James Wood

January 29, 2013 | 8 minute read

Navigating the BOPF: Part 4 – Advanced BOPF API Features

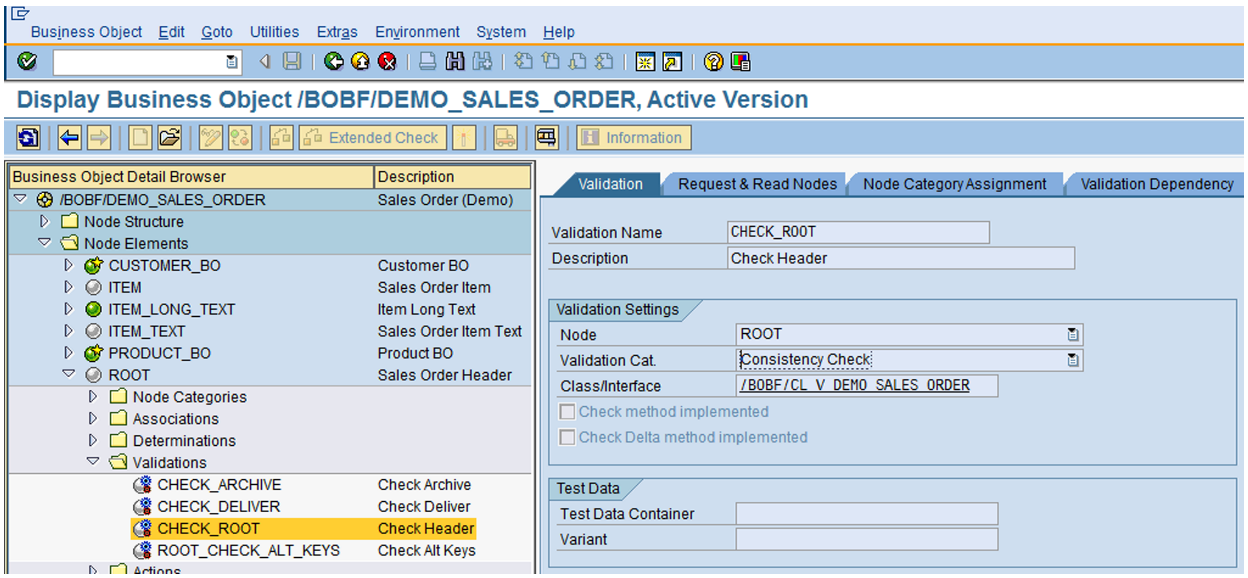
In my previous blog post, I introduced the BOPF API and demonstrated how the API could be used to perform routine CRUD operations on a business object. With this basic introduction out of the way, we're now ready to tackle more advanced API features such as consistency checks, the execution of actions, and transaction management. So, without further ado, let's get started.

