code>	Deletes one or multiple node instances

Method	Description
<i>DO_ACTION</i>	Executes a BOBF action
<i>DO_MODIFY</i>	Modifies a set of node instances
<i>END_MODIFY</i>	Triggers a block of modifications on node instances explicitly
<i>NOTIFY_CHANGE</i>	Indicates node instances as changed without executing an modification
<i>NOTIFY_ASSOCIA- TION_CHANGE</i>	Indicates outgoing associations of a node as changed without executing a modification
<i>NOTIFY_PROP- ERTY_CHANGE</i>	Indicates node instance properties as changed
<i>UPDATE</i>	Updates the data of a node instance

Method Details

[Method CREATE \[page 103\]](#)

[Method DELETE \[page 106\]](#)

[Method DO_ACTION \[page 107\]](#)

[Method DO_MODIFY \[page 108\]](#)

[Method END_MODIFY \[page 108\]](#)

[Method NOTIFY_CHANGE \[page 110\]](#)

[Method NOTIFY_ASSOCIATION_CHANGE \[page 111\]](#)

[Method NOTIFY_PROPERTY_CHANGE \[page 112\]](#)

[Method UPDATE \[page 113\]](#)

2.5.1 Method CREATE

Creates a new node instance

The signature allows the caller to assign a key to the node instance to be created. If the caller does not state a key in parameter *IV_KEY* , the framework assigns a new key automatically. This key is then returned to the

caller in the parameter *EV_KEY* . The result of the method call is a created instance of a specified node in the buffer.

<i>importing</i>	<i>IV_NODE</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IV_NODE_CAT</i>	<i>type</i>	<i>/BOBF/ OBM_NODE_CAT_KEY</i>
	<i>IS_DATA</i>	<i>type ref to</i>	<i>DATA</i>
	<i>IT_CHANGED_FIELDS</i>	<i>type</i>	<i>/BOBF/T_FRW_NAME</i>
	<i>IV_ASSOC_KEY</i>	<i>type</i>	<i>/BOBF/OBM_ASSOC_KEY</i>
	<i>IV_SOURCE_NODE_KEY</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_SOURCE_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IV_ROOT_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
<i>exporting</i>	<i>EV_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>

Parameters

Deprecated

Do not use the parameter *IV_NODE_CAT*!

Parameter	Description
<i>IV_NODE</i>	Model Key [page 131] of that node for which a node instance is to be created
<i>IV_KEY</i>	The Technical Key [page 132] that is to be used for the created node instance If <i>IV_KEY</i> is not provided, the framework generates a new key automatically. The newly generated key is returned by the exporting parameter <i>EV_KEY</i> . - <i>Optional</i> parameter
<i>IV_NODE_CAT</i>	<i>Deprecated!</i>
<i>IS_DATA</i>	Data of the node that is to be used for node instance creation - <i>Optional</i> parameter

Parameter	Description
<i>IT_CHANGED_FIELDS</i>	Table with attribute names that is to be taken into account in the structure <i>IS_DATA</i> when creating the node instance - <i>Optional</i> parameter
<i>IV_ASSOC_KEY</i>	Model key of the association that leads to the node for which the new node instance is to be created - <i>Optional</i> parameter
<i>IV_SOURCE_NODE_KEY</i>	Model key of the association's source node - <i>Optional</i> parameter
<i>IV_SOURCE_KEY</i>	Technical key of the source node instance - <i>Optional</i> parameter
<i>IV_ROOT_KEY</i>	Specifies the technical key of root node instance to which the newly created node instance belongs - <i>Optional</i> parameter
<i>EV_KEY</i>	Technical key of the newly created node instance

i Note

This parameter can be used for to improve the performance behavior since the framework does not need to discover the root node instance by its own.

However, note that wrong content can cause undefined behavior of the business object.

i Note

The importing parameters *IV_ASSOC_KEY* , *IV_SOURCE_NODE_KEY* , and *IV_SOURCE_KEY* are necessary if the node to be created is a sub node and not a root within the business object's node hierarchy.

