Abyson Joseph

September 26, 2013 | 6 minute read

Creation of Custom Kernel BADI and Calling it in a Custom Program

Author: Abyson Joseph Chavara

Created on : 26.09.2013

The BADI(Business Add-in's) is an object-oriented enhancement option. The BADI defines an interface that can be implemented by BADI-implementations that are transport objects of their own. The new BADI or Kernel BADI is fully integrated into the Enhancement Framework. Within the Enhancement Framework a BADI is an enhancement option or an anchor point for an object plug-in.

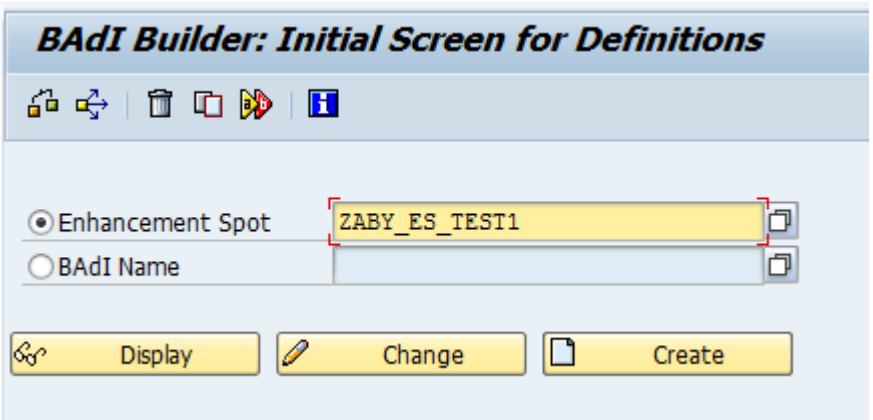
Kernel BAdI's Features

- Are integrated directly in the ABAP Language/Runtime
- Improved filter support allows non-character filter types and complex filter conditions
- Enable reusable implementation instances (Stateful BAdI)
- Control of the lifetime of implementations (BAdI-context)
- Allow for inheritance of implementations
- Can be switched by the Switch Framework

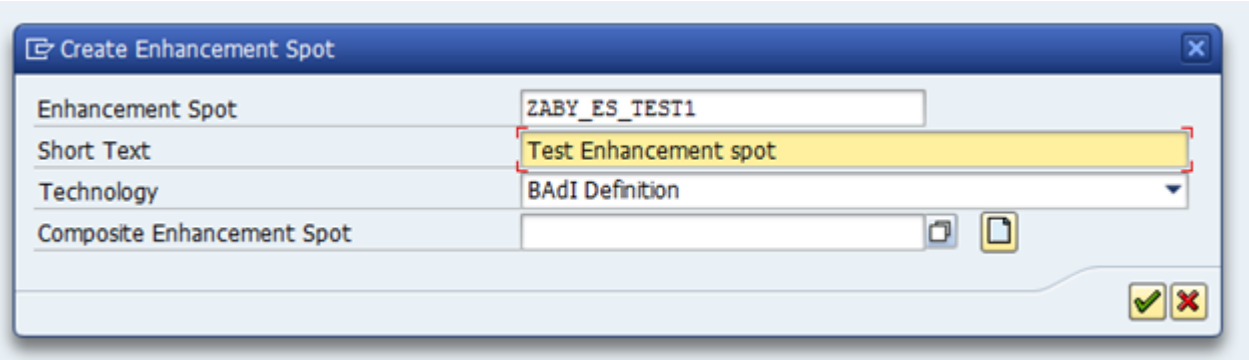
In this document, I have demonstrated the various steps for creating a Kernel BADI and calling it in our own Custom program.