Performing Consistency Checks & Validations

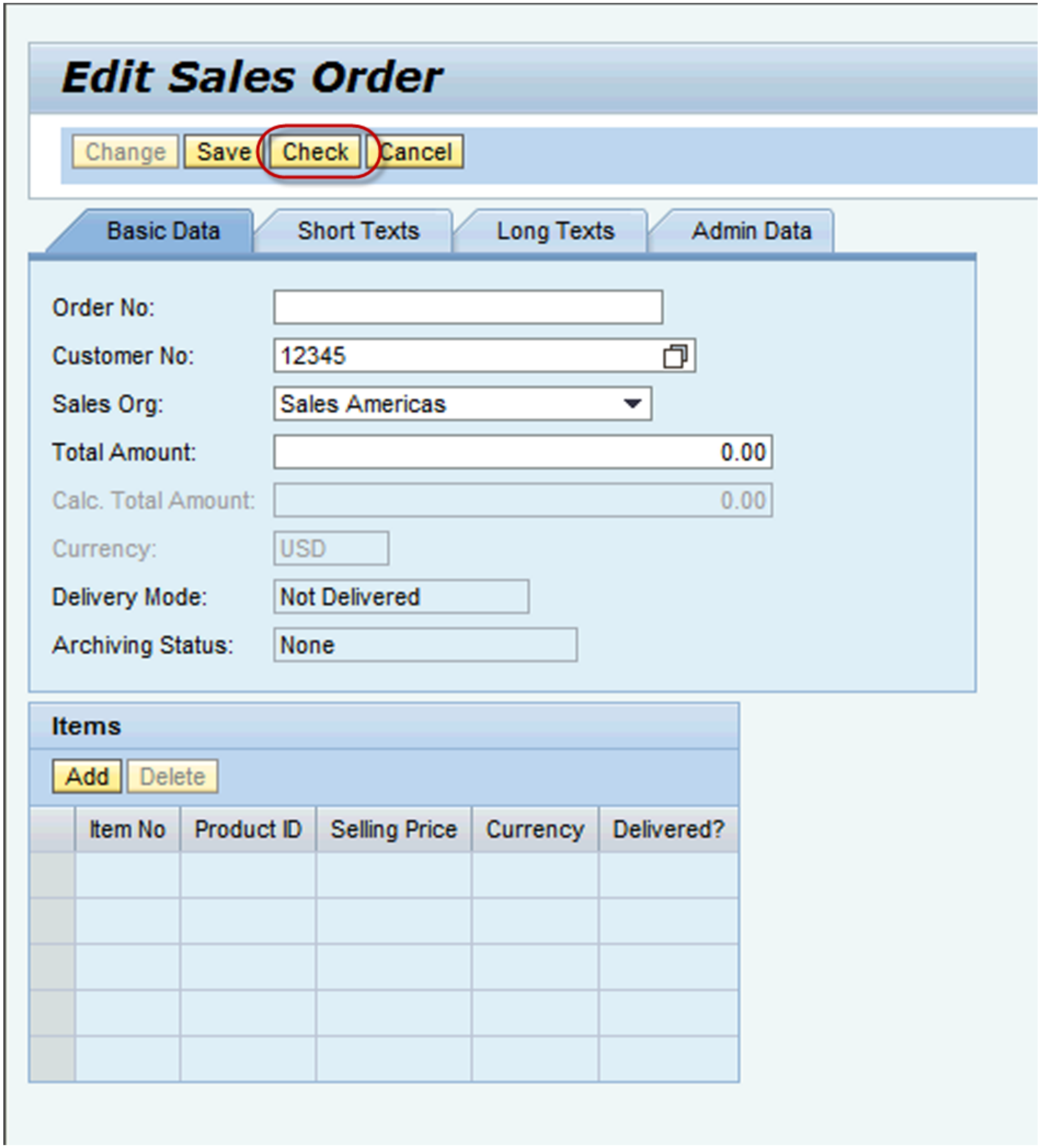
In keeping with the object-oriented paradigm, business objects (BOs) are designed to combine business data and business functions together into one tidy capsule (hence the term *encapsulation*). One of the primary benefits of combining these entities is to ensure that updates to business data are reliably filtered through a set of business rules. To put this concept into perspective, imagine a BO which defines a header-level status field (e.g. 01 = Initial, 02 = In Process, 03 = Closed). Now, from a pure data perspective, there's nothing stopping us from updating the status field using the

`MODIFY()` method of the `/BOBF/IF_TRA_SERVICE_MANAGER` interface (or heck, even via an SQL `UPDATE` statement). However, from a business perspective, there are probably some rules which define when, where, and how we should change the status field. For example, it might be that the BO cannot be closed until any open line items are closed out, etc.

Whatever the business rules might be, the point is that we want to ensure that a BO is consistent throughout each checkpoint in its object lifecycle. As we learned in part 2 of this blog series, the BOPF allows us to define these consistency checks in the form of *validations*. For example, in the screenshot below, you can see how SAP has created a validation called `CHECK_ROOT` for the `ROOT` node of the `/BOBF/DEMO_SALES_ORDER` demo BO. This validation is used to perform a consistency check on the sales order header-level fields to make sure that they are valid before an update is committed to the database.



One of the nice things about validations like `CHECK_ROOT` is that they are automatically called by the BOPF framework at specific points within the transaction lifecycle. However, sometimes we might want to trigger such validations interactively. For example, when building a UI on top of a BO, we might want to provide a check function which validates user input before they save their changes. This is demonstrated in the `/BOBF/DEMO_SALES_ORDER` Web Dynpro ABAP application shown below.