2.5.2 Method DELETE

Deletes one or multiple node instances

Signature

<i>importing</i>	<i>IV_NODE</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IV_ROOT_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>

Parameters

Parameter	Description
<i>IV_NODE</i>	Model Key [page 131] of the node to be deleted
<i>IV_KEY</i>	Technical Key [page 132] of the node instance to be deleted - <i>Optional</i> parameter
<i>IT_KEY</i>	Table of technical keys of the Node Instance [page 131] to be deleted - <i>Optional</i> parameter
<i>IV_ROOT_KEY</i>	Specifies the technical key of the root node instance to which the deleted node instance belongs - <i>Optional</i> parameter

i Note

This parameter can be used to improve the performance behavior since the framework does not need to discover the root node instance on its own.

Note, however, that incorrect content can cause undefined behavior of the business object.

2.5.3 Method DO_ACTION

Executes a BOPF action

The result of the method execution is identical to that of an action called externally (using the Service Manager API). It can include all kinds of modifications.

i Note

When you call an action using this method, the action validations will be processed in the same way as if the action had been called by means of the Service Manager API.

Signature

importing	IV_ACT_KEY	type	/BOBF/ACT_KEY
	IT_KEY	type	/BOBF/T_FRW_KEY
	IS_PARAMETERS	type ref to	DATA
exporting	EO_MESSAGE	type ref to	/BOBF/IF_FRW_MESSAGE
	ET_FAILED_KEY	type	/BOBF/T_FRW_KEY
	EO_CHANGE	type ref to	/BOPF/IF_FRW_CHANGE
	ET_DATA	type	INDEX TABLE

Parameters

Parameter	Description
IV_ACT_KEY	Model Key [page 131] of the action to be executed
IT_KEY	Table of technical Node Instance [page 131] keys for which the action is to be executed
IS_PARAMETERS	Parameter structure used as input for the action - if the action has a parameter structure defined - <i>Optional</i> parameter
EO_MESSAGE	Message container that comprises all messages created by the action

Parameter	Description
<i>ET_FAILED_KEY</i>	Table of technical node instance keys, for which the action was not executed successfully
<i>EO_CHANGE</i>	Change container that comprises all changes executed by the action
<i>ET_DATA</i>	The exporting type parameter specifies whether and if so what kind of data the action should be able to return to the caller during the action execution.

2.5.4 Method DO_MODIFY

Modifies a set of node instances

Signature

importing *IT_MODIFICATION* *type* */BOBF/T_FRW_MODIFICATION*

Parameters

Parameter	Description
<i>IT_MODIFICATION</i>	Table with changes on node instances

2.5.5 Method END_MODIFY

Explicitly triggers a block of modifications on node instances

Generally, BOPF buffers all modifications made by a content method. When the *END_MODIFY* method is called, the modifications that were made during the processing of the calling methods are immediately executed on the buffer. Based on the modifications, the change handler and a message instance are returned.

i Note

In most cases, it is not necessary to call this method. After processing the content method and before retrieving data, the framework automatically flushes all modifications. The only reason to trigger the modifications explicitly is if processing of the content method relies on values of the parameters *EO_MESSAGE* and *EO_CHANGE*.

Signature

<i>importing</i>	<i>IV_PROCESS_IMMEDIATELY</i>	<i>type</i>	<i>BOOL_D</i>
<i>exporting</i>	<i>EO_MESSAGE</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_MESSSAGE</i>
	<i>EO_CHANGE</i>	<i>type ref to</i>	<i>/BOBF/IF_FRW_CHANGE</i>

Parameters

Parameter	Description
<i>IV_PROCESS_IMMEDIATELY</i>	If this indicator is set to true, the changes that have been made so far will not only be immediately executed on the buffer, but also all corresponding determinations after modification will be executed. - <i>Optional</i> parameter
<i>EO_MESSAGE</i>	Message container with all messages that are created during the modification
<i>EO_CHANGE</i>	Change container that contains all changes that are executed by the modification

i Note

This functionality is only available within action implementations.

i Note

All changes that are performed by the Internal Access Object *IO_MODIFY* are not applied directly but stored in a modification buffer. Due to performance, the changes are executed automatically in one shot when the data is accessed from the business object (lazy modifications). By calling the *END_MODIFY* method, you are able to force the flush of the modification buffer immediately.

