James Wood

March 4, 2013 | 6 minute read

Navigating the BOPF: Part 6 – Testing & UI Integration

14 12 21,012

Follow

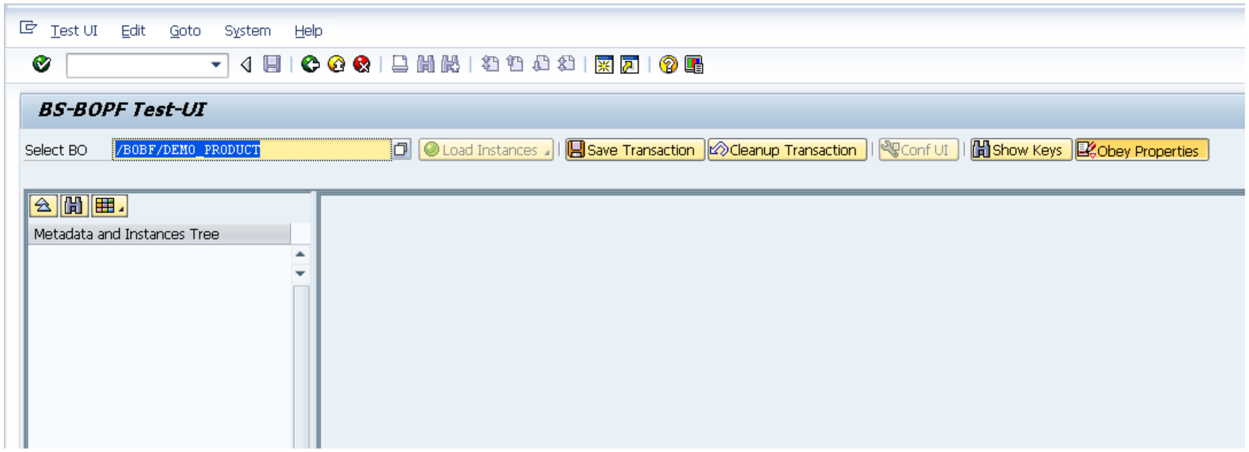
Like

RSS Feed

In my previous blog post, we looked at ways of enhancing business objects (BOs) defined within the BOPF. Once these enhancements are in place, it is a good idea to unit test the changes to make sure that the BO is working correctly. If you subscribe to the test-driven development (TDD) model, then an obvious choice here would be to use the BOPF API to write some ABAP Unit test cases. However, sometimes we just want to run a quick ad hoc test. Or, we might just want to display BO data at a glance without constructing a multi-table join in Transaction SQVI. For these tasks and others, SAP provides us with a very slick tool: the BOPF Test UI. In this final blog post, I will introduce this transaction and demonstrate its basic capabilities. Then, to round things out, I'll briefly touch on UI integration and the FBI framework.

Working with the BOPF Test UI

We can access the BOPF Test UI by opening up Transaction /BOBF/TEST_UI. When you initially open up the tool, you'll be presented with a screen like the one shown below. From here, we can begin working with a BO instance by plugging in the BO type in the *Select BO* input field (either key it in or use the provided input help) and hitting the ENTER key. This causes the BO metadata to be loaded into context so that we can use it to guide ourselves through the editing process.