Edit Sales Order
Change
Save
Check
Cancel
Basic Data
Short Texts
Long Texts
Admin Data
Order No:
Customer No:
12345
Sales Org:
Sales Americas
Total Amount:
0.00
Calc. Total Amount:
0.00
Currency:
USD
Delivery Mode:
Not Delivered
Archiving Status:
None
Items
Add
Delete
Item No
Product ID
Selling Price
Currency
Delivered?

From a code perspective, the heavy lifting for the check operation is driven by the `CHECK_CONSISTENCY()` method of the `/BOBF/IF_TRA_SERVICE_MANAGER` interface as shown in the code excerpt below. Here, we simply provide the service manager with the target node key and the BO instance key and the framework will take care of calling the various validations on our behalf. We can then check the results of the validation by looking at the `/BOBF/IF_FRW_MESSAGE` instance which was introduced in the previous blog.

```
DATA lt_key TYPE /bobf/t_frw_key.

FIELD-SYMBOLS <ls_key> LIKE LINE OF lt_key.

DATA lo_message TYPE REF TO /bobf/if_frw_message.

TRY.

  APPEND INITIAL LINE TO lt_key ASSIGNING <ls_key>.

  <ls_key>-key = iv_key.     "<== Sales Order BO Key

  CALL METHOD mo_svc_mngr->check_consistency

    EXPORTING

      iv_node_key    = /bobf/if_demo_sales_order_c=>sc_node-root

      it_key         = lt_key

      iv_check_scope = '1'

    IMPORTING

      eo_message     = lo_message.

  ...

CATCH /bobf/cx_frw INTO lx_frw.
```

```
  ...

ENDTRY.
```

I'll show you how to implement validations within a BO in an upcoming blog entry.

Triggering Actions

The behaviors of a business object within the BOPF are defined as *actions*. From a conceptual point-of-view, actions are analogous to methods/functions in the object-oriented paradigm. The following code excerpt demonstrates how actions are called using the BOPF API. Here, we're calling the `DELIVER` action defined in the `ROOT` node of the `/BOBF/DEMO_SALES_ORDER` demo BO. As you can see, the code reads like a dynamic function/method call since we have generically pass the name of the action along with its parameters to the `DO_ACTION()` method of the `/BOBF/IF_TRA_SERVICE_MANAGER` interface. Other than that, it's pretty much business as usual.

```
DATA lt_key TYPE /bobf/t_frw_key.

FIELD-SYMBOLS <ls_key> LIKE LINE OF lt_key.

DATA ls_parameters     TYPE /bobf/s_demo_sales_order_hdr_d.

DATA lr_s_parameters   TYPE REF TO data.

DATA lo_change         TYPE REF TO /bobf/if_tra_change.

DATA lo_message        TYPE REF TO /bobf/if_frw_message.

DATA lt_failed_key     TYPE /bobf/t_frw_key.

DATA lt_failed_act_key TYPE /bobf/t_frw_key.



TRY.
```

```
"Set the BO instance key:

APPEND INITIAL LINE TO lt_key ASSIGNING <ls_key>.

<ls_key>-key = iv_key.     "<== Sales Order BO Key



"Populate the parameter structure that contains parameters passed

"to the action:

ls_parameters-item_no = '001'.

GET REFERENCE OF ls_parameters INTO lr_s_parameters.



"Call the DELIVER action defined on the ROOT node:

CALL METHOD mo_svc_mngr->do_action

  EXPORTING

    iv_act_key           = /bobf/if_demo_sales_order_c=>sc_action-root-deliver

    it_key               = lt_key

    is_parameters        = lr_s_parameters

  IMPORTING

    eo_change            = lo_change

    eo_message           = lo_message
```

```
      et_failed_key        = lt_failed_key

      et_failed_action_key = lt_failed_act_key.



    ...

CATCH /bobf/cx_frw INTO lx_frw.

  ...

ENDTRY.
```

We can verify if a BOPF action call was successful by querying the `EO_MESSAGE` object reference and/or the `ET_FAILED_KEY` table parameters returned by the `DO_ACTION()` method. Refer back to my previous blog for an example of the former technique. As always, remember to commit the transaction using the

`SAVE()` method defined by the `/BOBF/IF_TRA_TRANSACTION_MGR` interface.