2.5.6 Method NOTIFY_CHANGE

Indicates node instances as changed without executing an modification (without real data change)

As a result of the method call, a change notification is created and used to explicitly trigger determinations without any real data change.

i Note

Do not use this method to notify changes that were already made explicitly using CREATE , UPDATE , or DELETE . In latter case, the changes are notified by the framework.

Signature

<i>importing</i>	<i>IV_NODE_KEY</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IV_CHANGE_MODE</i>	<i>type</i>	<i>/BOBF/ CONF_CHANGE_MODE</i>

Parameters

Parameter	Description
<i>IV_NODE_KEY</i>	Model Key [page 131] of the node for which the change is to be notified
<i>IV_KEY</i>	Technical Key [page 132] of the node instance for which the change is to be notified - <i>Optional</i> parameter
<i>IT_KEY</i>	Table with technical keys of Node Instance [page 131] for which the change is to be notified More: - <i>Optional</i> parameter

Parameter	Description
<i>IV_CHANGE_MODE</i>	The kind of change that is to be notified - Default value = <code>/BOBF/IF_FRW_C =>SC_MODIFY_UPDATE</code>

2.5.7 Method NOTIFY_ASSOCIATION_CHANGE

Indicates outgoing associations of a node as changed, without executing a modification

If consumers hold data in their own buffers, this notification is used as a trigger that the buffer might be outdated. To get recent data, the consumer would re-navigate the corresponding association for all source node instances that are currently stored in the buffer.

Signature

<i>importing</i>	<i>IV_NODE_KEY</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IV_ASSOC_KEY</i>	<i>type</i>	<i>/BOBF/OBM_ASSOC_KEY</i>

Parameters

Parameter	Description
<i>IV_NODE_KEY</i>	Model Key [page 131] of the association's source node for which the change is to be notified
<i>IV_KEY</i>	Technical Key [page 132] of the node instance for which the change is to be notified - <i>Optional</i> parameter

Parameter	Description
<i>IT_KEY</i>	Table with technical keys of Node Instance [page 131] for which the change is to be notified - <i>Optional</i> parameter
<i>IV_ASSOC_KEY</i>	Model key of the association for which the change is to be notified If this parameter is not provided, the change will be notified for all associations that have been indicated as the source node (in the parameter <i>IV_NODE_KEY</i>). - <i>Optional</i> parameter

2.5.8 Method NOTIFY_PROPERTY_CHANGE

Indicates node instance properties as changed

As result of method call, a property change notification is created and passed to the service consumer.

i Note

In general, it is sufficient to model the dependency between the change notifications and the property change notifications using the BOPF configuration tools. However, in some cases, the modelling capabilities might not be detailed enough to cover all use cases. This method enables the application developers to explicitly create property change notifications.

Signature

<i>importing</i>	<i>IV_NODE_KEY</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IT_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>IV_NODE_CHANGED</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>IV_NODE_ATTRIB- UTE_CHANGED</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>IV_ASSOCIATION_CHANGED</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>IV_ACTION_CHANGED</i>	<i>type</i>	<i>BOOLE_D</i>

Parameters

Parameter	Description
<code>IV_NODE_KEY</code>	Model Key [page 131] of the node for which the property change is to be notified
<code>IV_KEY</code>	Technical Key [page 132] of the node instance for which the property change is to be notified - <i>Optional parameter</i>
<code>IT_KEY</code>	Table with technical keys of Node Instance [page 131] for which the property change is to be notified - <i>Optional parameter</i>
<code>IV_NODE_CHANGED</code>	If the value is true, a property change of the node will be notified - Default value = <code>ABAP_FALSE</code>
<code>IV_NODE_ATTRIBUT- UTE_CHANGED</code>	If the value is true, a property change of the node attributes will be notified - Default value = <code>ABAP_FALSE</code>
<code>IV_ASSOCIATION_CHANGED</code>	If the value true, a property change will be notified for all associations with the node as the source node - <i>Default value</i> = <code>ABAP_FALSE</code>
<code>IV_ACTION_CHANGED</code>	If the value is true, a property change will be notified for all actions of the node - <i>Default value</i> = <code>ABAP_FALSE</code>

2.5.9 Method UPDATE

Updates the data of a node instance

As result of the method call, the node instance is updated in the buffer. If the `IT_CHANGED_FIELDS` parameter is not passed, all fields of the node instance will be updated. Due to performance reasons, you can also use the `ROOT_KEY` parameter.

