



**Masoud Aghadavoodi Jolfaei**

March 26, 2014 12 minute read

## ABAP Channels Part 2: Publish/Subscribe Messaging Using ABAP Messaging Channels

[Follow](#)

[RSS feed](#)

4 Likes 8,409 Views 26 Comments

### Introduction

For introduction to ABAP Channels please refer to [Introduction to ABAP Channels](#). For tangible examples in the system see also the blog [ABAP Channels Examples](#). See also [FAQ – ABAP Channels](#).

This blog focuses on the eventing framework for messages exchange between different ABAP sessions based on publish/subscribe channels,, also known as **ABAP Messaging Channel (AMC)**.

For live demonstration and step-by-step implementation look at [ABAP Messaging Channel Video](#).

### Integration of Messaging Channels into the ABAP Engine

In order to pass an event from an ABAP session to another session we decided to implement a publish/subscribe infrastructure in the ABAP engine. Any ABAP session can subscribe to channels and any ABAP session can publish messages to channels. The channels can be used to exchange messages between sessions in an ABAP system containing several application servers, no matter on which application server these sessions reside. The ABAP Messaging Channel (AMC) brokers take care of the exchange of messages between application servers (see figure 2.0).

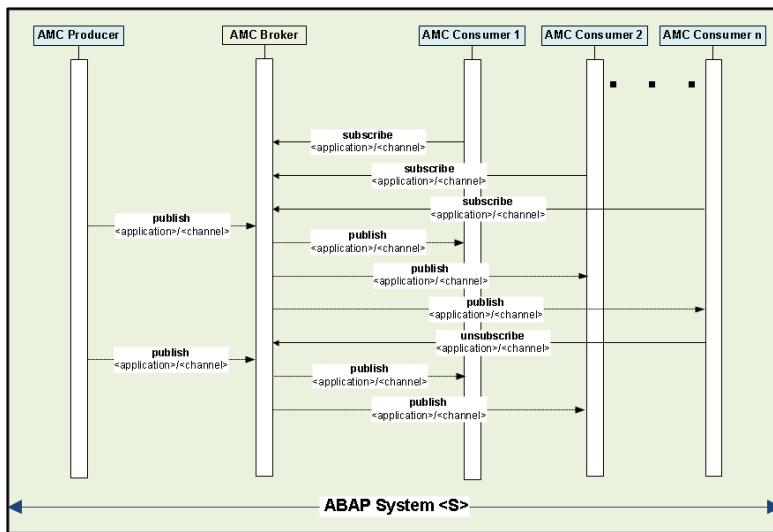


Figure 2.0: Simple interaction model of ABAP Messaging Channel

The following diagram (figure 2.1) shows the interaction model between AMC consumers and AMC producers in different ABAP sessions which are residing on different application servers. On top of the unidirectional communication also other communication patterns can be established, e.g. bidirectional communication. Furthermore a session can act simultaneously in both roles as AMC consumer and producer. The present ABAP Messaging Channels support only the best-effort quality of service of messaging but no reliable messaging such as *exactly-once* or *exactly-once in order*.

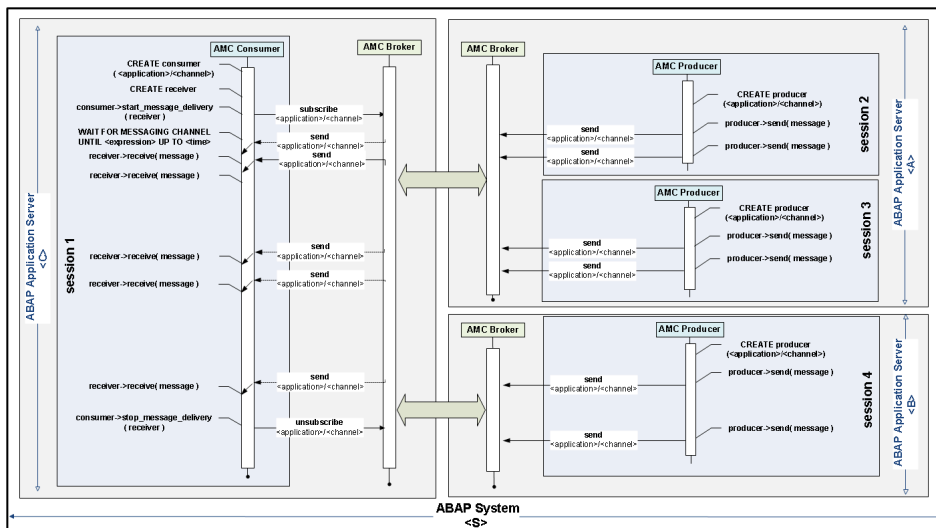


Figure 2.1: Exchange of messages based on ABAP Messaging Channel

The ABAP Messaging Channels support also the exchange of messages between ABAP sessions and WebSocket clients. This kind of communication is called “Collaborative” scenario. In order to reach this goal the ABAP Push Channel applications (see my blog on the [ABAP Push Channel](#)) are able to bind their WebSocket connection to ABAP Channels in a way, that the WebSocket client acts as consumer of AMC messages. This scenario is described in the dedicated blog “Collaborative Scenario”. Figure 2.2 illustrates a simple scenario for publishing a message, which is consumed by a WebSocket client.

