# [Содержание](#Содержание)

# [Object Oriented ABAP - Triggering And Handling Events](#Triggering_And_Handling_Events)

[**Event Based Process Chains**](https://sapbwblog.wordpress.com/2013/09/05/event-based-process-chains/) **– 05.09.2013**

<https://sapbwblog.wordpress.com/2013/09/05/event-based-process-chains/>

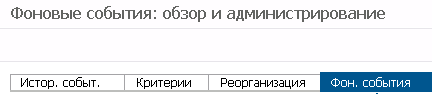
There may be a requirement where you may need to schedule a process chain after an event is being triggered.

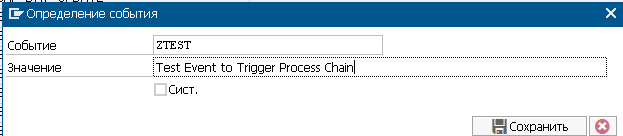
In this Example I have a process *chain A* runs weekly, when completed should trigger an event which will result in execution of Process *chain B*.

Steps to be followed is as follows

Step 1 - Create an Event

Go to Transaction code SM62   –> Select  *BckProcEvnts* [Фон. события] tab  –> Create an event.





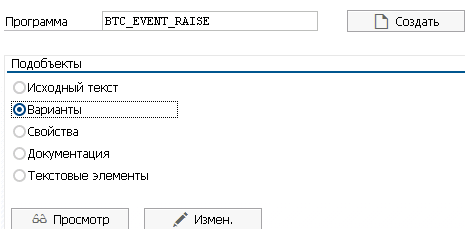
In the snapshot  above  we can see I have created an Event ‘ZTEST’.

*zbi\_t*

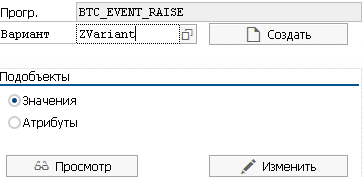
*Incomplete data compared to the previous similar day of the week*

Step 2 - Create variant for Program BTC\_EVENT\_RAISE

Go to se38 –> Enter Program name BTC\_EVENT\_RAISE -> Select Variant radio Button -> Click *Display*

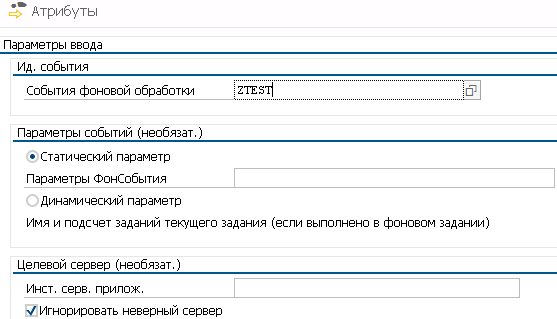


Enter a name for Variant and click *Create*.

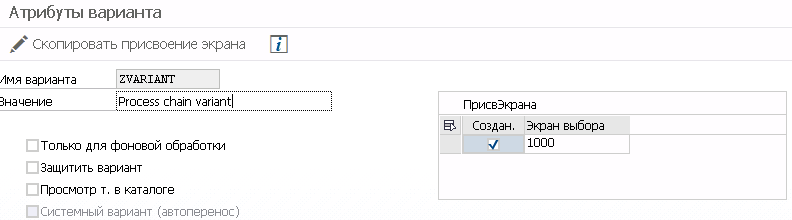


*ZVariant\_bi\_t*

In the following screen - enter the *event name* which you have created in SM62 (in Step 1). In my case I had created *ZTEST* Event.



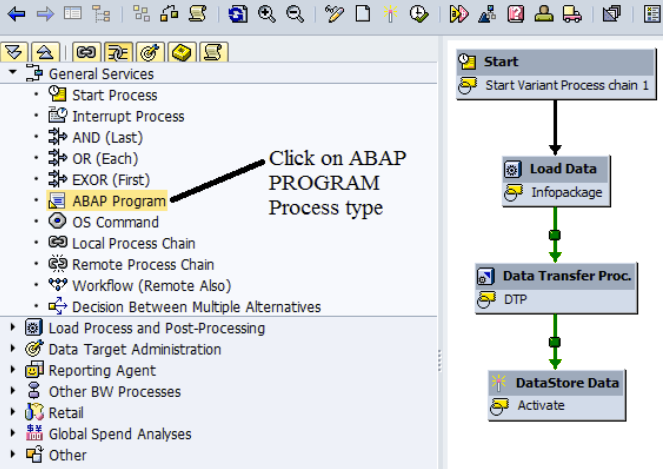
Click on *Attributes* , enter description for variant  and click on *Save*