Signature

<i>importing</i>	<i>IV_NODE</i>	<i>type</i>	<i>/BOBF/OBM_NODE_KEY</i>
	<i>IV_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IV_ROOT_KEY</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IS_DATA</i>	<i>type ref to</i>	<i>DATA</i>
	<i>IT_CHANGED_FIELDS</i>	<i>type</i>	<i>/BOBF/T_FRW_NAME</i>

Parameters

Parameter	Description
<i>IV_NODE</i>	Model Key [page 131] of the node that is to be updated
<i>IV_KEY</i>	Technical Key [page 132] of the node instance that is to be updated
<i>IV_ROOT_KEY</i>	Technical key of the corresponding root Node Instance [page 131] More: - <i>Optional</i> parameter
<i>IS_DATA</i>	Provides node instance data for update It is represented as a data reference that is typed with node's <i>Combined Structure</i> .
<i>IT_CHANGED_FIELDS</i>	Table of names for attributes that are to be used for the update. If this parameter is not provided, all attributes will be updated. - <i>Optional</i> parameter

2.6 Changing Object Interface

/BOBF/IF_FRW_CHANGE provides access to change notifications that were done on business object instances during a roundtrip

An instance of this interface is returned by all changing methods on a business object. Please note that not all methods of the interface will be used by business object implementations like actions, determinations, and so on. They are used within the framework only and will not be described in detail.

Method Summary

Method	Description
<i>ADD</i>	Adds a list of change notifications. To be used within the framework only.
<i>ADD_CHANGE</i>	Adds a single change notification on node instance data. To be used within the framework only.
<i>ADD_CONTENT_CHANGE</i>	Adds a single change notification to the content of an instance (for example, change of an association). To be used within the framework only.
<i>CONDENSE</i>	Removes redundant change notifications. To be used within the framework only
<i>GET</i>	Returns all change notifications
<i>GET_CHANGES</i>	Returns the specified change notifications
<i>HAS_CHANGES</i>	Returns true if the change object has any change notifications
<i>MERGE</i>	Adds the changes of any given change object instance. To be used within the framework only

Method Details

[Method GET \[page 116\]](#)

[Method GET_CHANGES \[page 117\]](#)

[Method HAS_CHANGES \[page 119\]](#)

2.6.1 Method GET

Returns all change notifications

The result of this method can be influenced by the *IV_SORTED* parameter for performance-optimized processing. For example, if an implementation is interested in the changes for specific node instances, this method might be called so that the result is sorted by the instance keys.

Signature

<i>importing</i>	<i>IV_SORTED</i>	<i>type</i>	<i>/BOBF/ FRW_CHANGE_SORT</i>
	<i>IV_NO_LOAD</i>	<i>type</i>	<i>BOOLE_D</i>
<i>exporting</i>	<i>ET_CHANGE</i>	<i>type</i>	<i>/BOBF/T_FRW_CHANGE</i>
	<i>EV_SORTED</i>	<i>type</i>	<i>/BOBF/ FRW_CHANGE_SORT</i>
	<i>ET_CONTENT_CHANGE</i>	<i>type</i>	<i>/BOBF/ T_FRW_CHANGE_CONTENT</i>
	<i>ET_CHANGE_LOAD</i>	<i>type</i>	<i>/BOBF/T_FRW_CHANGE</i>
	<i>ET_CHANGED_LOAD_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_CHANGE</i>
	<i>EV_RELOAD_INT</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>EV_RELOAD_DEL</i>	<i>type</i>	<i>BOOLE_D</i>