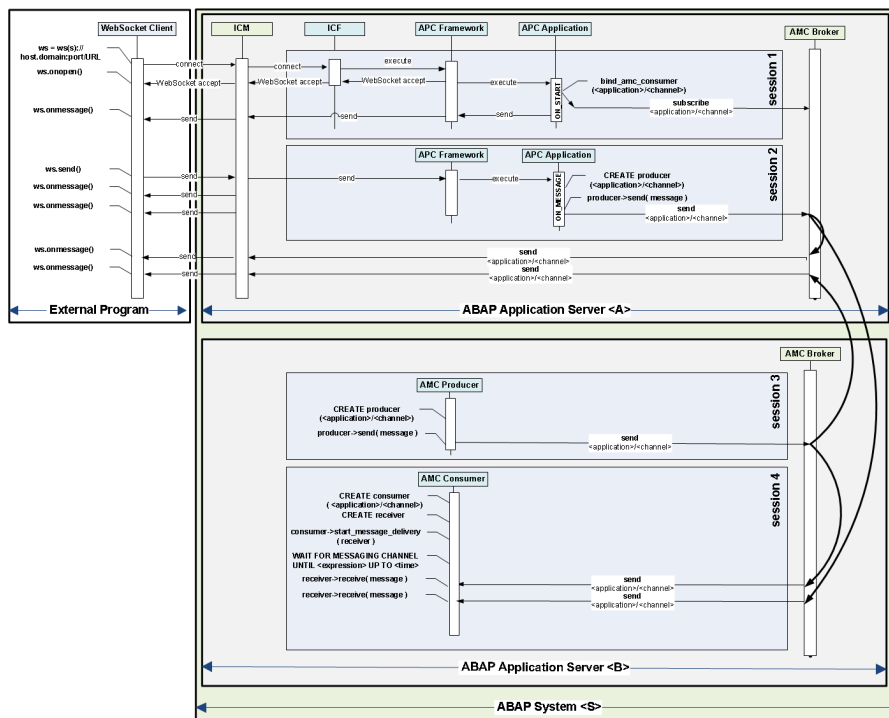


Figure 2.2: Publishing AMC messages to a WebSocket client using ABAP Push Channel

## Multicasting Granularity of ABAP Messaging Channels

The channels provide following possibilities to define/restrict the number of recipient of a message:

- The channel attribute “activity scope” describes the area to which the message is provided. The attribute can have the following values (see also next section):
  - *System*: The producers and consumers reside in the same ABAP system. An example for such a channel is the system messages channel which is used to send messages from transaction SM02 to the active users in the system.
  - *Client*: The producers and consumers reside in same client of the ABAP system. In other words the messages published in a client 100 can only be consumed by the session in the client 100.
  - *User*: The producers and consumers running under the same user identity, i.e. same client and user id.
- A channel consists of three sections “<application name><channel id>[<channel extension id>]”:
  - *Application name*: This section identifies the name space of the channel which contains the list of all channels that are maintained for this application.
  - *Channel ID*: This section is used to maintain the scope and security attributes. This is used to identify the consumers of a message.
  - *Channel Extension ID (optional)*: This as extension to *Channel ID* is used to identify the consumers of a message. With this the application is able to restrict the number of recipients of a message at runtime.

## Creating an ABAP Messaging Channel (AMC) Application

This section contains a step by step description of the creation of an ABAP Messaging Channel application.

An ABAP Messaging Channel (AMC) application can be created in two ways.

In transaction **SE80** using the context menu entry (“Right-Click on top-level package object”) “**Create**” -> “**Connectivity**” -> “**ABAP Messaging Channel**” (see figure 3.1):

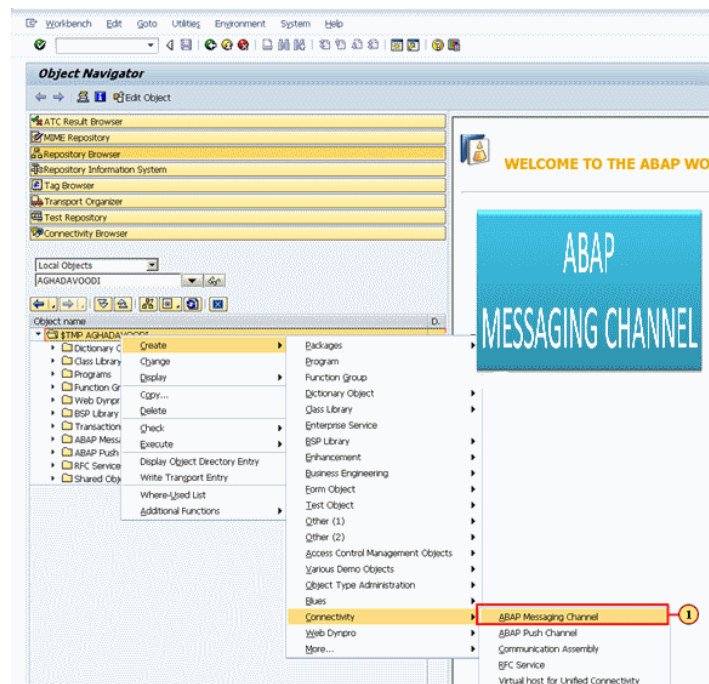


Figure 3.1.

As of release 740 SP5: In the transaction **SAMC** using the context menu entry ("Right-Click") on the "**ABAP Messaging Channel**" entry and selecting "**Create**" (see figure 3.2):

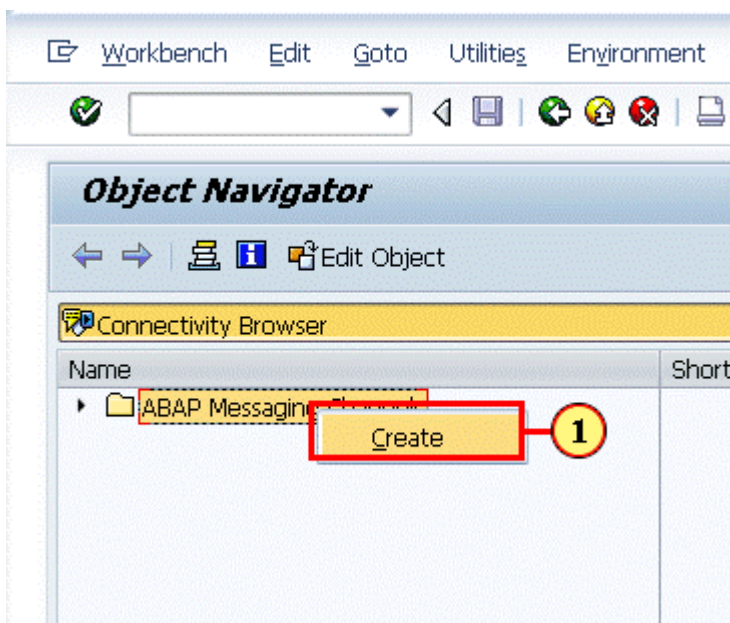


Figure 3.2.