1. First create a Enhancement spot from SE18. Kernel Badi's belong to an enhancement spot.

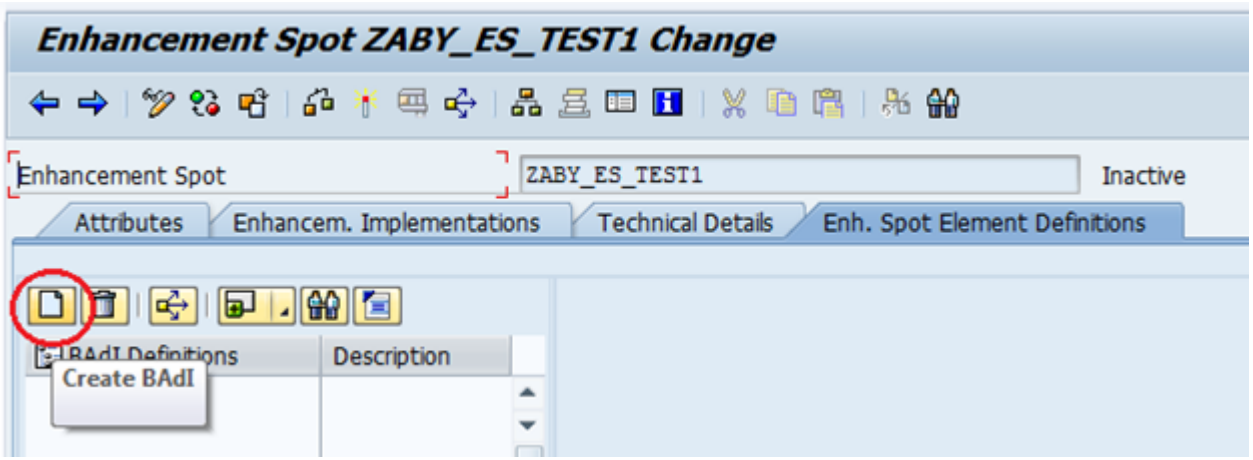
   Enhancement spots carry information about the positions at which enhancement options were created. One enhancement spot can manage several enhancement options of a Repository object. Conversely, several enhancement spots can be assigned to one enhancement option.



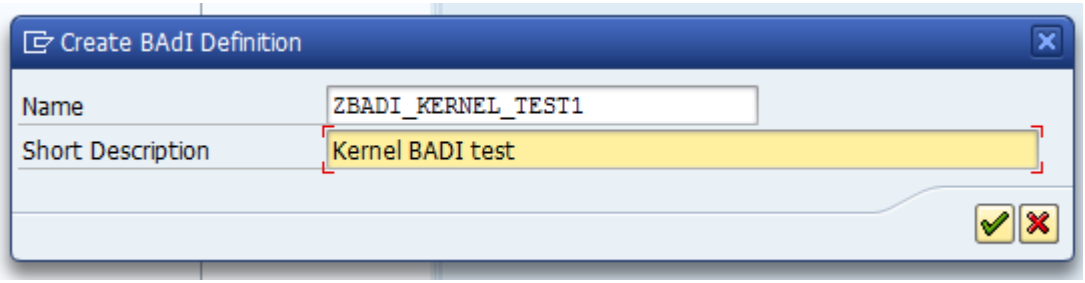
   Enter the description and if you want you can assign the new enhancement spot to a composite enhancement spot. Composite enhancement spots are used for the semantic grouping of simple enhancement spots. A composite enhancement spot contains either one or more simple enhancement spots and/or one or more composite enhancement spots of the relevant type. You can use composite enhancement spots to combine simple enhancement spots into meaningful units.



2. On creating the Enhancement spot, you will be directed to Enhancement Spot Editor. Now we need to create a new BADI Definition.