*Incomplete data*

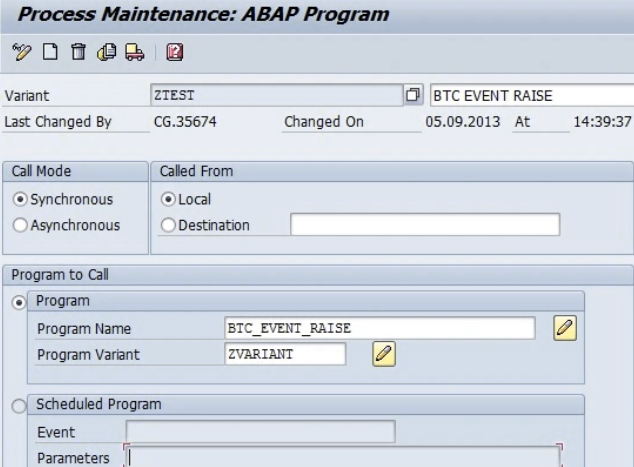
Step 3 - Go to Process Chain A

At the end of your Process chain, insert process type *ABAP Program*.

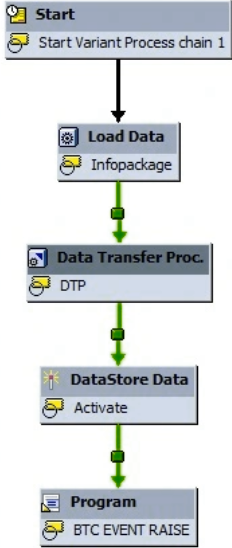


In the next Screen insert  the ABAP Program *BTC\_EVENT\_RAISE*.

Enter the variant created in Step 2.



Your Process chain A will look like this

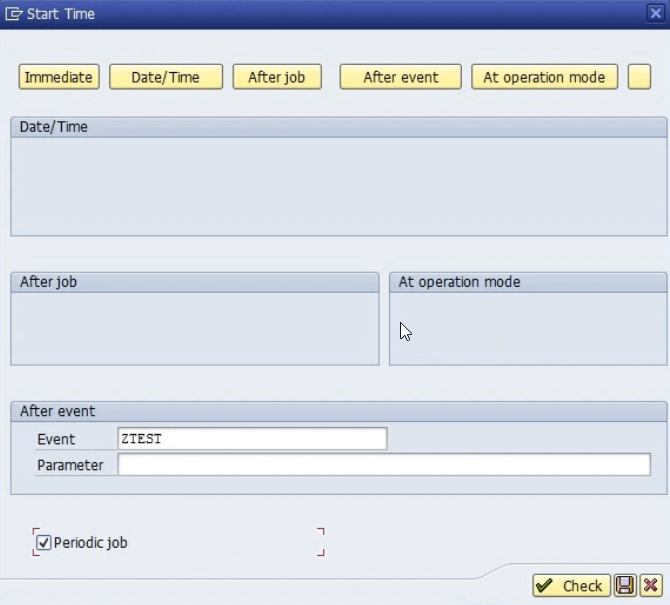


Schedule process chain A as per your requirement. Example I scheduled it weekly [on Sunday, Morning 6 AM].

Step 4 - Go to your RSPC and open Process chain B

Go to Start Variant –> Change selections –> After Event –> Enter the Event Name created in Step 1.

Check box *Periodic job*



<https://sapspaces.com/how-to-create-rap-business-events-in-sap-btp-abap-environment/>

## **An Introduction to Enterprise Event Enablement for SAP BTP ABAP Environment**

<https://community.sap.com/t5/technology-blogs-by-sap/an-introduction-to-enterprise-event-enablement-for-sap-btp-abap-environment/ba-p/13521831>

## **Consume an Event using Event Consumption Model**

<https://community.sap.com/t5/technology-blogs-by-sap/consume-an-event-using-event-consumption-model/ba-p/13537871>

# Business Event Consumption

<https://help.sap.com/docs/ABAP_PLATFORM_NEW/fc4c71aa50014fd1b43721701471913d/e3446499d68f40db9013b0ad4580e2fe.html>

# Object Oriented ABAP - Triggering And Handling Events

# <https://discoveringabap.com/2021/09/27/object-oriented-abap-3-triggering-and-handling-events/>

Date: [September 27, 2021](https://discoveringabap.com/2021/09/27/object-oriented-abap-3-triggering-and-handling-events/)Author: [Jagdish](https://discoveringabap.com/author/jpsapabap21/)

# [Содержание](#Содержание)

In this post, you will learn about events in ABAP classes.