In the next step, enter the name of the AMC application and confirm the popup screens (press "**Enter**"). In this example "**YAMC\_TEST**" is used as application name (see figure 3.3):

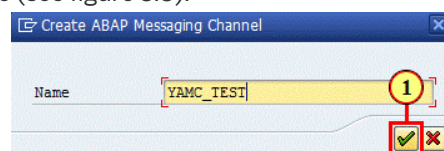


Figure 3.3.

Also choose an appropriate package entry, in this case as “Local Object” (see figure 3.4):

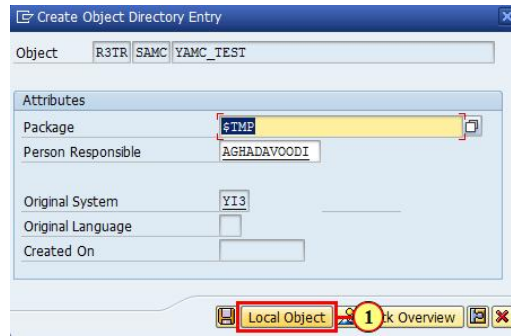


Figure 3.4.

Additionally, maintain the field “Description” (see figure 3.5):

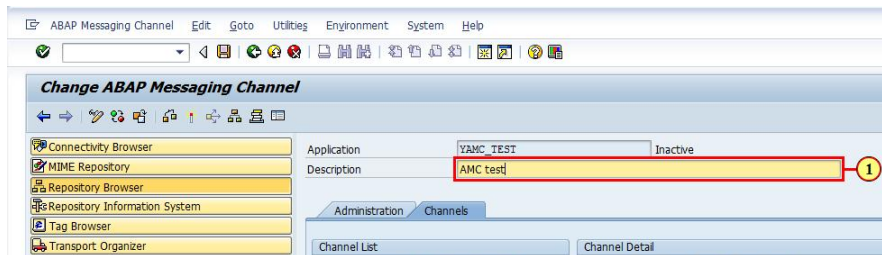


Figure 3.5.

On the tab „Channels“ enter a channel name of your choice e.g. „/ping“ and click on this field. The „Channel“ name **must** begin with slash (/) (see figure 3.6):

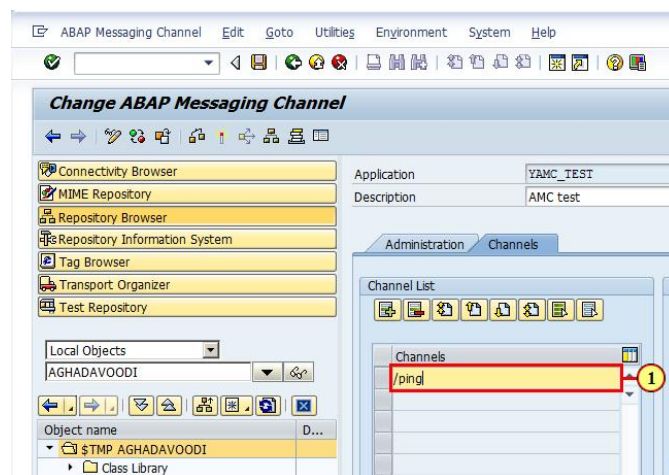


Figure 3.6.

Then, fill the next fields:

- **Message-Type ID** : A dedicated message type is assigned to each channel. Only messages of the assigned message type can be transferred over the channel. Using message types, the access APIs are type safe and the ABAP compiler and ABAP runtime ensure the consistency. Currently, the following message types are provided with the corresponding ABAP types and the producer and receiver interfaces.
  - Message type “TEXT”:
  - Corresponding ABAP type “STRING”

- Producer interface: IF\_AMC\_MESSAGE\_PRODUCER\_TEXT
- Receiver interface: IF\_AMC\_MESSAGE\_RECEIVER\_TEXT
- Message type “**BINARY**”:
- Corresponding ABAP type “XSTRING”
- Producer interface: IF\_AMC\_MESSAGE\_PRODUCER\_BINARY
- Receiver interface: IF\_AMC\_MESSAGE\_RECEIVER\_BINARY
- Message type “**PCP**” for Push Channel Protocol:
- Corresponding ABAP interface “IF\_AC\_MESSAGE\_TYPE\_PCP”
- Producer interface: IF\_AMC\_MESSAGE\_PRODUCER\_PCP
- Receiver interface: IF\_AMC\_MESSAGE\_RECEIVER\_PCP

The construction of the Push Channel Protocol (PCP) is very similar to an HTTP message with (header) fields, i.e. field name/field value pairs and a body with the exception that PCP does not contain any request/response line. Furthermore the field names are case-sensitive and in Unicode. The *Body* part is optional and can be either binary or text. A PCP object in ABAP is created via the class factory method CL\_AC\_MESSAGE\_TYPE\_PCP=>CREATE( ).

The PCP message type is the preferred message type used in the ABAP basis technology. This is because with the PCP messages not only application is able to add additional meta-data as field to the message but also the AMC runtime inserts per default the name of the channel to the PCP message before it is sent. This information is often necessary for the consumers which subscribe to different AMC channels. Using the *channel* field, i.e. “pcp-channel”, consumers can identify the channel to which the message belongs.