3. Click on the create BADI button.

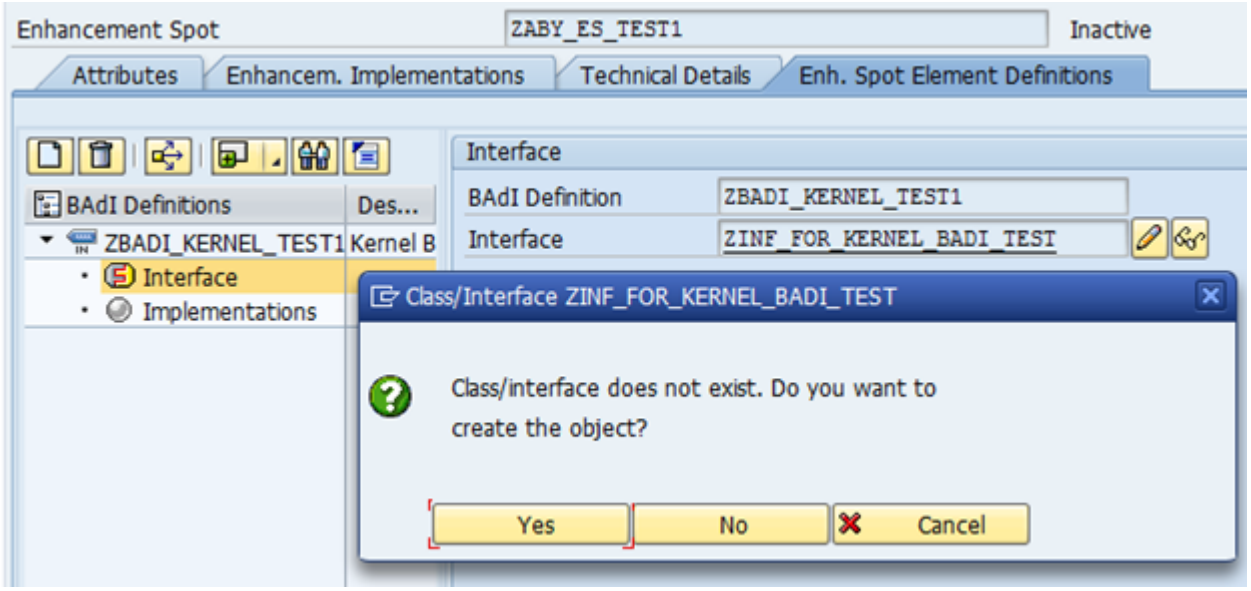


4. You will find certain options for the BADI definitions as below.

Usability – Multiple use – that is, there can be several active implementations Limited Filter Use – This makes the BADI Filter-dependent – that is, you apply a filter value to each method called (for example, a country). A different (active) implementation is then called for each value. Possible filter values are characterized by the filter type.
Instance Generation Mode – This property controls the instantiation of the object plug-ins during execution of the statement GET BADI. The first two specifications define context-free BAdIs. **Newly Created Instantiation** – New object plug-ins are created at each execution of the statement GET BADI. Reusing Instantiation – An object plug-in that was used once in the current internal mode is reused, if it is required more than once. **Context-Dependent Instantiation** – A context must be specified for GET BADI. This context controls the instantiation. Only one object plug-in is created for each context and implementing class. Each time there is another GET BADI with the same context, it is reused. A context is an instance of a class that implements the tag interface if_badi_context. The specification takes place in the form of a reference to such an instance.

Fallback Class – Fallback class for a BADI is used if there is no active BADI implementation. This means: The GET BADI command returns a handle to an instance of the fallback class and the respective CALL BADI calls the methods of the fallback class instance. As soon as there is an active BADI implementation, the fallback class is not used any longer at runtime.