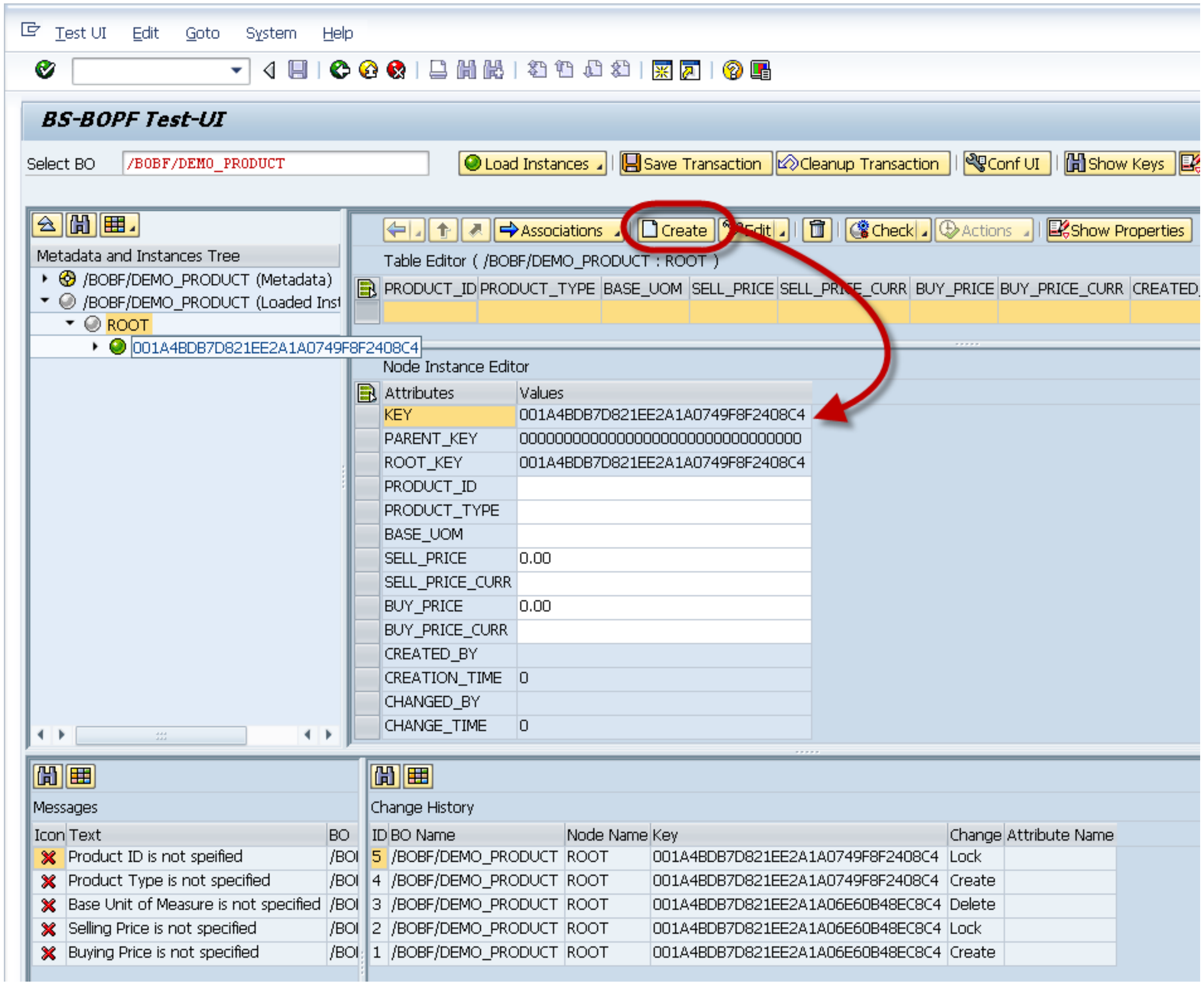


Editing BO Instances

Once the BO metadata is loaded, we have two choices for maintenance:

1. To create a new BO instance, we can double-click on the root node contained in the "Metadata and Instances" tree on the left-hand side of the editor screen and then select the Create button in the toolbar (see below). This will cause a new record to be created and loaded into an editable ALV grid. From here, we can begin filling in node attributes, creating sub-node instances, and so on. Here, I would draw your attention to the *Messages* panel located in the

bottom left-hand corner of the editor. These messages can be used to help you fill in the right data.



BS-BOPF Test-UI

Select BO /BOBF/DEMO_PRODUCT

Metadata and Instances Tree

- /BOBF/DEMO_PRODUCT (Metadata)
- /BOBF/DEMO_PRODUCT (Loaded Inst
  - ROOT
    - 001A4BDB7D821EE2A1A0749F8F2408C4

Table Editor (/BOBF/DEMO_PRODUCT : ROOT)

PRODUCT_ID	PRODUCT_TYPE	BASE_UOM	SELL_PRICE	SELL_PRICE_CURR	BUY_PRICE	BUY_PRICE_CURR	CREATED_

Node Instance Editor

Attributes	Values
KEY	001A4BDB7D821EE2A1A0749F8F2408C4
PARENT_KEY	00000000000000000000000000000000
ROOT_KEY	001A4BDB7D821EE2A1A0749F8F2408C4
PRODUCT_ID	
PRODUCT_TYPE	
BASE_UOM	
SELL_PRICE	0.00
SELL_PRICE_CURR	
BUY_PRICE	0.00
BUY_PRICE_CURR	
CREATED_BY	
CREATION_TIME	0
CHANGED_BY	
CHANGE_TIME	0