Parameters

Deprecated

Do not use the parameters

- *ET_CHANGE_LOAD*
- *ET_CHANGE_LOAD_KEY*
- *EV_RELOAD_INT*
- *EV_RELOAD_DEL*

Parameter	Description
<i>IV_SORTED</i>	Indicates according to which criteria the results in <i>ET_CHANGE</i> are to be sorted -- for example, by change mode, node instance key. This could be used to influence the result for optimized processing - <i>Optional</i> parameter -Default value = <i>/BOBF/IF_FRW_C=>SC_CHANGE_SORT_COMPLETE</i> (sorting on all fields)
<i>IV_NO_LOAD</i>	If true, no load notifications will be returned by <i>ET_CHANGE</i>, <i>ET_CHANGE_LOAD</i> and <i>ET_CHANGE_LOAD_KEY</i> - Optional parameter
<i>ET_CHANGE</i>	A table of node data change notifications
<i>EV_SORTED</i>	Indicates how the result in <i>ET_CHANGE</i> was sorted
<i>ET_CONTENT_CHANGE</i>	A table of content change notifications.
<i>ET_CHANGE_LOAD</i>	<i>Deprecated!</i>
<i>ET_CHANGE_LOAD_KEY</i>	<i>Deprecated!</i>
<i>EV_RELOAD_INT</i>	<i>Deprecated!</i>
<i>EV_RELOAD_DEL</i>	<i>Deprecated!</i>

2.6.2 Method GET_CHANGES

Returns all node data change notifications

In contrast to the method GET this method returns node data change notifications only.

Signature

<i>importing</i>	<i>IV_CHANGE_MODE</i>	<i>type</i>	<i>/BOBF/ CONF_CHANGE_MODE</i>
	<i>IS_CHANGE_MODE</i>	<i>type</i>	<i>/BOBF/ S_CONFRO_CHG_MODE</i>
	<i>IV_FAILED</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>IV_NODE_KEY</i>	<i>type</i>	<i>/BOBF/EBM_NODE_KEY</i>
<i>exporting</i>	<i>ET_CHANGE</i>	<i>type</i>	<i>/BOBF/T_FRW_CHANGE</i>
	<i>ET_CHANGED_KEY</i>	<i>type</i>	<i>/BOBF/T_FRW_KEY</i>
	<i>ET_CHANGED_NODE</i>	<i>type</i>	<i>/BOBF/T_FRW_NODE</i>

Parameters

Parameter	Description
<i>IV_CHANGE_MODE</i>	To filter the result for a special kind of change notification. Could be used if the consumer is only interested in specific changes (for example, create, update, and so on). If this parameter is not provided, the method will return all kinds of notifications -Optional parameter
<i>IS_CHANGE_MODE</i>	To filter the result for a set of special change notifications. If this parameter is not provided, the method will return all kinds of notifications -Optional parameter
<i>IV_FAILED</i>	If true, the result the method will return also notifications about failed changes - <i>Optional</i> parameter
<i>IV_NODE_KEY</i>	Model Key [page 131] of a node to filter the result for a specific node - <i>Optional</i> parameter
<i>ET_CHANGE</i>	Table of change notifications

Parameter	Description
<i>ET_CHANGED_KEY</i>	Table of Node Instance [page 131] keys that have been changed
<i>ET_CHANGED_NODE</i>	Table of node instance keys that have been changed including the corresponding node model keys

2.6.3 Method HAS_CHANGES

Indicates whether changes exists

Signature

<i>importing</i>	<i>IV_SELECTION</i>	<i>type</i>	<i>/BOBF/ FRW_CHANGE_SCOPE</i>
	<i>IV_CHANGE_MODE</i>	<i>type</i>	<i>/BOBF/ CONF_CHANGE_MODE</i>
<i>returning</i>	<i>EV_EXISTS</i>	<i>type</i>	<i>BOOLE_D</i>

Parameters

Parameter	Description
<i>IV_SELECTION</i>	Defines the scope for the result. The scope can be set on all, successful, or failed changes - <i>Optional</i> parameter - <i>Default</i> = <i>/BOBF/IF_FRW_C=>SC_CHANGE_SUCC_ONLY</i>
<i>IV_CHANGE_MODE</i>	Indicates which kind of change is to be taken into consideration - <i>Optional</i> parameter

Parameter	Description
<i>EV_EXISTS</i>	True if changes exists

2.7 Message Object Interface

`/BOBF/IF_FRW_MESSAGE`

Interface of a message container object that provides access to all (class-based) messages that are issued during a BOPF API call.