For this basic example we choose the message type TEXT by clicking on input help **Message Type ID** (see figure 3.7):

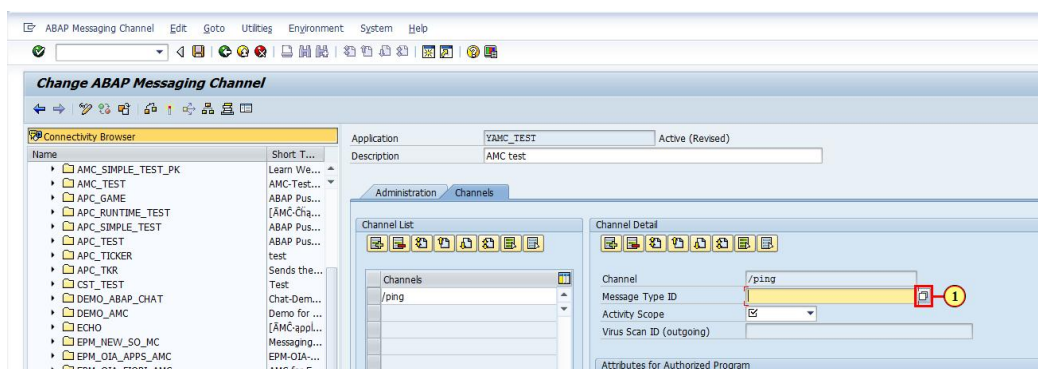


Figure 3.7.

Choose entry **TEXT** (see figure 3.8):

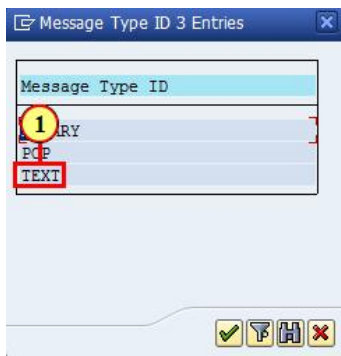


Figure 3.8.

- **Activity Scope (System, Client or User):** The channel access scope defines whether a channel is system-specific, i.e. cross-client, client-specific or user-specific. With client- and user-specific channel access scopes, the exchange of messages between producer and consumer sessions can be limited to sessions residing in the same client or sessions of the same user.

For this example select the entry Client (see figure 3.9):

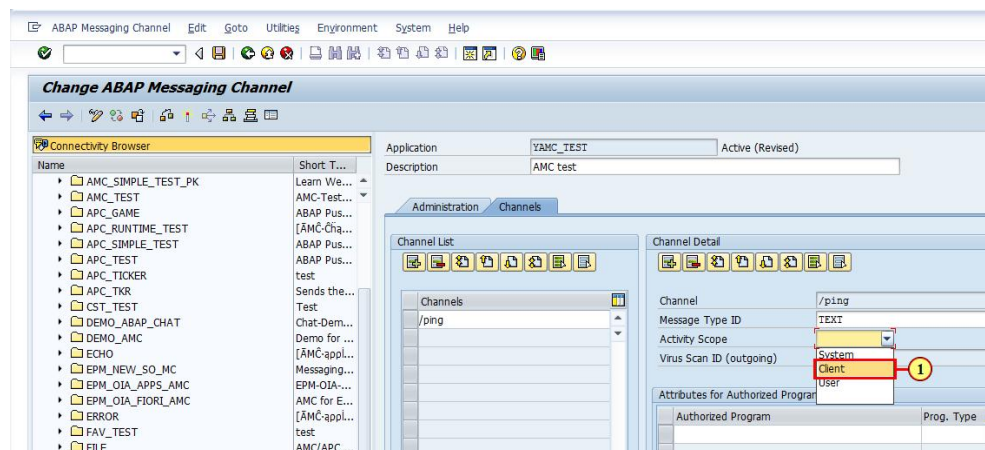


Figure 3.9.

- **Authorized Program:** The access rights for the channels are realized via code based authorization. For each channel, and depending on the access role as consumer or producer, a white list of ABAP reports, function groups, classes has to be maintained otherwise the access will be rejected.

For the maintenance of the access rights of the AMC channels dedicated reports in the role of producer, i.e. sender, and consumer, i.e. receiver, of AMC messages is needed. In the next steps following example reports will be created:

- Report YRS\_AMC\_RECEIVE\_MESSAGE\_TEXT will act as consumer for the AMC application “YAMC\_TEST” and the channel “/ping”.
- Report YRS\_AMC\_SEND\_MESSAGE\_TEXT will act as producer for the AMC application “YAMC\_TEST” and channel “/ping”.

The programming model of ABAP Messaging Channels is simple. The class factory CL\_AMC\_CHANNEL\_MANAGER provides AMC producer and consumer object. After creation of the AMC producer object for the respective channel, i.e. “<application



name><channel id>[<channel extension id>]”, messages can be published with the provided interface method SEND. For the consumer receiving messages can be started with the interface method START\_MESSAGE\_DELIVERY. For this action an (call back) object has to be provided which implements the interface receiver method RECEIVE. Furthermore in order to receive the message the session has to be in an “interrupted” state. This can be achieved explicitly with the new ABAP statement WAIT FOR MESSAGING CHANNELS <expression> [UP TO <time> SECONDS].