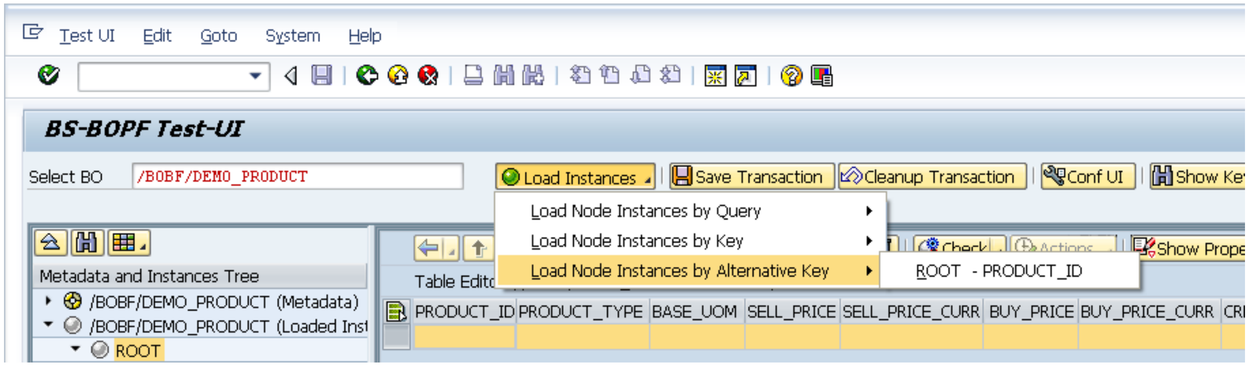
Messages

Icon	Text	BO
✗	Product ID is not speified	/BOI
✗	Product Type is not specified	/BOI
✗	Base Unit of Measure is not specified	/BOI
✗	Selling Price is not specified	/BOI
✗	Buying Price is not specified	/BOI

Change History

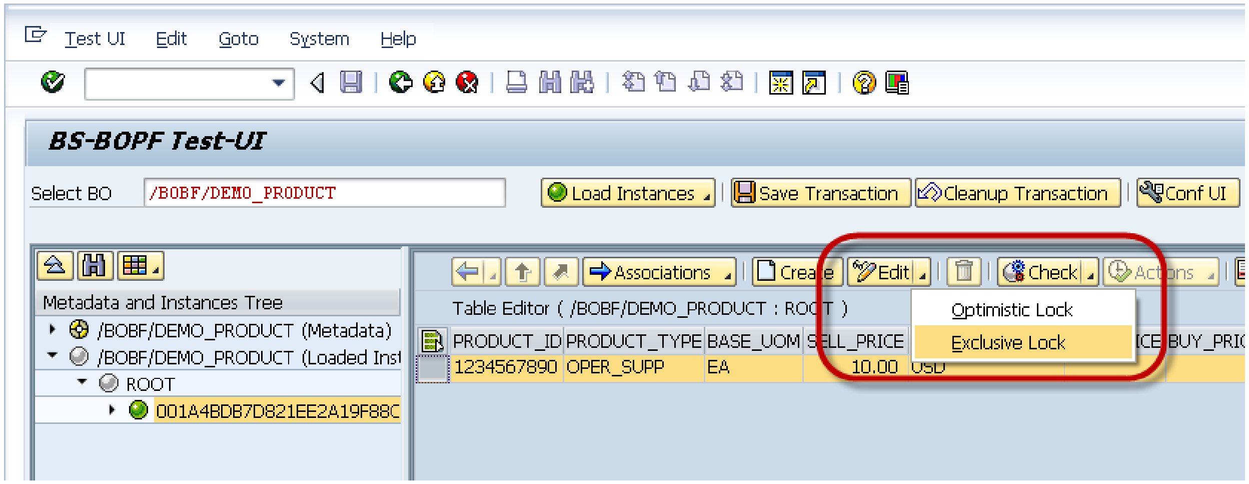
ID	BO Name	Node Name	Key	Change	Attribute Name
5	/BOBF/DEMO_PRODUCT	ROOT	001A4BDB7D821EE2A1A0749F8F2408C4	Lock	
4	/BOBF/DEMO_PRODUCT	ROOT	001A4BDB7D821EE2A1A0749F8F2408C4	Create	
3	/BOBF/DEMO_PRODUCT	ROOT	001A4BDB7D821EE2A1A06E60B48EC8C4	Delete	
2	/BOBF/DEMO_PRODUCT	ROOT	001A4BDB7D821EE2A1A06E60B48EC8C4	Lock	
1	/BOBF/DEMO_PRODUCT	ROOT	001A4BDB7D821EE2A1A06E60B48EC8C4	Create	