Action Validations

In part two of this blog series, I mentioned that there are technically two different types of validations: the consistency check validations discussed earlier in this blog and action validations which are used to determine whether or not an action can be executed at runtime. Since the BOPF framework invokes action validations automatically whenever actions are invoked, it is rare that you will have a need to invoke them directly. However, the `/BOBF/IF_TRA_SERVICE_MANAGER` interface does provide the `DO_ACTION()` method if you wish to do so (see the code excerpt below).

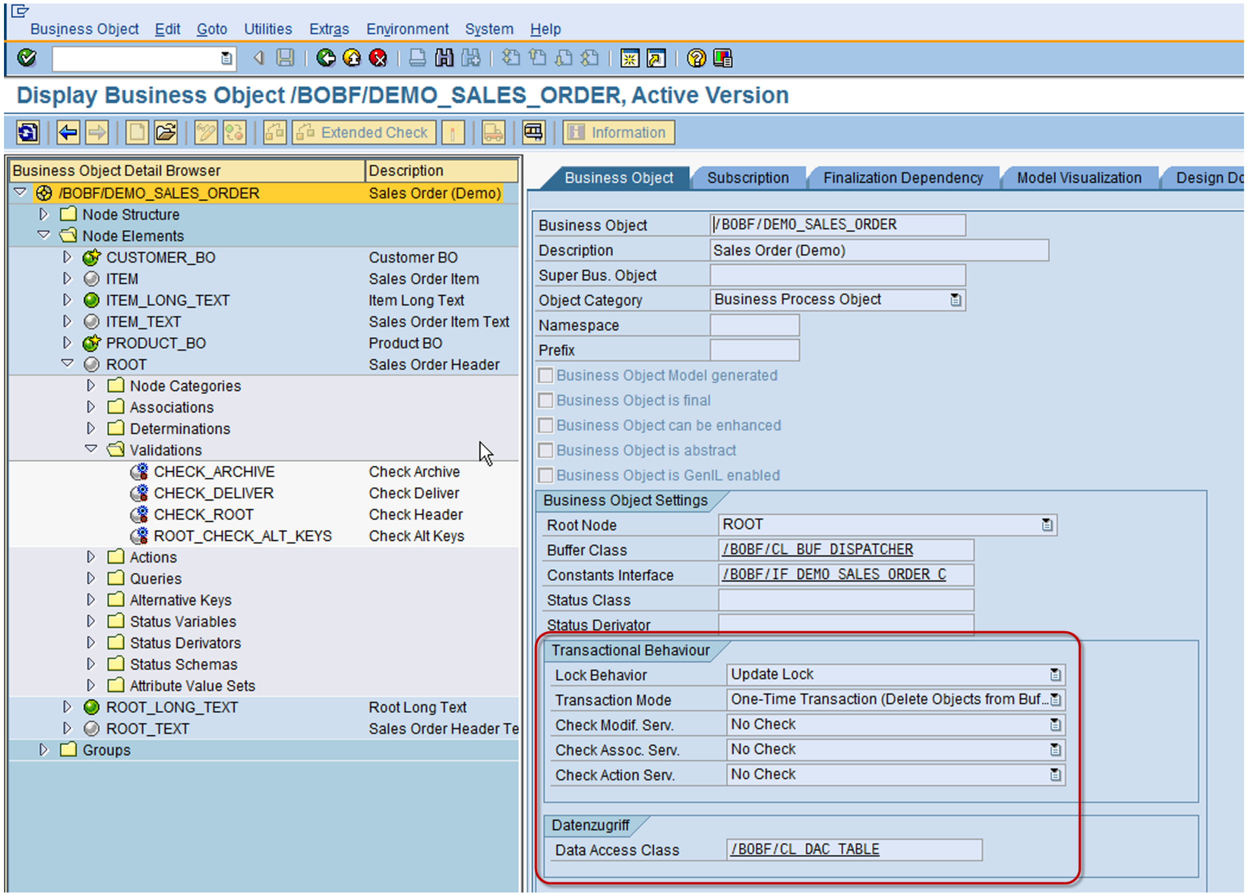
```
DATA lt_key TYPE /bobf/t_frw_key.

FIELD-SYMBOLS <ls_key> LIKE LINE OF lt_key.

DATA ls_parameters     TYPE /bobf/s_demo_sales_order_hdr_d.

DATA lr_s_parameters   TYPE REF TO data.
```

```
DATA lo_message         TYPE REF TO /bobf/if_frw_message.

DATA lt_failed_key      TYPE /bobf/t_frw_key.

DATA lt_failed_act_key  TYPE /bobf/t_frw_key.



TRY.

  "Set the BO instance key:

  APPEND INITIAL LINE TO lt_key ASSIGNING <ls_key>.

  <ls_key>-key = iv_key.     "<== Sales Order BO Key



  "Populate the parameter structure that contains parameters passed

  "to the action:

  ls_parameters-item_no = '001'.

  GET REFERENCE OF ls_parameters INTO lr_s_parameters.



  "Check to see if the DELIVER action can be invoked:

  CALL METHOD mo_svc_mngr->check_action

    EXPORTING

      iv_act_key           = /bobf/if_demo_sales_order_c=>sc_action-root-deliver
```

```
      it_key               = lt_key
      is_parameters        = lr_s_parameters
    IMPORTING
      eo_message           = lo_message
      et_failed_key        = lt_failed_key
      et_failed_action_key = lt_failed_act_key.


    ...
CATCH /bobf/cx_frw INTO lx_frw.
  ...
ENDTRY.
```