Event is an action or occurrence of something that is recognized by the application. Handling events means response by the application to the action or that occurrence.

As an example, when ALV is displayed and user clicks on something, event is triggered by the user’s action and the event handler simply responds to the user action i.e. handles the event of the user action.

***Event Definition***

Similar to methods, events can be instance or static. We will see the instance ones. For static *CLASS-EVENTS* key word is used. *Events can have only exporting parameters.*

EVENTS *something\_happened* EXPORTING VALUE(about\_event) TYPE string.

***Event Handler***

Event handler can be in the same class of in a different class. These are nothing but methods. In the event handler, we specify the event name which we are going to handle and the class name where it is defined.

METHODS *handle\_something* FOR EVENT *something\_happened* OF *zjp\_abap*

IMPORTING *about\_event*.

***Trigger Event***

The event can be triggered from a method in the same class. The details about the events or parameters that we need to pass to handler method as passed using EXPORTING.

RAISE EVENT *something\_happened* EXPORTING *about\_event* = 'Something'.

Note that the which event handler needs to be triggered for the event raised is determined at the runtime. Hence, we need to specify this information at runtime by registering the event handlers.

***Register Event Handler***

This is pretty much self explanatory.

SET HANDLER *lo\_abap\_new->handle\_something* FOR *lo\_abap\_new*.

Now, you should be able to understand the events and event handlers and how the control will flow from the statement RAISE EVENT to the EVENT HANDLER method.

Going back to the ALV example, the events are already defined in the the ALV class CL\_GUI\_ALV\_GRID and are also triggered from appropriate actions. All we need to do is define event handler methods in the class and register these event handlers.

WHERE (YYYYMM = CAST(LEFT(REPLACE(CAST(DATEADD(month, - 1, GETDATE()) AS date), '-', ''), 6) AS int))

## **Leveraging SAP Event Trigger for Event-Driven Architecture – Part 2**

<https://blog.kawanlama.tech/2023/12/11/leveraging-sap-event-trigger-for-event-driven-architecture-part-2/>

<https://blog.kawanlama.tech/2023/11/22/leveraging-sap-event-trigger-for-event-driven-architecture-part-1/>

# BI\_EVENT\_HANDLER SAP Interface - Event handler

<https://www.se80.co.uk/sap-oop-pre-ecc/b/bi_e/bi_event_handler.htm>

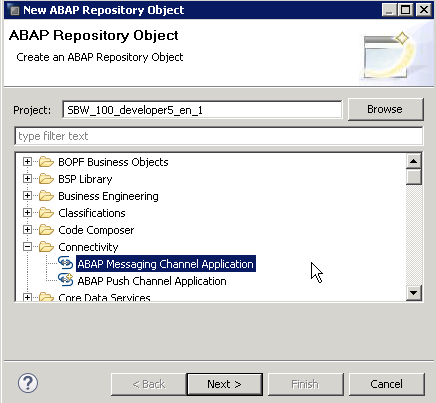
# Trigger/Handle events between programs in different ABAP sessions

<https://stackoverflow.com/questions/48442723/trigger-handle-events-between-programs-in-different-abap-sessions>

− I have two programs running in separated sessions. I want to send a event from *program A* and catch this event in *program B*. How can I do that ?

− There is a mechanism that you can use to send messages between sessions - [ABAP Messaging Channels](https://help.sap.com/doc/abapdocu_751_index_htm/7.51/en-US/abenamc.htm). You can send anything that is either a text string, a byte string or can be serialised in any of the above.

You will need to create such a message channel using the ***repository browser SE80*** [*Create -> Connectivity -> ABAP Messaging Channel*] or with the Eclipse ADT *[New -> ABAP Messaging Channel Application]*.



In there, you will have to define

* The message type [text vs binary].
* The ABAP programs that are authorised to access the message channel.
* The scope of the messages [i.e. do you want to send messages between users? or just for the same user? what about between application servers?]