Create the test report YRS\_AMC\_RECEIVE\_MESSAGE\_TEXT in the transaction SE38 and copy-paste following ABAP code as context for the report (see figure 3.10):

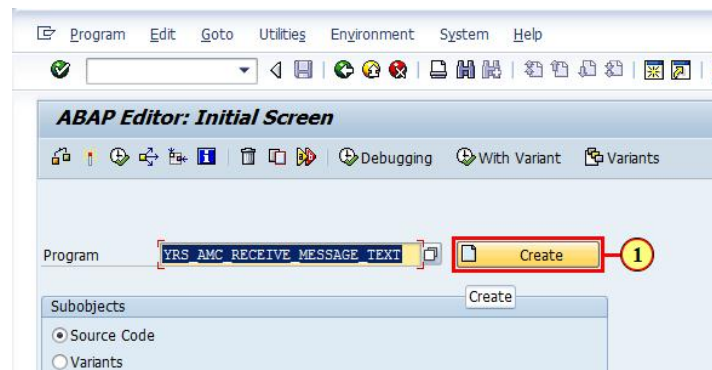


Figure 3.10.

Insert the title “Consumer test for ABAP Messaging Channel YAMC\_TEST” for the report and choose following Attributes (see figure 3.11):

- Type: “Executable program”
- Status: “Test Program”

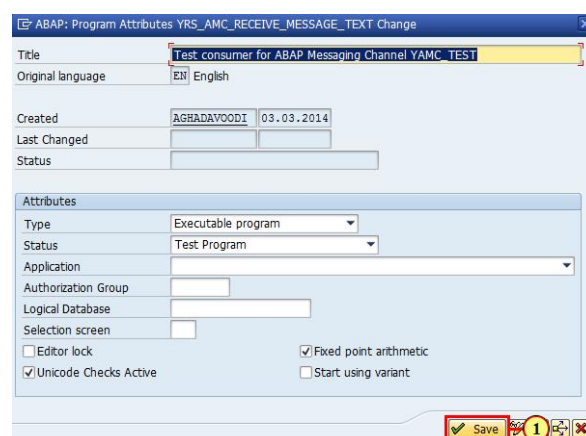


Figure 3.11.

Additionally click **Save** and select **Local Object** (see figure 3.12):



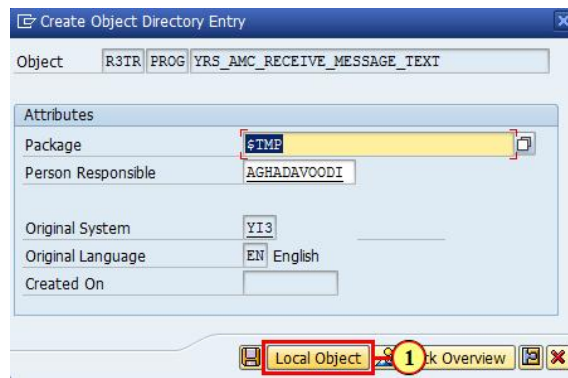


Figure 3.12.

Now insert the following ABAP code into the report and save the report (see figure 3.13):

### AMC consumer example code

```
REPORT yrs_amc_receive_message_text.
PARAMETERS: msg_nr TYPE i DEFAULT 1, "number of expected messages
wait_sec TYPE i DEFAULT 20. "waiting time to

DATA: lo_consumer TYPE REF TO if_amc_message_consumer.
DATA: gt_message_list TYPE TABLE OF string.
DATA: lv_message TYPE string.
DATA: lx_amc_error TYPE REF TO cx_amc_error.

* implemenation class for AMC receiver interface
CLASS lcl_amc_test_text DEFINITION
FINAL
CREATE PUBLIC .

PUBLIC SECTION.
* interface for AMC messages of type TEXT
INTERFACES if_amc_message_receiver_text .
ENDCLASS.

* Consumer test for ABAP Messaging Channel YAMC_TEST"
CLASS lcl_amc_test_text IMPLEMENTATION.
METHOD if_amc_message_receiver_text~receive.
*
* remark: in this method should the message be received and the main actions
* should be processed in the main program and usually after WAIT statement.
* Any kind of communication, e.g. RFCs, HTTP and message handling,
* e.g. error message is not supported and will lead to ABAP dump.
*
```

## AMC consumer example code

```
* insert received messages into the global table
  APPEND i_message TO gt_message_list.
ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
DATA: lo_receiver_text TYPE REF TO lcl_amc_test_text.

TRY.
lo_consumer = cl_amc_channel_manager=>create_message_consumer( i_application_id = 'YAMC_TEST' i_channel_id = '/ping' ).
  CREATE OBJECT lo_receiver_text.

" start of message delivery
lo_consumer->start_message_delivery( i_receiver = lo_receiver_text ).
  CATCH cx_amc_error INTO lx_amc_error.
    MESSAGE lx_amc_error->get_text( ) TYPE 'E'.
  ENDTRY.

*
* wait until a message is received but not longer than waiting time in seconds
*

WAIT FOR MESSAGING CHANNELS UNTIL lines( gt_message_list ) >= msg_nr UP TO wait_sec SECONDS.

IF sy-subrc = 8 AND lines( gt_message_list ) = 0.
  WRITE: '):- Time out occurred and no message received :-('.
ELSE.
  LOOP AT gt_message_list INTO lv_message.
* print out the list of received AMC messages
    WRITE: / sy-tabix, lv_message.
  ENDLOOP.
ENDIF.

ENDIF.
```

Figure 3.13.

Finally activate the report (see figure 3.14):

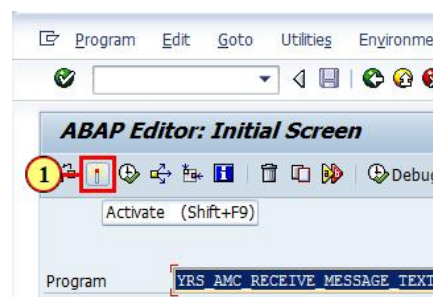


Figure 3.14.

Create the test report YRS\_AMC\_SEND\_MESSAGE\_TEXT in transaction SE38 and copy-paste the following ABAP code as context for the report (similar to figure 3.10-2.13) Insert the title "Producer test for ABAP Messaging Channel YAMC\_TEST" for the report.