All BOPF messages are class-based messages inheriting from `/BOBF/CM_FRW`.

Method Summary

Method	Description
<i>ADD</i>	Merges two message containers
<i>ADD_CM</i>	Adds a class-based message to the message object
<i>ADD_CM_WITH_NEW_LOCATION</i>	Adds a class-based message with new location information to the message container
<i>ADD_EXCEPTION</i>	<i>Deprecated!</i>
<i>ADD_MESSAGE</i>	Adds a SYMSG-typed message to the message object
<i>CHECK</i>	Returns true if the message object contains error messages
<i>GET</i>	Returns all messages
<i>GET_MESSAGES</i>	Returns all specified messages

Method Details

[Method ADD \[page 121\]](#)

[Method ADD_CM \[page 123\]](#)

[Method ADD_CM_WITH_NEW_LOCATION \[page 124\]](#)

[Method ADD_EXCEPTION \[page 125\]](#)

[Method ADD_MESSAGE \[page 126\]](#)

[Method CHECK \[page 127\]](#)

[Method GET \[page 127\]](#)

[Method GET_MESSAGES \[page 128\]](#)

2.7.1 Method ADD

Adds all messages contained in a given message object to this message object.

To be used to merge two message objects.

Signature

itk/
DB
PD
DB
HE
DE
IS
AF
SE
EE
R
W
-
M
E
S
S
A
G
E

Parameters

D
e
B
a
r
a
m
e
t
e
r
-
M
e
s
s
a
g
e
s
A
n
d
t
h
a
t
i
s
t
o
b
e
a
d
d
e
d
t
o

D
e
B
a
r
a
m
e
t
e
n
-
t
h
e
m
e
s
s
a
g
e
o
b
j
e
c
t
-

2.7.2 Method ADD_CM

Adds a class-based message to the message object

A class-based message can only be added to the message object if it inherits from `/BOBF/CM_FRW`

Signature

<i>importing</i>	<code>IO_MESSAGE</code>	<i>type ref to</i>	<code>/BOBF/IF_FRW_MESSAGE</code>
	<code>IT_MESSAGE</code>	<i>type</i>	<code>/BOBF/CM_FRW=>TT_FRW</code>

Parameters

Parameter	Description
<i>IO_MESSAGE</i>	Message instance that is to be added to the message object
<i>IT_MESSAGE</i>	Table of message instances that is to be added to the message object

2.7.3 Method ADD_CM_WITH_NEW_LOCATION

Adds a class-based message with new location information the message object

To be used if an existing message instance is to be reused while providing information about origin location, severity, and so on.

Signature

<i>importing</i>	<i>IO_MESSAGE</i>	<i>type ref to</i>	<i>/BOBF/CM_FRW</i>
	<i>IV_NEW_BOPF_LOCATION</i>	<i>type</i>	<i>/BOBF/CONF_KEY</i>
	<i>IS_NEW_ORIGIN_LOCATION</i>	<i>type</i>	<i>/BOBF/S_FRW_LOCATION</i>
	<i>IT_NEW_ENVIRONMENT_LOCATION</i>	<i>type</i>	<i>/BOBF/T_FRW_LOCATION</i>
	<i>IV_NEW_LIFETIME</i>	<i>type</i>	<i>/BOBF/CM_FRW=>TY_MESSAGE_LIFETIME</i>
	<i>IV_NEW_SYMPTOM</i>	<i>type</i>	<i>/BOBF/CM_FRW=>TY_MESSAGE_SYMPTOM</i>
	<i>IV_NEW_SEVERITY</i>	<i>type</i>	<i>/BOBF/CM_FRW=>TY_MESSAGE_SEVERITY</i>