The message channels work through a *publish - subscribe mechanism*. You will have to use specialised classes to publish to the channel [inside report A] and to read from the channel [inside report B]. In order to wait for a message to arrive once you have subscribed, you can use the statement [WAIT FOR MESSAGE CHANNELS](https://help.sap.com/doc/abapdocu_750_index_htm/7.50/en-US/abapwait_amc.htm).

Example code

*" publishing a message*

CAST *if\_amc\_message\_producer\_text*(

cl\_amc\_channel\_manager=>*create\_message\_producer*(

i\_application\_id = 'DEMO\_AMC'

i\_channel\_id = '/demo\_text'

i\_suppress\_echo = abap\_true )

)->*send*( i\_message = text\_message ).

*" subscribing to a channel*

DATA(lo\_receiver) = NEW *message\_receiver*( ).

cl\_amc\_channel\_manager=>create\_message\_consumer(

i\_application\_id = 'DEMO\_AMC'

i\_channel\_id = '/demo\_text'

)->start\_message\_delivery( i\_receiver = lo\_receiver )

" waiting for a message

WAIT FOR MESSAGING CHANNELS

UNTIL lo\_receiver->text\_message IS NOT INITIAL

UP TO time SECONDS.

If you want to avoid waiting inside your subscriber report B and to do something else in the meanwhile, then you can wrap the WAIT FOR... statement inside a RFC and call this RFC using the [aRFC variant](https://help.sap.com/doc/abapdocu_750_index_htm/7.50/en-US/abapcall_function_starting.htm). This would allow you to continue doing stuff inside report B while waiting for an event to happen. When this event happens, the aRFC callback method that you defined inside your report when calling the RFC would be executed.

Inside the RFC, you would simply have the subscription part and the WAIT statement plus an assignment of the message itself to an EXPORTING parameter. In your report, you could have something like:

CALL FUNCTION 'ZMY\_AMC\_WRAPPER' STARTING NEW TASK 'MY\_TASK'

CALLING lo\_listener->my\_method ON END OF TASK.

" inside your 'listener' class implementation

METHOD my\_method.

DATA lv\_message TYPE my\_message\_type.

RECEIVE RESULTS FROM FUNCTION 'ZMY\_AMC\_WRAPPER'

IMPORTING ev\_message = lv\_message.

" do something with the lv\_message

ENDMETHOD.

### **ABAP Channels - ABAP Messaging and Push Channels**

<https://blog.maruskin.eu/2015/12/abap-channels-abap-messaging-and-push.html>

In [SAP Application Server ABAP 7.40](http://blog.maruskin.eu/2015/01/sap-netweaver-74.html) Support Package 05 a new concept is introduced via infrastructure for event based communication. The new concept is called ***ABAP channels*** created in SAP NetWeaver Application Server ABAP.

SAP Web UIs such as [WebGUI](http://blog.maruskin.eu/2015/04/few-notes-on-sap-screen-personas-30.html), [Business Server Pages](http://blog.maruskin.eu/search/label/BSP) - BSP, [Web Dynpro ABAP](http://blog.maruskin.eu/2008/03/wd4a-webdynpro-for-abap.html) - WD4A or [SAPUI5](http://blog.maruskin.eu/2013/09/sap-ui5.html) can use the *ABAP Push Channel* - ***APC*** which enables such a real-time communication via WebSockets [over TCP/IP socket]. There is a possibility of *push notifications* to UI as soon as data change happened in the ABAP backend. This is a sign for a user to request the changed data from ABAP backend and update UI.

Traditionally there would be a many *polling* performed to inquire the status of the data processing. The *ABAP Messaging Channel* - ***AMC*** infrastructure *replaces the traditional polling*. It acts as a broker for the messages exchange between different ABAP sessions [on different ABAP application servers]. The AMC basically provides real-time communication between ABAP sessions.

With support of *push channels*, *messaging channels* and *collaboration scenarios* it is possible to use WebSockets to facilitate the exchange of messages between different ABAP sessions and enable users to collaborate over the infrastructure in real time.

Finally how is this done technically? It is implemented by *Push Channel Protocol* - ***PCP***. It is SAP proprietary message based protocol. It is very similar to HTTP protocol. The PCP messages consist of header fields and a plain body. The body is separated by line feeds. Field values entries are in pairs of name-value separated by “:” sign. Implementation in the ABAP backend comprises of class ***CL\_AC\_MESSAGE\_TYPE\_PCP*** and interface ***IF\_AC\_MESSAGE\_TYPE\_PCP PCP*** which is having API. Implementation in [SAPUI5](http://blog.maruskin.eu/2013/09/sap-ui5.html) is within the class ***sap.ui.core.ws.SapPcpWebSocket***. For other UI technologies there is a library ***sap-pcp-websocket.js*** for particular PCP methods.