Now insert the following ABAP code into the report and save the report (see figure 3.15):

AMC produce example code
<pre>REPORT yrs_amc_send_message_text. PARAMETERS: message TYPE string LOWER CASE DEFAULT 'Hi there !'. DATA: lo_producer_text TYPE REF TO if_amc_message_producer_text. DATA: lx_amc_error TYPE REF TO cx_amc_error.  TRY. lo_producer_text ?= cl_amc_channel_manager=&gt;create_message_producer( i_application_id = 'YAMC_TEST' i_channel_id = '/ping' ).  " send message to the AMC channel lo_producer_text-&gt;send( i_message = message ).  CATCH cx_amc_error INTO lx_amc_error. MESSAGE lx_amc_error-&gt;get_text( ) TYPE 'E'. ENDTRY.</pre>

Figure 3.15.

Finally activate the report (see figure 3.14):

Choose *Authorized Program* and insert the report YRS\_AMC\_RECEIVE\_MESSAGE\_TEXT in the popup (see figure 3.16).

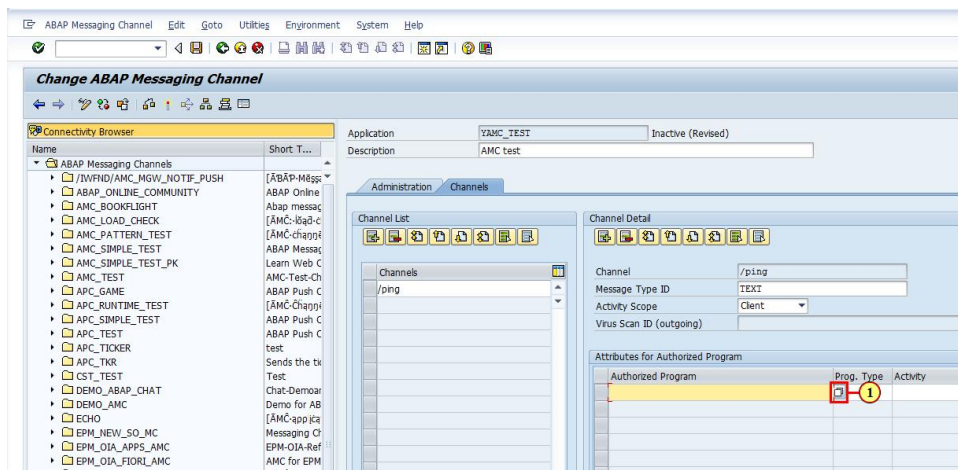


Figure 3.16.

Insert the report name YRS\_AMC\_RECEIVE\_MESSAGE\_TEXT and click **Accept** (see figure 3.17).

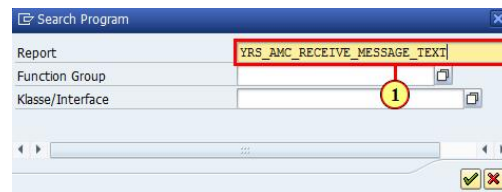


Figure 3.17.

Choose the activity Receive (see figure 3.18).

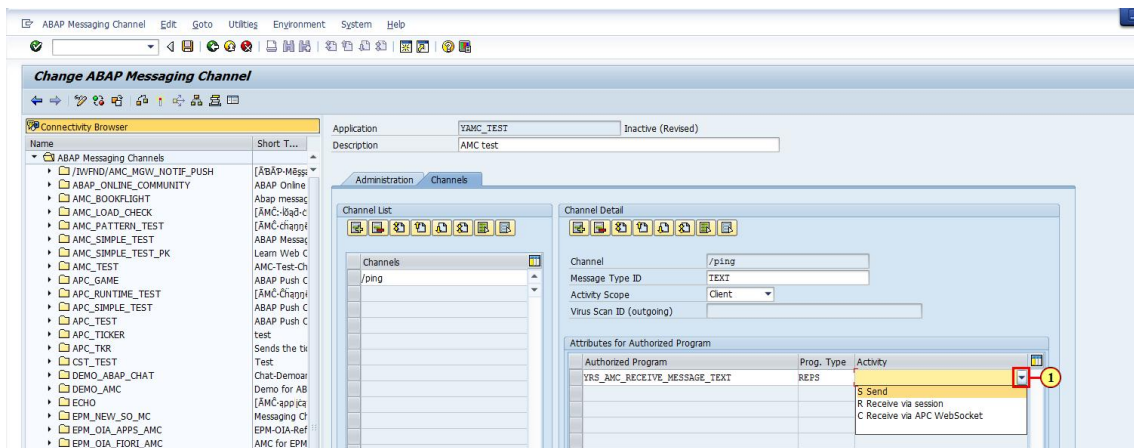


Figure 3.19.

Similar to the previous steps insert the report name YRS\_AMC\_SEND\_MESSAGE\_TEXT with the activity Send (see figure 3.20).