2. If the BO instance that we want to maintain/display exists already, then we can load it into context using the *Load Instances* button menu. As you can see in the screenshot below, this menu affords us with several different alternatives for loading node instances: via a BOPF node query, by the node instance key, or by an alternative key (e.g. ID). Regardless of the menu path that we take, the system will attempt to find the target node instance(s) and then load them into the editor window. From here, we can select individual node instances by double-clicking on them in the *Metadata and Instances* tree located on the left-hand side of the screen.



PRODUCT_ID	PRODUCT_TYPE	BASE_UOM	SELL_PRICE	SELL_PRICE_CURR	BUY_PRICE	BUY_PRICE_CURR	CRI

To edit node instances, we can select the node instance record in the editor on the right-hand side of the screen and choose the appropriate option from the *Edit* button menu (see below). Then, we can edit attributes for a node instance using the provided input fields. Alternatively, we also have the option of deleting a node instance (or indeed an entire BO instance in the case of a root node instance) by clicking on the *Delete Node Instances* button.

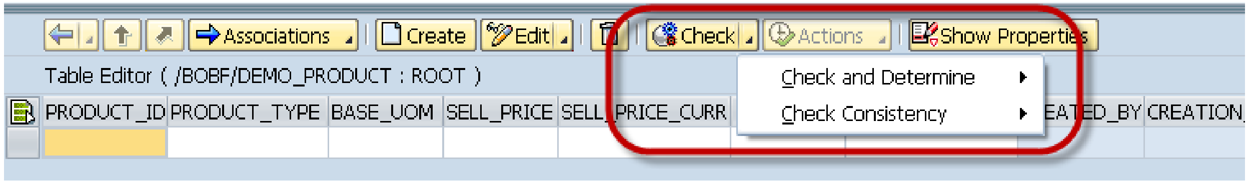


PRODUCT_ID	PRODUCT_TYPE	BASE_UOM	SELL_PRICE		ICE	BUY_PRIC
1234567890	OPER_SUPP	EA	10.00	USD		

Regardless of whether or not we're creating a new BO instance or editing an existing one, the entire scope of our changes is tracked via a BOPF transaction like the one we would create if we were doing all this by hand using the BOPF API. At any point along the way, we can choose to commit the changes using the *Save Transaction* button, or revert the changes using the *Cleanup Transaction* button. Then, we can start the process over by selecting another BO instance or editing the existing one in place. All in all, it's kind of like table maintenance on steroids. But wait, there's more!

Triggering Actions, Validations, & Determinations

In addition to the basic CRUD operations described earlier, the test UI also provides functions to call actions, validations, and even trigger determinations. For a given node instance, these functions can be accessed in the node instance toolbar via the *Check* and *Actions* button menus (see below). If you read through my blog posts related to the BOPF API, then these should feel quite intuitive.



UI Integration and the FBI Framework

Since the focus of this blog series has been primarily on introducing the BOPF framework, I have purposefully avoided digressing into specific applications of the BOPF (e.g. in Transportation Management or EHSM) since these products add additional layers on top of the BOPF that can sort of cloud the picture a bit if you don't understand core principles of the BOPF itself. However, before I bring this blog series to a close, I would be remiss if I didn't point out one important (and relatively generic) framework built on top of the BOPF: the *Floorplan Manager BOPF Integration* (FBI) framework. As the name suggests, this framework links BOs from the BOPF with Web UIs based on the Floorplan Manager (FPM) framework and Web Dynpro ABAP (WDA).