More information

[Say goodbye to polling real time eventing in abap with abap channels](http://scn.sap.com/community/abap/blog/2014/11/24/say-goodbye-to-polling-real-time-eventing-in-abap-with-abap-channels)

[ABAP Push Channel (APC)](https://help.sap.com/saphelp_nw74/helpdata/en/99/911508963d452c80d9def3dd864f4f/content.htm)

[Introduction to abap channels](http://scn.sap.com/community/abap/connectivity/blog/2014/11/27/introduction-to-abap-channels)

[ABAP Channels Part 1: WebSocket Communication Using ABAP Push Channels](http://scn.sap.com/community/abap/connectivity/blog/2013/11/18/websocket-communication-using-abap-push-channels)

[ABAP Channels Part 2: Publish/Subscribe Messaging Using ABAP Messaging Channels](http://scn.sap.com/community/abap/connectivity/blog/2014/03/26/abap-channels-part-2-publish-subscribe-messaging-using-abap-messaging-channels)

[ABAP Channels Part 3: Collaboration Scenario Using ABAP Messaging and ABAP Push Channels](http://scn.sap.com/community/abap/connectivity/blog/2015/07/27/specification-of-the-push-channel-protocol-pcp)

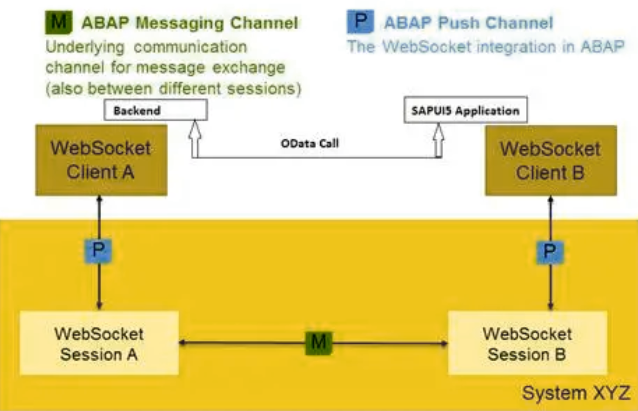
[Specification of the PCP](http://scn.sap.com/community/abap/connectivity/blog/2015/07/27/specification-of-the-push-channel-protocol-pcp)

# Push Notification in SAP – ABAP Push Channel, ABAP Messaging Channel in SAPUI5 – a Real Time Interaction

<https://www.erpqna.com/push-notification-in-sap-abap-push-channel-abap-messaging-channel-in-sapui5-a-real-time-interaction/>

In this article, we would learn how to create our first Real Time Client Server Communication using ABAP Push Channel - APC and [ABAP Messaging Channel - AMC](https://www.erpqna.com/sap-abap-7-5-certification/).

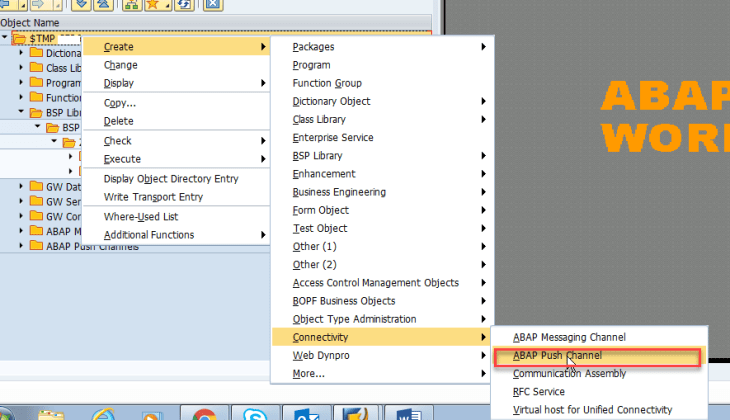
**The basic architecture is shown in this schema below**