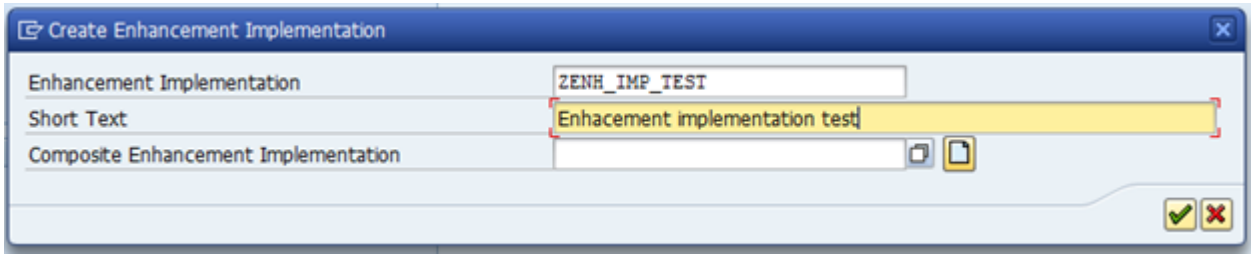
5. Now we need to add an interface to the BADI. Expand the Node of BADI definition name and double click on node Interface. You can either add existing interface or will be prompted to create.



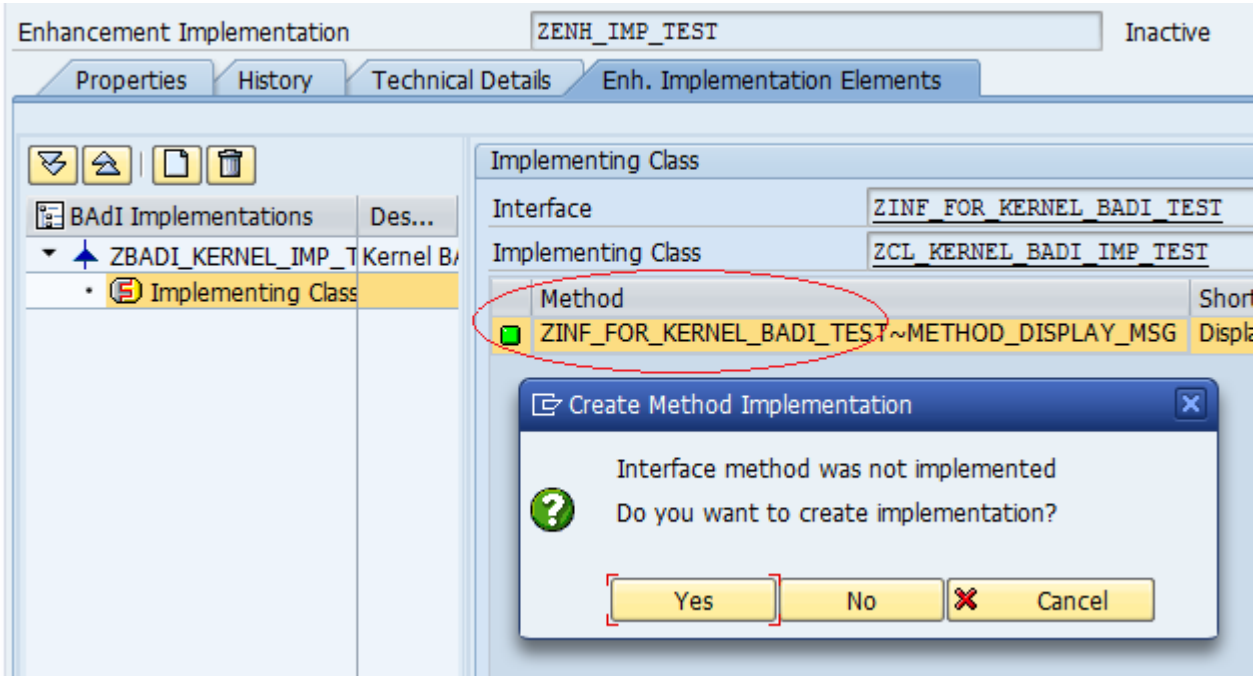
Save, activate and comeback to the BADI definition screen and activate the Enhancement Spot.

7. Next we need to implement the Enhancement spot and BADI. Right click on the BADI definition and select Create BADI Implementation.