Transaction Management

Another element of the BOPF API that we have glossed over up to now is the transaction manager interface `/BOBF/IF_TRA_TRANSACTION_MGR`. This interface provides us with a simplified access point into a highly sophisticated transaction management framework. While the details of this framework are beyond the scope of this blog series, suffice it to say that the BOPF transaction manager does more here than simply provide basic object-relational persistence. It also handles caching, transactional locking, and more. You can see how some of these features are implemented by looking at the *Transactional Behavior* settings of a business object definition in Transaction /BOBF/CONF_UI (see below).



So far, we have seen a bit of the `/BOBF/IF_TRA_TRANSACTION_MGR` on display whenever we looked at how to insert/update records. Here, as you may recall, we used `SAVE()` method of the `/BOBF/IF_TRA_TRANSACTION_MGR` interface to save these records. In many respects, the `SAVE()` method is analogous to the `COMMIT WORK` statement in ABAP in that it commits the transactional changes to the database. Here, as is the case with the `COMMIT WORK` statement, we could be committing multiple updates as one logical unit of work (LUW) – e.g. an insert followed by a series of updates.

Once a transaction is committed, we can reset the transaction manager by calling the `CLEANUP()` method. Or, alternatively, we can also use this method to abandon an in-flight transaction once an error condition has been detected. In the latter case, this is analogous to using the `ROLLBACK WORK` statement in ABAP to rollback a transaction.

During the course of a transaction, the BOPF transaction manager tracks the changes that are made to individual business objects internally so that it can determine what needs to be committed and/or rolled back. If desired, we can get a peek of the queued up changes by calling the `GET_TRANSACTIONAL_CHANGES()` method of the

`/BOBF/IF_TRA_TRANSACTION_MGR` interface. This method will return an object reference of type `/BOBF/IF_TRA_CHANGE` that can be used to query the change list, modify it in certain cases, and so on.

Next Steps

At this point, we have hit on most of the high points when it comes to interacting with the BOPF API from a client perspective. In my next blog, we'll shift gears and begin looking at ways of enhancing BOs using the BOPF toolset.