If you're developing Web UIs on top of BOs from the BOPF, then the FBI is definitely something to take a look at. Essentially, the FBI exploits the genericity of the BOPF API and the accessibility of BO model data to enable the rapid development of *Generic User Interface Building Blocks* (GUIBBs) based on BO nodes. Here, for example, we could create a form GUIBB that allows users to populate the data for a BO node using a simple input form. In many applications, this can be achieved without having to write a single line of code. While a detailed discussion of the FBI is beyond the scope of this blog series, a quick Google search will lead you to some pretty decent resource materials. If you're new to FPM, I would also offer a shameless plug for my book Web Dynpro ABAP: The Comprehensive Guide (SAP PRESS, 2012).

Conclusion

When I first started working with the BOPF almost a year ago, I was surprised at how little documentation there was to get started with. So, what you've seen in this series is the result of a lot of trial-and-error and lessons learned by debugging past application-specific frameworks into the heart of the BOPF itself. If you're just getting started with the BOPF, then I hope that you'll find this series useful to get you up and running. In the coming months and years, I think many more learning resources will materialize to supplement what I've offered here. Indeed, the number of new dimension applications based on the BOPF appears to be growing by the day…

One complaint I sometimes hear from other developers is that the BOPF API is cumbersome to work with. On this point, I can agree to a point. However, I would argue that such complexities can be abstracted away pretty easily with a wrapper class or two and some good old fashioned RTTI code. Other than that, once you get used to the BOPF, I think you'll find that you like it. And this is coming from a developer who has had many bad experiences with BO frameworks (both in and outside SAP…curse you EJBs!!!). All in all though, I have found the BOPF to be very comprehensive and flexible. For me, one of the feel tests I normally conduct to gauge the effectiveness of a framework is to ask myself how often the framework gets in my way: either because it's too intractible, limited in functionality or whatever. I have yet to run into any such occurrences with the BOPF. It does a good job of providing default behaviors/functionality while at the same time affording you the opportunity to tweak just about everything. For example, if I want to build my own caching mechanism, I can do so by plugging in my own subclass. If I want to pull data from a HANA appliance in real time, I can do so in a determination. You get the idea. It's all there, so just poke around a bit and I think you'll find what you need.

Alert Moderator

Assigned Tags

ABAP Development

abap

bopf

fbi

floorplan manager

fpm

Web Dynpro

View more...

Similar Blog Posts

Creating a draft enabled Sales Order Fiori App using the new ABAP Programming Model - Part 1: Overview

By Geert-Jan Klaps Mar 11, 2019

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

By Oliver Jaegle Sep 03, 2015

Where to find learning material about BOPF?

By Martin Fischer May 01, 2018

Related Questions

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

How to create a BOPF message by hand?

By Jerry Zhang May 28, 2019

14 Comments

You must be Logged on to comment or reply to a post.

Former Member

March 6, 2013 at 10:32 am

Hi James,

its very nice series of blog.Now i hope the BOPF stuff is available in SCN with your contibution !

Like 0 | Share

Bharath Komarapalem

March 13, 2013 at 2:04 pm

Hi James,

I should thank and appreciate your efforts for preparing such blog series. I just started with SAP TM and BOPF. I am in the same situation now in what situation you were 1 year back.

But your blog is really helpful. Although it is very difficult to understand and digest in the first read but i hope over and after some period there will be some improvement in understanding..

Nice blog and nicely documented.

Regards,

Bharath.

Like 0 | Share

Former Member

April 11, 2013 at 4:00 am

Hi James,

It was very nice blog. Now, I hope the BOPF stuff are avaible in community..

Thanks,

kris.

Like 0 | Share

Former Member

August 27, 2013 at 7:57 pm

Really good stuff james

Like 0 | Share

Former Member

September 26, 2013 at 1:26 am

It is Really Good James..

Like 0 | Share

Sagar Pilli

October 1, 2013 at 5:21 am

Hi James,

Its really very good stuff and I encourage who ever comes on my way to work on BOPF to refer your blogs with any doubt.

As on date, this blog is the priceless source of Information on BOPF.

Cheers

Sagar

Like 0 | Share

Zhenbo Wang

April 9, 2014 at 10:30 am

Hi James,

great article. It has changed my opinion regarding the BOPF framework (more or less).

Do you know a way to convert the bopf-key to a readable name during debuging? Search it in the constant interface is disturbing.

thanks and regards,

Zhenbo

Like 0 | Share

James Wood | Blog Post Author
April 9, 2014 at 11:13 am

Hi Zhenbo,

Unfortunately, no. This is a pain point for sure. Sorry I don't have a better answer for you.