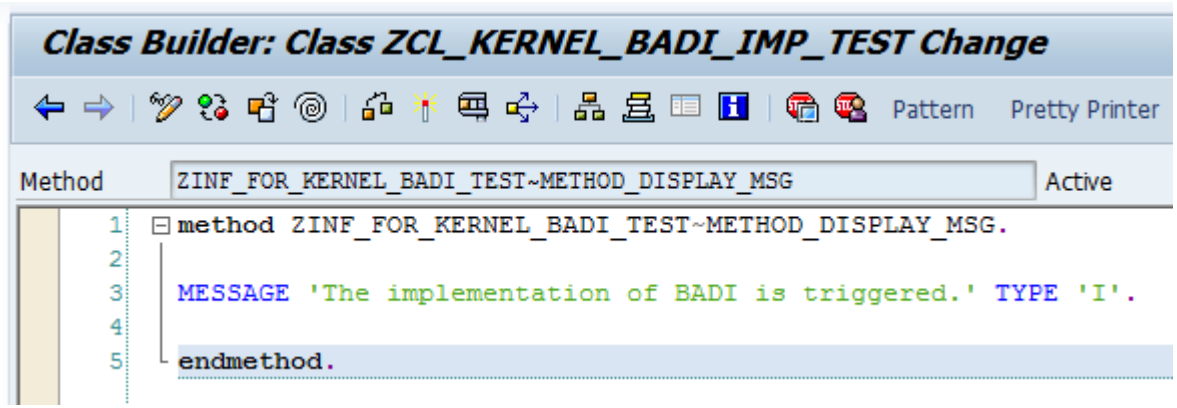
8. First the system will ask for enhancement implementation name. Please enter a name and description.



11. You will be directed to the enhancement implementation screen, shown below. Double click on the Interface method and you will be prompted to create implementation for the method.



12. On clicking yes, you will be navigated to editor for the method. Add the following code for showing a message or anything as per your requirement. Save it and activate.

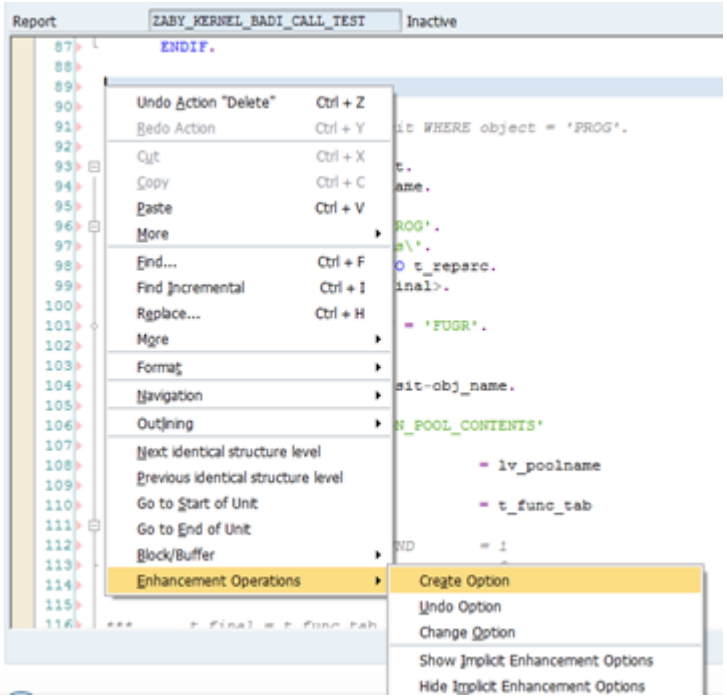


So the BADI definition and implementation part is complete.