Parameters

Deprecated

Do not use the parameters

- *IT_NEW_BOPF_LOCATION*
- *IT_NEW_ENVIRONMENT_LOCATION*

Parameter	Description
<i>IO_MESSAGE</i>	Message instance that is to be added to the message object
<i>IV_NEW_BOPF_LOCATION</i>	<i>Deprecated. Do not use!</i>
<i>IS_NEW_ORIGIN_LOCATION</i>	New origin location for the added message instance - <i>Optional</i> parameter
<i>IT_NEW_ENVIRONMENT_LOCATION</i>	<i>Deprecated. Do not use!</i>
<i>IV_NEW_LIFETIME</i>	New lifetime for the added message instance - <i>Optional</i> parameter
<i>IV_NEW_SYMPTOM</i>	New symptom for the added message instance - <i>Optional</i> parameter
<i>IV_NEW_SEVERITY</i>	New severity for the added message instance - <i>Optional</i> parameter

2.7.4 Method ADD_EXCEPTION

Deprecated

Do not use this method!

2.7.5 Method ADD_MESSAGE

Adds a SYMSG-typed message to the message object

Signature

<i>importing</i>	<i>IS_MSG</i>	<i>type</i>	SYMSG
	<i>IV_NODE</i>	<i>type</i>	/BOBF/OBM_NODE_KEY
	<i>IV_KEY</i>	<i>type</i>	/BOBF/CONF_KEY
	<i>IV_ATTRIBUTE</i>	<i>type</i>	STRING
	<i>IV_ATTRIBUTE</i>	<i>type</i>	/BOBF/ CM_FRW=>TY_MES- SAGE_LIFETIME

Parameters

Parameter	Description
<i>IS_MSG</i>	Message that is to be added to the message object
<i>IV_NODE</i>	Node to be used in the origin location
<i>IV_KEY</i>	Instance key to be used in the origin location
<i>IV_ATTRIBUTE</i>	Attribute to be used in the origin location
	i Note If you want to specify more than one attribute, use Method ADD_CM [page 123] .
<i>IV_LIFETIME</i>	Lifetime of the message

2.7.6 Method CHECK

Returns true if the message object has messages with severity error.

Signature

<i>importing</i>	<i>IV_CONSISTENCY_MESSAGES</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>IV_ACTION_MESSAGES</i>	<i>type</i>	<i>BOOLE_D</i>
<i>returning</i>	<i>EV_ERROR_MESSAGES</i>	<i>type</i>	<i>BOOLE_D</i>

Parameters

Parameter	Description
<i>IV_CONSISTENCY_MESSAGES</i>	To indicate whether only messages of consistency validations are to be taken into consideration - <i>Optional</i> parameter
<i>IV_ACTION_MESSAGES</i>	To indicate whether only messages of action validations are to be taken into consideration - <i>Optional</i> parameter
<i>EV_ERROR_MESSAGES</i>	True if the message object contains error messages

2.7.7 Method GET

Returns all messages of the message object

Signature

<i>exporting</i>	<i>ET_MESSAGE</i>	<i>type</i>	<i>/BOBF/CM_FRW=>TT_FRW</i>
------------------	-------------------	-------------	--------------------------------

Parameters

Parameter	Description
<i>ET_MESSAGE</i>	Table of message instance that are contained in the message object

2.7.8 Method GET_MESSAGES

Returns a table of message specified by the importing parameters of the method

Signature

<i>importing</i>	<i>IV_SEVERITY</i>	<i>type</i>	<i>/BOBF/CM_FRW=>TY_MESSAGE_SEVERITY</i>
	<i>IV_CONSISTENCY_MESSAGES</i>	<i>type</i>	<i>BOOLE_D</i>
	<i>IV_ACTION_MESSAGES</i>	<i>type</i>	<i>BOOLE_D</i>
<i>exporting</i>	<i>ET_MESSAGE</i>	<i>type</i>	<i>/BOBF/T_FRW_MESSAGE_K</i>

Parameters

Parameter	Description
<i>IV_SEVERITY</i>	To indicate that only messages of a certain severity are to be returned - <i>Optional</i> parameter
<i>IV_CONSISTENCY_MESSAGES</i>	To indicate that only messages of consistency validations are to be returned - <i>Optional</i> parameter

Parameter	Description
<i>IV_ACTION_MESSAGES</i>	To indicate that only messages of action validations are to be returned - <i>Optional</i> parameter
<i>ET_MESSAGE</i>	Table of message instances

2.8 Semantics of Parameter *IV_BEFORE_IMAGE*

The parameter *IV_BEFORE_IMAGE* of the reading operations has different semantics - depending on the context when the application logic is called.