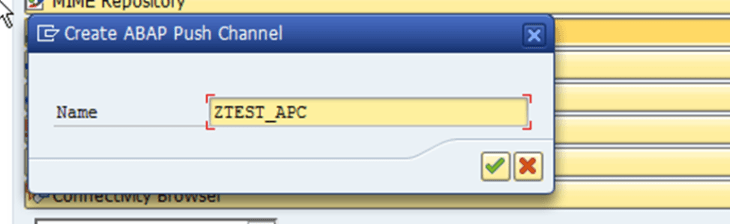
Follow the below steps to implement a real time interaction scenario

**Create *ABAP Push Channel***

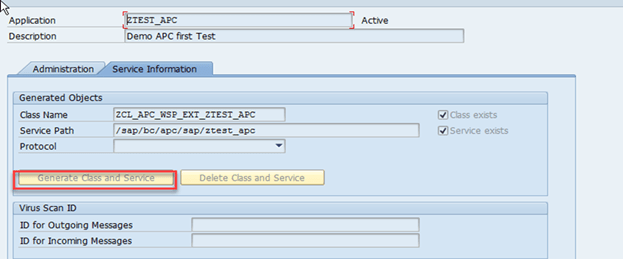
Go to SE80 transaction and *create ABAP Push Channel*



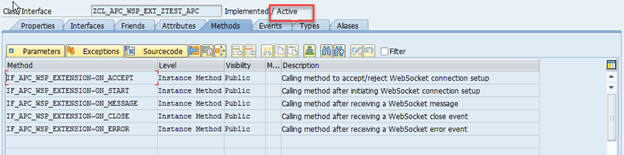
Give some name to your ABAP Push Channel *ZTEST\_APC*

 ​

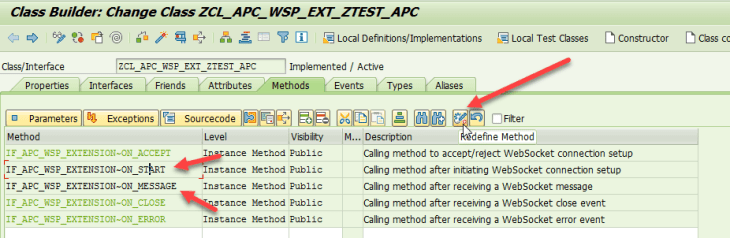
Click on *Generate Class and Service* button as mentioned below



Once the generation is successfully done, a new class *ZCL\_APC\_WSP\_EXT\_ZTEST\_APC* will be created.



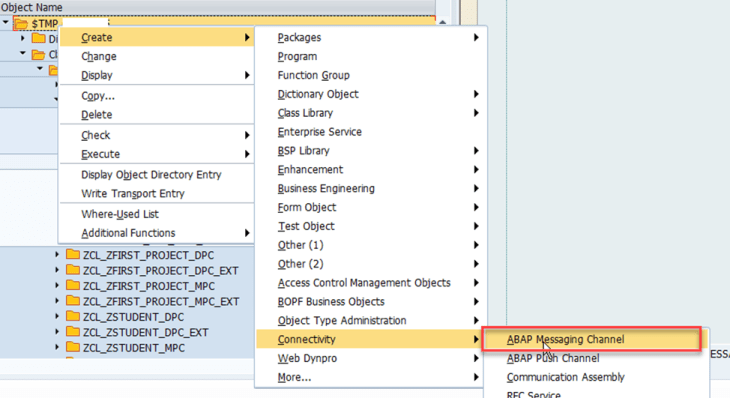
You need to re-define the ON\_START method​​ and ON\_MESSAGE method​​ of the generated class to activate it.​​ Hit the *Re-define* icon.​​ The code snippet for ON\_START is provided below.



The class should be​​ *Active*​​ now.

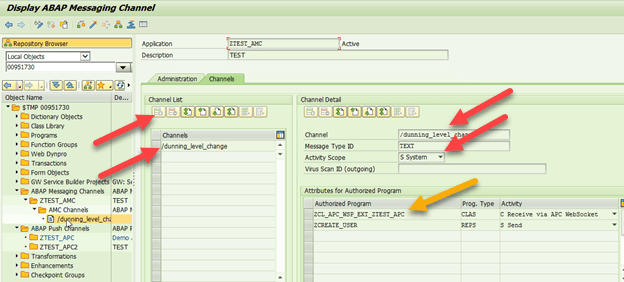
**Create AMC - ABAP Messaging Channel**

Go to SE80, *Create -> Connectivty -> ABAP Messaging Channel*



Add the following information

* Message Type Id [Example: /dunning\_level\_change]
* Activity Scope – System



Then add the following entry to list of Authorized Programs

* *Authorized Program* - ZCL\_APC\_WSP\_EXT\_ZTEST\_APC [Class was created in the previous step​​]​​

*Prog. Type* - CLAS​​

*Activity* -​​ Receive via APC WebSocket

* *Authorized Program* - ZCREATE\_USER [Must be created as a report]

*Prog. Type* - REPS

*​​Activity* - Send

Implement​​the IF\_APC\_WSP\_EXTENSION~*ON\_START* ​​ method in​​ ZCL\_APC\_WSP\_EXT\_ZTEST\_APC.