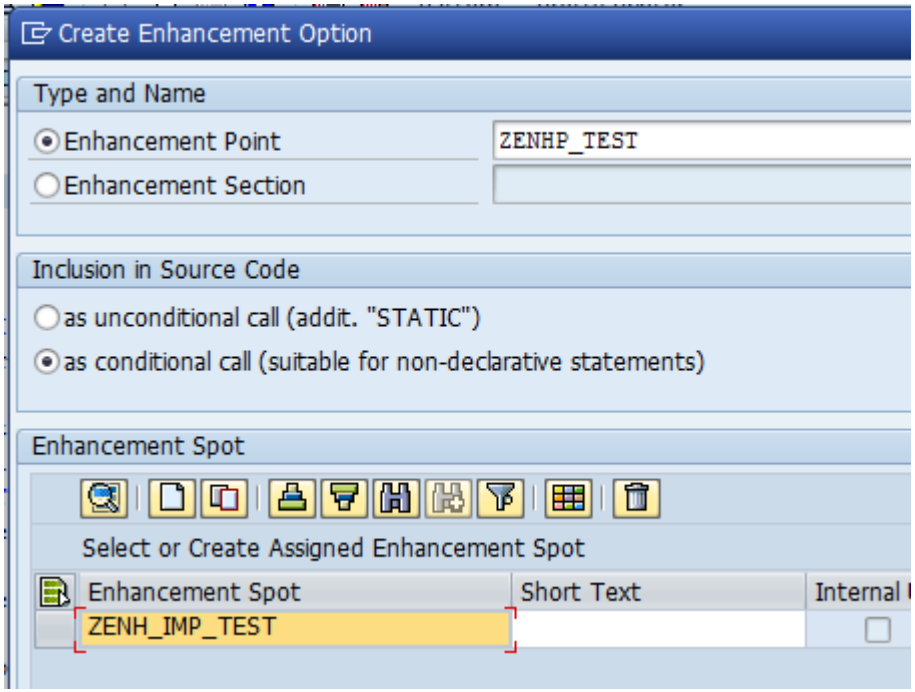
Next we will see how we can call this Kernel BADI in our Custom program.

13. Create or open an existing program in SE38. Point to the line where you need to add enhancement option. Right click and select Create Option.

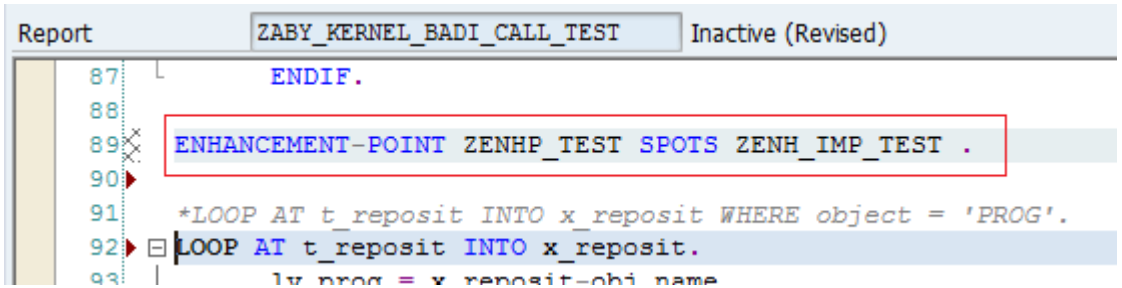
(Please note, to call a Kernel BADI in a Custom program we don't need any enhancement options, as we can just write the code anywhere. Just meant to show reader the creation of enh option and a point in a program. While creating an option either you could use an existing enh spot imp or can create a new one there. Rather than explaining the topic in an atmosphere of real business scenario, I focused on giving the reader an understanding in the concepts.)