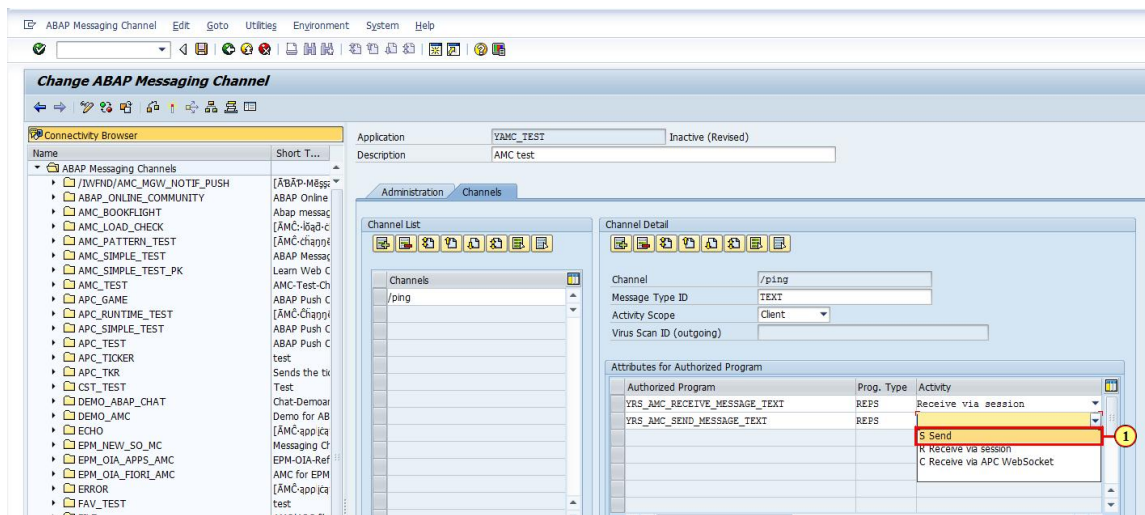


Figure 3.20.

Finally Save (see figure 3.21) and Activate (see figure 3.22) the AMC application. To accomplish these steps just choose Continue (press Enter) in the next popup screens (see figure 3.23).

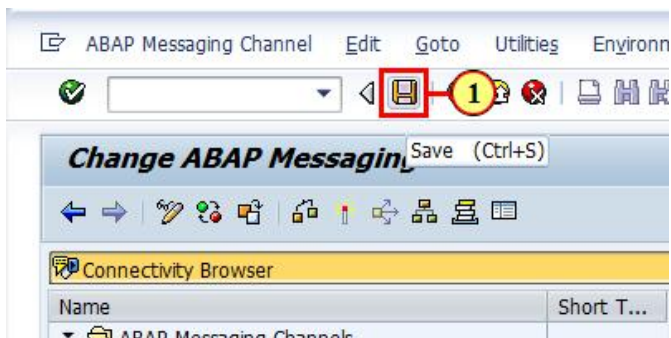


Figure 3.21.

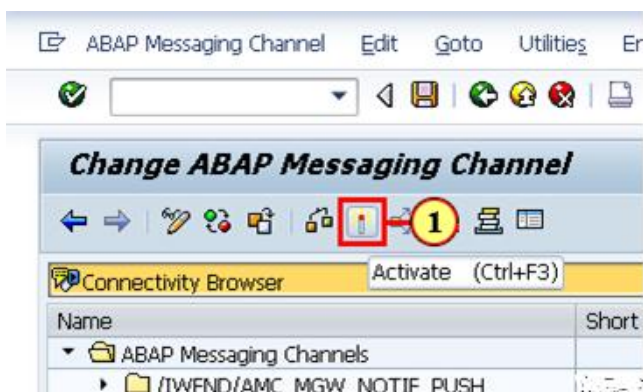


Figure 3.22

Click **Continue** (see figure 3.23).

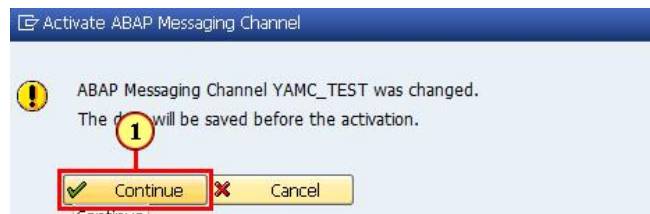


Figure 3.23.

Now execute first the ABAP report YRS\_AMC\_RECEIVE\_MESSAGE\_TEXT (see figure 3.24) and then the report YRS\_AMC\_SEND\_MESSAGE\_TEXT (see figure 3.25) in two parallel sessions. While the report YRS\_AMC\_RECEIVE\_MESSAGE\_TEXT report subscribes to the application YAMC\_TEST and channel /ping and waits for incoming messages, the report YRS\_AMC\_SEND\_MESSAGE\_TEXT sends messages to the same application and channel.

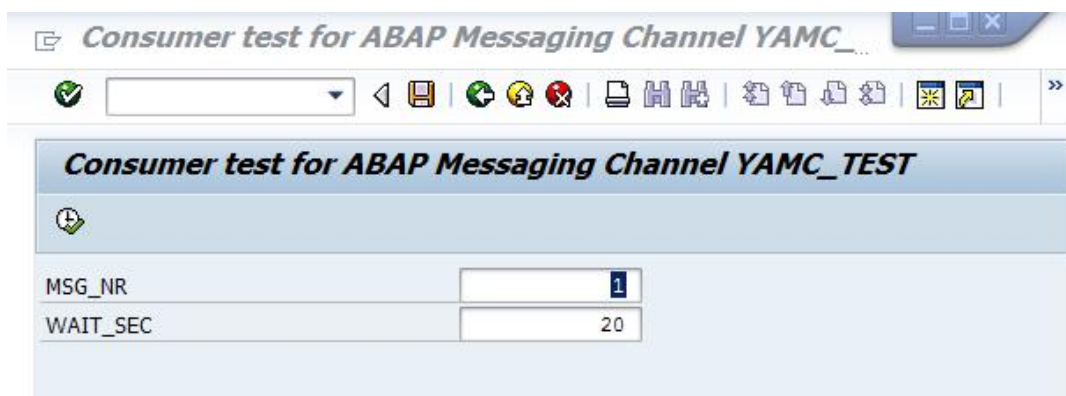


Figure 3.25.