We have already informed you to re-define the ON\_START method. Now simply add the following code

method IF\_APC\_WSP\_EXTENSION~***ON\_START***.

DATA: *lo\_binding* TYPE REF TO if\_apc\_ws\_binding\_manager.

DATA: *lx\_error* TYPE REF TO cx\_apc\_error.

DATA: *lv\_message* TYPE string.

*\* bind the APC WebSocket connection to an AMC channel*

​​ TRY.

​​ lo\_binding = i\_context->*get\_binding\_manager*( ).

​​ lo\_binding->*bind\_amc\_message\_consumer*( i\_application\_id = ‘ZTEST\_AMC’

i\_channel\_id = ‘/dunning\_level\_change’ ).

​​ CATCH cx\_apc\_error INTO lx\_error.

​​ lv\_message = lx\_error->*get\_text*( ).

​​ ENDTRY.

endmethod.

In the above code snippet

*i\_application\_id* ​​ - is​​ the *name of the messaging channel* we’ve already created.

*i\_channel\_id* -​​ is​​ the *channel identifier*

Create the report​​ program - ZCREATE\_USER

ZCREATE\_USER - ​ is an ABAP Report used for​​ adding​​ users to the ZUSER table. Once the addition​​ is successfully done, we invoke the APC to notify the dunning level change.

REPORT ZCREATE\_USER.

tables *zuser*.

data *wa\_user* type zuser.

DATA:

​​ *lo\_producer* TYPE REF TO if\_amc\_message\_producer\_text,

​​ *lv\_message* TYPE string.

​​ *lv\_message* = ‘Dunning level has been changed. Reload the page to view the updated dunning levels’.

​​ SELECTION-SCREEN begin of block block1 with frame title text-001.

​​ ​​​​ PARAMETERS: p\_user type string.

​​ ​​​​ PARAMETERS: p\_first type string.

​​ ​​​​ PARAMETERS: p\_last type string.

SELECTION-SCREEN end of block block1.

START-OF-SELECTION.

wa\_user-FIRSTNAME = p\_first.

wa\_user-USERNAME = p\_user.

wa\_user-LASTNAME = p\_last.

INSERT zuser from wa\_user.

*\* Code to invoke the APC to notify the dunning level change*

​TRY.

​​ lo\_producer *?*= cl\_amc\_channel\_manager=>*create\_message\_producer*(

​​i\_application\_id = ‘ZTEST\_AMC’

i\_channel\_id = ‘/dunning\_level\_change’ ).

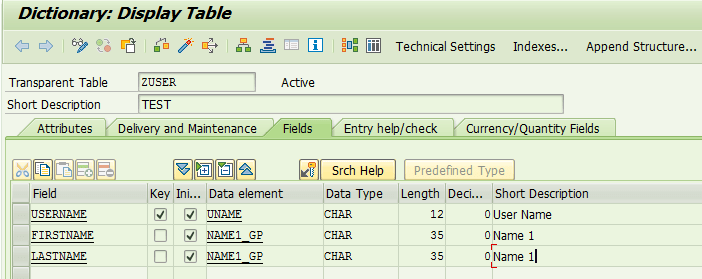
​​lo\_producer->send( i\_message = p\_first ).

​​CATCH cx\_amc\_error INTO DATA(lx\_amc\_error).

​​RETURN.

​​ENDTRY.

***ZUSER***​​ is a simple table where we​​ have our data.



* ***CREATE\_MESSAGE\_PRODUCER***​​ will be triggered once a new entry is added to the table with two parameters *i\_application\_id*, *i\_channel\_id*.
* ***SEND***​​ method will send the customizing message to the frontend part.