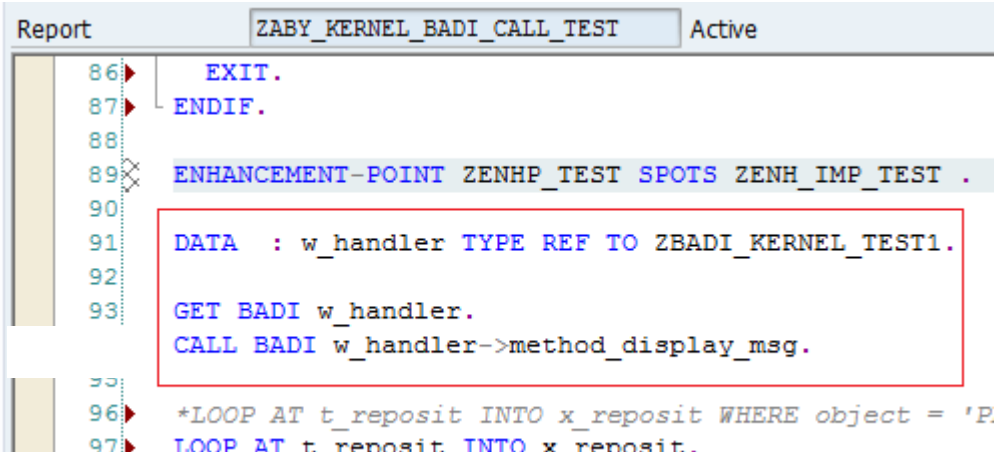
14. Give an Enhancement point or section name and Enhancement spot implementation name. Here I have opted an Enhancement point and have entered Enhancement spot implementation name as ‘ZENH_IMP_TEST’ (which we have created before).



A new line will be added to the existing code as shown below.



15. Then add the below code which is shown in the red box below.



Alert Moderator

Assigned tags

ABAP Development

abap

enhancement spot

kernel badi

Apart from Classic BADI's which are been called by Proxy class cl_exithandler, Kernel BADI's are called directly with the reference to the BADI definition via GET BADI and CALL BADI statements. That is one of the reasons why Kernel BADI is faster than classic BADI. Also in Classic BADI, while we call it via cl_exithandler, we use the reference to the interface rather than BADI definition

16. Now activate the program and execute it. When the cursor reaches the enhancement point, where the BADI is called it

Similar Blog Posts

Badi_Sorter with Kernel Badi and other sorting options

By Former Member Oct 02, 2016

If the program fails to trigger the BADI implementation, please recheck whether everything associated with it is 'Activated' after the creation.

CCLM: How to create own attribute and implement BADI to extract value

By Extra Li Mar 09, 2015

Thank You.

Create Custom Tab at Header in Purchase Requistion

By Siva rama Krishna Pabbraju Jun 05, 2014

Related Questions

BAPI Check in BADI Implementation

By Former Member Feb 17, 2009

No Implementation for BADI MRM_RELEASE_CHECK in ECC6

By Former Member Jul 15, 2011

how to link a custom BADI with standard report.

By Former Member Sep 17, 2008

17 Comments

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

Former Member

February 22, 2014 at 6:44 am

Very informative blog. Just one question though - I didn't understand why did you assign an enhancement spot implementation to the enhancement point in step 14. Enhancement points can be used to plug-in source code at the position in the future state and directly assigning an implementation doesn't make much sense.

> *14. Give an Enhancement point or section name and Enhancement spot implementation name. Here I have opted an Enhancement point and have entered Enhancement spot implementation name as 'ZENH_IMP_TEST' (which we have created before).*

Like 0 | Share

Abyson Joseph | Blog Post Author
February 23, 2014 at 9:33 am

To call a Kernel BADI in a custom program we don't need any enhancement options, as we can just write the code anywhere. Just meant to show reader the creation of enh option and a point in a program. While creating an option either you could use an existing enh spot imp or can create a new one there. Rather than explaining the topic in an atmosphere of real business scenario, I focused on giving the reader an understanding in the concepts.

Like 0 | Share

abilash n
April 24, 2014 at 11:15 am

Hi Abyson,

can you please tell me what is the diff between kernel badi and normal badi implementation. As you have implemented in normal way.

I know that kernel badi is faster than normal classical badi. but i never understood the kernel badi. if you dont mind can you please reply to this or create new blog on kernel badi,,,,

Like 0 | Share

Abyson Joseph | Blog Post Author
April 24, 2014 at 11:35 am

Hi Abilash,

Hope you will find the answer by checking my document Is Kernel BADI 'really' faster than Classic BADI ?