Assigned Tags

ABAP Development

abap

bopf

floorplan manager

Web Dynpro

Similar Blog Posts

One truth, multiple views on it: The various BOPF modeling environments

By Oliver Jaegle Sep 03, 2015

BOPF-SADL Mapping – Demystifying Limitations

By Ivo Vollrath Jun 06, 2016

BOPF: Custom Lock/Unlock Action for Legacy DAC

By Gaurav Sharan Jun 29, 2018

Related Questions

How to call another BOPF from action of one BOPF ?

By Surya Sarathi Basu Dec 26, 2018

Where the integration (link) of Bopf_ewb and Fpm done

By Former Member Nov 25, 2015

BOPF-Business Object

By mustafa cengiz Aug 17, 2020

17 Comments

Oliver Jaegle

February 25, 2014 at 2:21 pm

Dear James,

in your sample about executing an action, you pass an ID as parameter.

If this is the ID of the sales order item you're "delivering", this action design was one I'd discourage.

Let me try to explain that: Usually, the instance(s) you are executing an action on is/are referred to by its technical key (which is passed as part of IT_KEY).

The parameter shall be used for passing variable options which influence the behavior of the action (e. g. delivering a split quantity which makes the action create a subsequent delivery item).

You should only pass the reference to the instance as an action parameter if you intend not to operate on this instance but to use it as parameter of another logic (e. g. if you create a new instance based on the instance given in the parameter as template).

The advantage of is obvious if you intend to prevent the execution of the action with a validation or when it comes to mass-processing: You can always refer to the KEYs which are passed to the action: You can validate them, exclude some of them from the execution as they failed to pass a validation (failed keys), you can enable or disable the action using properties.

If you use a static action, you will be redundantly converting the parameter into a technical key within the implementations and you cannot refer to the instance in created message objects.

Hope I could make it a bit clear.

Oliver

Like 0 | Share

James Wood | Blog Post Author

February 25, 2014 at 2:33 pm

Hi Oliver,

This action is not something I defined; rather it's part of a demo BO provided by SAP. Here, the DELIVER action defines a parameter structure which receives the item number of the sales order item being delivered. I agree that this is not the best of conventions, but my purpose with this article is more to demonstrate how to work with the API and pass parameters than it was to critique the validity of the sample BOs. Make sense?

Thanks,

James

Like 0 | Share

Oliver Jaegle

February 25, 2014 at 2:43 pm

Hi James,

Of course, your demo for using the APIs makes perfect sense. I also understood it that way.We'll have to talk to the BOPF guys for preparing better samples 😉

However, with your blog being the prime source for everyone who has a first glance at the framework, I wanted to comment on the sample for clarifying that instances and parameters are not interchangeable - and this difference is also part of the API-usage.

Cheers, Oliver

Like 0 | Share

Former Member

September 3, 2014 at 12:26 pm

Excellent post! Thanks for the information. I have a query regarding the service manager.

My UI right now shows the current BO data (persisted in the DB) in read mode. If some other application changes this BO data later, how does the service manager know that the data in the DB has changed? How will my UI display the right(current) data?

Thanks, Suhas

Like 0 | Share

James Wood | Blog Post Author
September 3, 2014 at 5:58 pm

Hi Suhas,

Hmm, that's a bit of a loaded question. The FBI (view) layer is generally aware of changes to BOPF records, so it does have the ability to refresh data. If you can provide me with more specifics, I could comment further.

Thanks,

James

Like 0 | Share

Former Member
September 4, 2014 at 5:12 am

Hi James,

Let's take an example. We have a customer BO and have built a FBI UI on it to display the data. We also have the mode handling in place. If the UI is right now displaying data for ID=ABC in read mode(APPL1) and another application(APPL2) updates the data for the same ID(e.g. change of status or review date), how will APPL1know that the data has been updated? Does the APPL1 need to do anything specific to update itself?

Hope my context is clear now.

Thanks,

Suhas.

Like 0 | Share

James Wood | Blog Post Author

September 4, 2014 at 1:10 pm

Hi Suhas,

A couple of questions:

1. Is APPL2 also a UI application?
2. If so, are we assuming that APPL2 has committed the changes?

Thanks,

James

Like 0 | Share

Former Member

September 4, 2014 at 2:27 pm

Hi James,

The APPL2 could be a background process too. But it is safe to assume that the changes are committed.