Thanks,

James

Like 0 | Share

Paul Hardy
January 8, 2015 at 11:17 am

Mr.James,

Have you ever got change documents working in the BOPF with external keys?

I can get entries popping up in CDHDR and CDPOS easily enough, but redefining the method GET_EXTERNAL_IDS from a subclassof /BOFU/CL_CDO_BO_GENERICCALLBACK never gets called i.e. the breakpointis never reached, although the rdefinition of IS_CDO_CREATION does.

Having an external ID in the change document log would be lovely.

The code seems to be looking in table TCDOB which I think is the SCDO and then seeing if the entry is a table type as opposed to a transparent table. Only then is the GET_EXTERNAL_ID called. If this is true then you have to define your Z table type with a really short name, as the SCDO transaction still thinks we are in version 3 with a limit of 16 characters on DB table names.

Cheersy Cheers

Paul

Like 0 | Share

James Wood | Blog Post Author

January 8, 2015 at 2:45 pm

Hi Paul,

I've used these callback classes as a hook for triggering downstream interfaces, but I've never tried to implement a scenario like what you're describing. That being said, have you tried setting a break-point in the PCASBGDTCRD_GET_EXT_IDS() method of class /BOFU/CL_CHANGE_DOC_HANDLING? It looks like this is the launching point for the external ID handling. Maybe you can glean some insight from there (or see how you might enhance the framework to support what you're looking for)?

Thanks,

James

Like 0 | Share

Paul Hardy

January 8, 2015 at 8:48 pm

Hello,

What I did was go through the steps in the official SAP help to set up the change document adapter for my custom business object i.e. attach the standard BOPf change document object as a delegated object.

This works fine, entries appear in CDHDR/CDPOS, but with the GUID. It is no big thing to get the proper ID i.e. a number a human can relate to based on the GUID but the SAP help says that you can redefine the method ion the callback class, and there is an example in the demo customer callback class, not that the demo customer business object actually works for change documents (does not have a change document delegated object).

In the /BOFU/CL_CHANGE_DOC_HANDLING nothing gets triggerd unless the flag MV_KEY_EXTID_MAP_REQUIRED is set to ABAP_TRUE. this gets set in the CONSTRUCTOR if MV_KEY_TO_EXTID_MAP_STRUCT is not initial.

This value in turn is created during ,ethod INIT_CDO_NODES_AND_TABS. this method looks at the entries in TCDOB for the object class. I think TCDOB is the table where the entries in transaction SCDO are stored.

The method loops through the entries in TCDOB looking for an entry which NOT a transparent table e.g. a structure, in fact i think it is looking for /BOFU/CDTEXTID specifically. Then that structure gets analysed and the program decides if the first include is /BOFU/S_CD_EXTID_KEYS. if this is true, then the MV_KEY_TO_EXTID_MAP_STRUCT variable is filled.

I will have a go at playing with this, probably adding /BOFU/CDTEXTID to SCdO for my custom object and then do some more debugging.

There also does not seem to be any standard way to get the CDHDR/CDPOS entries back outside of the WDA/FPM environment. What if you wanted to use this with UI5 or whatever comes next after that?

Cheersy Cheers

Paul

Like 0 | Share

Paul Hardy
January 8, 2015 at 9:08 pm

Yes, that worked fine.

Now i get two entries in CDPOS, both with the GUID as the OBJECTID, but one of them has /BOFU/CDTEXTID as the "table" being changed and that entry contains the external ID.

I am really happy bunny about getting this working, if the SAP help had just told me waht to do it would not have been anywhere near as much fun!

Cheersy Cheers

Paul

Like 0 | Share

James Wood | Blog Post Author
January 8, 2015 at 9:17 pm

Nicely done and thanks for sharing. This will be a nice little tip to keep in everyone's back pocket going forward.

P.S. I find that SAP provides us with plenty of opportunities to have the kind of fun you're describing. Nothing else is quite like it. 🙂

Like 0 | Share

Sagar Prusty
September 12, 2019 at 7:36 am

Hi James.

It was a lovely experience looking into these.
I am very new to BOPF and successfully had an overview Idea.

Thanks for sharing

Like 0 | Share

Find us on

Privacy	Terms of Use

Legal Disclosure	Copyright
Trademark	Cookie Preferences
Newsletter	Support