Thanks and Regards,

Abyson Joseph

Like 0 | Share

abilash n

April 26, 2014 at 4:07 am

Thanks Abyson for sharing the blog. Now i got some good idea on it. If possible please share some more info on the kernel and classic badi.

Another question - As there are limitations in customer exit so sap came up with badi's previously. And what made sap to move to new badis even after classic badis(apart from fast).

Thanks for your feedback.

Like 0 | Share

Former Member

April 26, 2014 at 9:53 am

Hi Abyson,

Good document with proper explanation. □

Like 0 | Share

Former Member

September 28, 2014 at 10:43 pm

Hi All,

I may sound too layman,but i wanted to know. why a custome program needs a enhancement spot?

can we not implement directly in the custom program.for me this seems a overdo.

Experts,please clarify my doubts.

Thanks & Regards,

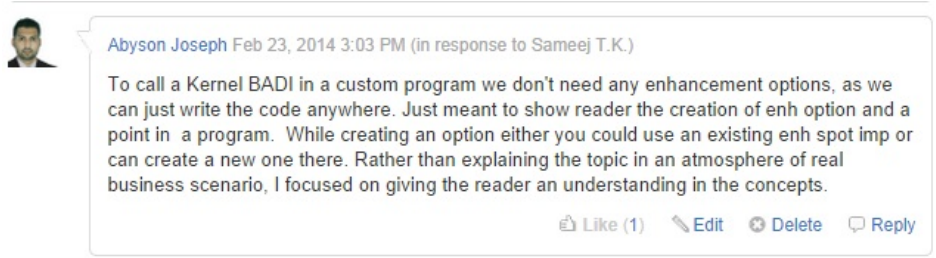
Lakshmi

Like 0 | Share

Abyson Joseph | Blog Post Author
October 6, 2014 at 1:45 pm

Hi Sri Lakshmi Naidu

Your doubt doesn't sound too layman. We don't need an enhancement spot in custom program. But I have explained it already above, why i have added the spot in the demo custom program.



Thanks and Regards,

Like 0 | Share

Matthew Billingham
October 6, 2014 at 2:41 pm

Custom BADIs are useful in CMOD and other old-style user exits, to keep implementations separate, and to use the new technology in "old" places. They're also useful if you are, for example, building a template system that is rolled out globally, but allows local enhancements.

Like 0 | Share

Abyson Joseph | Blog Post Author
October 6, 2014 at 3:01 pm

Thanks Matthew for sharing the valuable information..

Like 0 | Share

Former Member
October 7, 2014 at 4:03 am

Thanks Abyson Joseph.

Like 0 | Share

Former Member
April 25, 2015 at 7:50 am

Thanks Abyson Joseph

Like 0 | Share

Former Member
May 27, 2015 at 5:17 am

Excellent explanation Abyson. Thanks a lot.

Like 0 | Share

Former Member
August 4, 2017 at 9:47 am

Hi Joseph,

Still I have a question how to connect this kernel badi in the standard code and still we are safe during the upgrade process.

Thanks,

Amar.

Like 0 | Share

Divi Srilaxmi
November 1, 2017 at 6:50 am

Hi Joseph,

Thank you so much for explaning about Kernal Badi. □

I just wanted to know..How we will find out this enhancement spots names which are there for standard Tcodes??

As of know I only know the procedure to get classic Badi by using CL_EXITHANDLER.

Please let me know the procedure to find the kernal badi's.

Thanks!

Srilaxmi

Like 0 | Share

Frank Walter
December 7, 2017 at 5:51 pm

Very good description

Like 0 | Share

Pandiri Balaraju
June 12, 2019 at 10:11 am

Thank you for nice information on Kernal BADI

Like 1 | Share

Find us on

Privacy	Terms of Use
Legal Disclosure	Copyright
Trademark	Cookie Preferences
Newsletter	Support