Thanks,

Suhas

Like 0 | Share

Oliver Jaegle

September 4, 2014 at 2:09 pm

Dear Suhas,

Two separate browser windows running a webdynpro-application are isolated sessions in the backend. Meaning: They don't know anything about each other and there are no in-built-mechanisms to notify each other about modified data.

Assuming that the changes in the separate session are committed, they will be visible to other sessions. As APPL1 has already loaded its state though, APPL1 will have to actively re-read the newly committed data in order to see it.

This is what a good UI-controller should anyway do: Before switching from read-only to edit-mode, re-read the current data and - in the very same request - lock it exclusively. And this is exactly what FBI does: On the EDIT-event, all UIBBs are requested to reload from the database (technically, a retrieve with edit mode exclusive is being performed).

This way, the APPL2 will appear to have been notified about the changes of the other session.

In contrast to that, the changes occurring within a session are propagated differently: Each core-service via the service manager returns a change-object (/bobf/if_tra_change) which provides the caller with information which data has been modified (by the consumer itself or implicitly within a determination). Based upon this change-object, it's up to the consumer (e. g. the feeder) to re-read the data (e. g. in order to pass it into the UI-structures). This is what the generic FBI-components provide.

Hope I could clarify some aspects of change handling,

Oliver

Like 0 | Share

Former Member

September 4, 2014 at 2:25 pm

Dear Oliver,

Thanks for the explanation. I now understand that the FBI controller re-reads the data again via an implicit call to retrieve. However, for the FBI based UI(APPL1), I need to implement a "refresh" kind of function without changing the read-only mode. Is it possible?

Triggering the "FPM_REFRESH" does not automatically update the data on the UI. My understanding is that "FPM_REFRESH" does not reload the buffer.

Thanks,

Suhas

Like 0 | Share

Oliver Jaegle

September 5, 2014 at 6:22 am

As per the refresh-action, this is something I don't think is provided "out-of-the-box" by FBI, but I believe you have to on the event by yourself. I couldn't test this in my system now, but I assume that you should be fine doing by

- Having a button raise an event for the refresh-request (whether you use the FPM_REFRESH or a custom name does not matter technically - I don't know the semantics of the FPM_REFRESH event, you may ask in the FPM space)
- As feeder of the UIBB which represents the root of the data which shall be refreshed (there might be dependent nodes in other UIBBs), configure a custom one: Inherit from the FBI-feeder, redefine process_event.
- When processing the event, retrieve the data again (requesting to read the current image and lock it
- Propagate the change-object

The rest (refreshing the data in the UI-structures and in dependent (wired) UIBBs should be taken care of by FBI without further ado.

Let us know (either here in the comments or an own blogpost) whether and how you succeeded!

Cheers,

Oliver

Like 0 | Share

Thirumoorthy Rajaram
February 20, 2016 at 10:11 am

Hi James,

Very excellent post!!..

I have a small query. I have created an action, when i am trying to trigger that action using the DO_ACTION method provided by the trx_mgr, it is not getting triggered. Failed key table is filled. I am not getting what exactly is happening, tried debugging but no luck. Could you please me what exactly i am missing in calling the action. The key which i am passing as an import parameter is the SC_BO_KEY of my custom object created by me.

Regards,

Thiru

Like 0 | Share

James Wood | Blog Post Author
February 22, 2016 at 8:10 pm

Hi Thiru,

I'd need more context to go on to comment on this I think. Send me further details and I'll try to help as best I can.

Thanks,

James

Like 0 | Share

Thirumoorthy Rajaram
February 24, 2016 at 8:11 am

Hi James,

Thank you very much for the reply!!

The issue which I have raised in my previous comment is solved now. First of all I want to let you know that your blog and the docs related to BOPF framework is very nice and understandable, in simple words it is awesome for someone like me who is a new comer. Thank you for sharing the knowledge.