The following list shows the different meanings of the parameter, depending on the entity that is used by the consumer (when you set the parameter to *abap_true*).

Entity	Categories (for Determinations)	Semantics of Before Image
<i>Action</i>		The caller retrieves data that corresponds to the database state. Note that the database state is also buffered if the data is read multiple times.
<i>Action Validation</i>		If the consumer triggers validation by standard actions, the caller retrieves data corresponding to the current database state. If the validation is triggered by framework actions such as create, update, or delete, the caller retrieves the last saved <i>BEFORE MODIFICATION</i> state.
<i>Consistency Validation</i>		If the validation is triggered by create, update, or delete, the caller retrieves data corresponding to the state after last validation run. If the validation is triggered by a check, the caller retrieves data corresponding to the current database state.

Entity	Categories (for Determinations)	Semantics of Before Image
Association		The before image corresponds to the current state and is equal to the requested state of consumer.
Determination	Calculate transient attributes	This parameter can only be used during the execution time point After Modify (the execution time point is indicated with attribute <i>EXECTIME</i> = <i>/BOBF/</i> <i>IF_CONF_C=>SC_TIME_AFTER_MODIFY</i> of the importing structure <i>IS_CTX</i>). In this case, the caller retrieves data corresponding to the state after the last determination run.
	Calculate properties	The caller retrieves data corresponding to the current database state
	React after modification	The caller retrieves data corresponding to the state after last determination run
	React Before Save, React on Check and Determine, React before Save	The caller retrieves data corresponding to the current database state

3 Glossary

Before Image	A data image that is prior to the current transaction image. Depending of the context, the before image can have different semantics.
Combined Structure	Structure type of a node that contains the technical key attributes as well as the persistent and transient structures as an include
Combined Table	Table type of a node that has the <i>Combined Structure</i> type as its line type
Constants Interface	An ABAP interface that is dedicated for a specific business object. The interface includes constants for each business object's entity like nodes, attributes, actions and so on.
Content Change Notification	Indicates the change of a business object's entity. For example, a change of an association indicates that the result of an association might have changed. Consumers that have their own data buffers can use this kind of notification as a trigger to re-read the data.
Message Lifetime	Property of a message instance that can be used by UIs to decide how long a message will be displayed to the user. For example, only once after a roundtrip because a certain action was not possible, or permanently because the some BO data is inconsistent and will be corrected by the user
Message Symptom	Property of a message instance to indicate why the message has been issued. For example, because of a detected data inconsistency, locking conflict, and so on.
Model Key	A key that specifies an entity of a given business object. Model keys can be found as constants in the <i>Constants Interface</i> of the corresponding business object.
Node Instance	A data set of a business object node that is identified by its technical key
Origin Location	Provides the error location in a BO for an issued message

Technical Key	The primary key of a <i>node instance</i> . The primary key is stored in an attribute with name <i>KEY</i> along with the business-related attributes of a node.
Transactional Image	A data image that represents the state of the current transaction including changes that not have been committed to the database yet.

4 References

How to get started?

- [Introduction to BOPF](#)  - Overview
- [Getting Started with BOPF](#)  – guides you through all the steps required to create your first business object from scratch and to implement a basic operation using the BOPF API.

Tutorials:

- [Creating and Implementing Consistency Checks](#)  in Business Object Builder
- [Getting Started with Determinations](#)  in Business Object Builder
- [How to Create Queries](#)  with Business Object Builder
- [How to Use Authorization Checks in BOPF](#) 

Related blogs - published on ABAP community pages:

- [Business Object Overview](#) 
- [Working with BOPF API](#) 
- [Advanced BOPF API Features](#) 
- [Enhancements Techniques](#) 
- [Testing and UI integration](#) 

Tool reference on SAP help portal:

- using ABAP Development Tools (Eclipse)
- [BO Builder Help](#)

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
 - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
 - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

© 2020 SAP SE or an SAP affiliate company. All rights reserved.

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

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

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

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.