We created a simple​​ SAPUI5 application​​ with worklist with list of users stored in the​​ ZUSER​​ database table​​

In the​​ ***onInit***​​ method in the worklist controller we add the​​ ***initDunningRunFeed***​​ method to open the connection using the​​ WebSocket​​ between SAPUI5 Application and APC which will raise the​​ IF\_APC\_WSP\_EXTENSION~ON\_START​​ method

***initDunningRunFeed***: function() {

var *that* = this ;

*varhostLocation* = window.location,​​ ​​​​ socket, socketHostURI, webSocketURI;

​​ ​​ ​​​​ if (hostLocation.*protocol* === “https:”) { socketHostURI = “wss:”;​​​​ } else { socketHostURI = “ws:”;​​​​ }

socketHostURI += “//” + hostLocation.*host*;

webSocketURI = socketHostURI + ‘/sap/bc/apc/sap/ztest\_apc’ ;

​​ ​​ ​​ ​​​​// varoSocket = this.oSocket = new sap.ui.core.ws.SapPcpWebSocket(webSocketURI,

​​ ​​ ​​ ​​// sap.ui.core.ws.SapPcpWebSocket.SUPPORTED\_PROTOCOLS.v10);

​​ ​​ ​​ ​​socket = new *WebSocket*(webSocketURI);

socket.*onopen* = *function*() {};

socket.*onmessage* = function(*dunningRunFeed*) {

​​ ​​ ​​ ​​ ​​ ​​ ​​ if (dunningRunFeed.*data* !== undefined) {

jQuery.sap.*require*(“sap.m.MessageBox”);

sap.m.InstanceManager.*closeAllDialogs*();

that.getOwnerComponent().getModel().*refresh*();

sap.m.MessageBox.show(

​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​​​ ‘You ve added a new user : ​​ ‘+ dunningRunFeed.data,

sap.m.MessageBox.Icon.INFORMATION,

​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​​​ “APC Notification”, [sap.m.MessageBox.Action.OK]

​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​​​ );

​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​ ​​​​ }

​​ ​​ ​​ ​​ ​​ ​​ ​​​​ };

socket.onclose = function() {};

​​ ​​ ​​​​ },

If a new entry is added to the​​ ZUSER​​ data base table,​​ the ​​ socket.onmessage​​ callback will be triggered. Inside this callback we perform a new refresh to the model to make the list updated.​​

We hope, you know how to create a simple OData Service and redefine the method USERSET\_GET\_ENTITYSET.

<code>method USERSET\_GET\_ENTITYSET.

data ls\_user type zuser.

data lt\_user type TABLE OF zuser.

select \* from ZUSER into CORRESPONDING FIELDS OF TABLE LT\_USER.

MOVE-CORRESPONDING LT\_USER TO ET\_ENTITYSET.

endmethod.</code>

## **ABAP Objects Events – Raising and Handling**

<https://zevolving.com/2013/09/abap-objects-events-raising-and-handling/>

CLASS **lcl\_main** DEFINITION.

PUBLIC SECTION.

DATA: *v\_num* TYPE i.

METHODS: ***process*** IMPORTING iv\_num TYPE i.

EVENTS: *cutoff\_reached*.

ENDCLASS. *"lcl\_main DEFINITION*

CLASS **lcl\_event\_handler** DEFINITION.

PUBLIC SECTION.

METHODS: ***handle\_cutoff\_reached*** FOR EVENT *cutoff\_reached* OF lcl\_main.

ENDCLASS. *"lcl\_event\_handler DEFINITION*

START-OF-SELECTION.

DATA: *lo\_main* TYPE REF TO lcl\_main.

DATA: *lo\_event\_handler* TYPE REF TO lcl\_event\_handler.

CREATE OBJECT lo\_main.

CREATE OBJECT lo\_event\_handler.

SET HANDLER lo\_event\_handler->handle\_cutoff\_reached FOR lo\_main.

lo\_main->process( 5 ).

CLASS **lcl\_main** IMPLEMENTATION.

METHOD ***process***.

v\_num = iv\_num.

IF iv\_num GE 2.

RAISE EVENT *cutoff\_reached*.

ENDIF.

ENDMETHOD. *"process*

ENDCLASS. *"lcl\_main IMPLEMENTATION*

CLASS **lcl\_event\_handler** IMPLEMENTATION.

METHOD ***handle\_cutoff\_reached***.

WRITE: 'Event Processed'.

ENDMETHOD. *"handle\_cutoff\_reached*

ENDCLASS. *"lcl\_event\_handler IMPLEMENTATION*