I have some more questions for you, would say as clarification

My requirement is to have 3 nodes

Root->Parent->Child

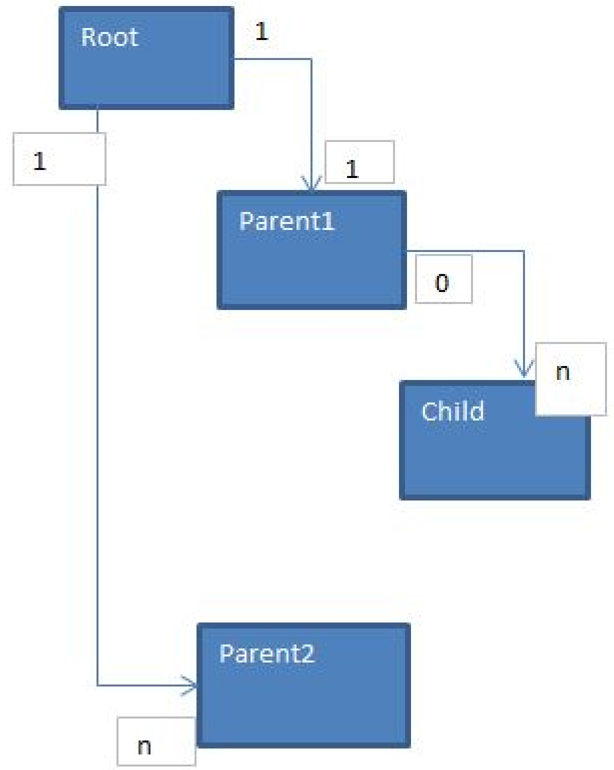
The cardinality from ROOT to PARENT is 0..1 and

The cardinality from PARENT to CHILD is 1..N

Is there any option or possibility to restrict the records based on cardinality of the nodes by caller program(or class) or any configuration set up. we tried by setting the association cardinality of Root(1..1) and Parent node(1..1) but when tried creating a record in BOBT it is allowing more than one record even though cardinality is 1…1.

Our main requirement is to have only one record at root and parent node, but child node can have multiple records corresponding to parent node key. Please find the below image of our BOPF object structure in pictorial representation.

Would be really nice if you can guide us!!



Regards,

Thiru

Like 0 | Share

Oliver Jaegle

February 24, 2016 at 10:05 am

Hi Thiru,

BOPF does not automatically validate cardinalities as this is not always requested and as every validation comes with a performance penalty.

However, BOPF offers a generic validation implementation (/BOBF/CL_LIB_V_ASSOC_CA) which you can configure. Please find the details in the class-documentation:

CL /BOBF/CL_LIB_V_ASSOC_CARD

Short Text

Validation: Checks if association cardinality is consistent

Functionality

Generic validation to check the cardinality of associations. The association cardinalities are checked for the node where the validation is configured.

The associations are retrieved and for all associations

- which have cardinality **one** or **one-to-many**, it is checked, whether at least one node instance is existing.
- which have cardinality **zero to one** or **one**, it is checked, whether at most one node instance is existing.

Relationships

The validation is intended to be used as action validation at "check before save" .

Notes

As the validation is quite generic is first of all intended for prototype use or during an early implementation state. If performance problems occur the validation implementation should be replaced by a specific implementation.

Cheers,

Oliver

P.s.: I would recommend to ask such good questions in the BOPF community: BOPF Application Framework. This has multiple advantages: More persons knowing about your question, more people sharing the problem and answer afterwards, I don't have a mixed feeling about answering questions directed to James

Like 0 | Share

Thirumoorthy Rajaram
February 24, 2016 at 5:04 pm

Hi Oliver,

Thank you for the reply. Next time would make sure that I would post questions in the BOPF community.

Looking into your suggestions for the validations. Will check and let you know:) Thank you!!

Regards,

Thiru

Like 0 | Share

Christopher Solomon
March 13, 2019 at 5:57 pm

First off, GREAT series!!! Loving it.....although, not a fan of learning yet ANOTHER BO framework from SAP. hahahaha Just a quick question/correction..

In the ACTION VALIDATIONS section, you say "However, the `/BOBF/IF_TRA_SERVICE_MANAGER` interface does provide the `DO_ACTION()` method if you wish to do so (see the code excerpt below)."

I think you mean the CHECK_ACTION() method, correct?

Like 0 | Share