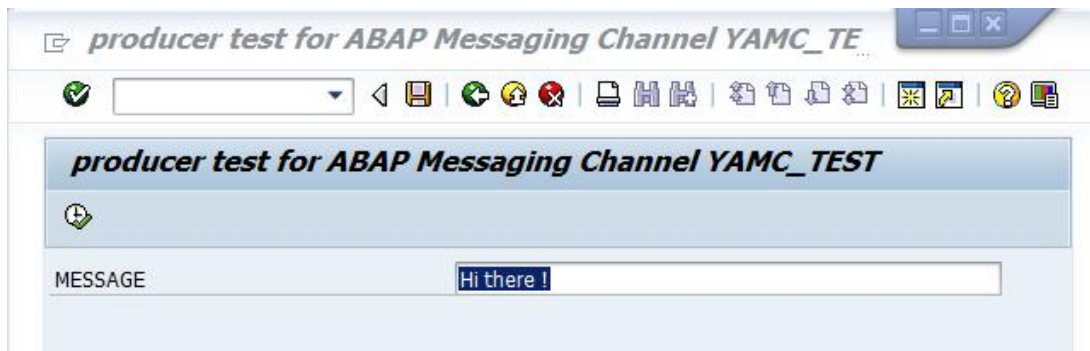


Figure 3.26.

**Remark:** If during of execution of an AMC program the AMC exception with the message

“No authority for program <program name> to access to ABAP Messaging Channel with application ID <application\_id> and Channel <channel\_id>”

is raised, this is because in the “Authorized Program” area (see figure 3.12) the reported program is not maintained appropriately.

**Limitation:** In the present version of AMC framework is the size of AMC messages limited to ~31 kbytes.

**Recommendation:** The basic goal of AMC messages is the notification of subscribers about an event. If the subscribers should be informed about any changes to an entity in this case is the transfer of the resource URL of the affected entity to the subscribers recommended. The URL should refer to the persisted entity record, e.g. in the database.

## AMC Security

The access rights for AMC channels are controlled via code based authorization and by maintenance of reports, i.e. classes, function groups or includes, which are permitted to apply an action, i.e. publish, subscribe, to the channel.

For the case, that a channel is bound to a WebSocket connection and is used to push messages from a session to a WebSocket client (see figure 2.2) , can a dedicated virus/content scan ID for outgoing messages be maintained.

## AMC Supportability

Since the AMC producers and consumers, besides the collaboration scenario, resides in an ABAP session, the existing ABAP supportability tools, e.g. debugger, ABAP runtime analysis or kernel trace can be used. Within the debugger the session breakpoints can be used to debug the session during send or receive phase of the messaging. Using transaction SAT the runtime analysis for the respective session and report can be activated. Via transaction SM04 you can activate and deactivate the user specific kernel trace for the respective session.

Furthermore with the following transactions you can manage the AMC logging (activate, deactivate and monitor recorded entries):

- Transaction **AMC\_LOG\_ADMIN**: Activate and deactivate logging for dedicated AMC channels.
- Transaction **AMC\_LOG\_PROCESSING**: Monitoring transaction for the AMC log entries.

**Note:** For the reorganization of AMC log entries a dedicated batch job has to be planned. The pre-defined standard batch-job

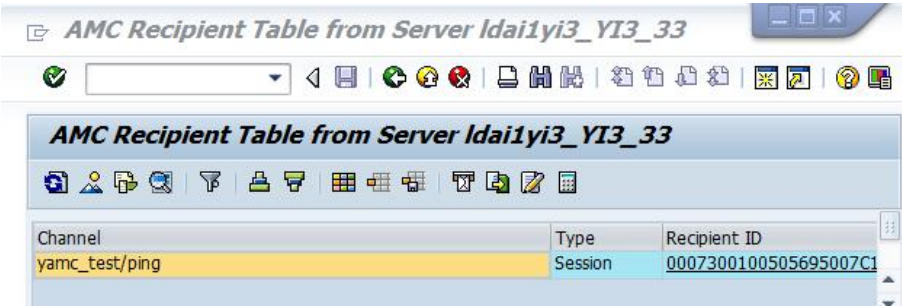
SAP\_ABAP\_CHANNELS\_MANAGEMENT has to be scheduled in the transaction SM36. The batch-job requires the authorization for the authorization object S\_AMC\_LOG with the activity "Delete". From release 740 SP09 is the standard job

SAP\_ABAP\_CHANNELS\_MANAGEMENT not available anymore and the planning should be done via transaction **AMC\_LOG\_ADMIN**.

For detail information refer to note 2043894.



The transaction „SMAMC“ (see figure 4.0) shows the list of the AMC channels (in our example the channel yamc\_test/ping) which are registered in sessions (Type “Session”) or in case of “Collaboration scenario” the WebSocket connections (Type is “WebSocket”) in the system as AMC consumers.



AMC Recipient Table from Server Idai1yi3_YI3_33		
Channel	Type	Recipient ID
yamc_test/ping	Session	0007300100505695007C1

Figure 4.0.

## Conclusion and outlook

The present implementation of AMC supports only the best-effort strategy without guarantee of delivery. We could think about providing also *reliable messaging* in future release.

The next related blog to ABAP Channels is [Collaboration scenario](#). The collaboration scenario blog describes, how by using **APC** and **AMC** can messages be pushed from ABAP sessions to WebSocket clients and vice versa.

Like    Alert Moderator

---

ABAP Connectivity | channel | communication | event | messaging |

[View more...](#)

## Related Blogs

---

[Real-time notifications and workflow using ABAP Push Channels \(websockets\) Part 1: Creating the APC and AMC](#)

By **Brad Pokroy** , Oct 16, 2014

[Introduction to ABAP Channels](#)

By **Olga Dolinskaja** , Nov 27, 2014

[ABAP Channels Part 3: Collaboration Scenario Using ABAP Messaging and ABAP Push Channels](#)

By **Masoud Aghadavoodi Jolfaei** , Apr 14, 2014

## Related Questions

---

[Asking for Hana Express / ABAP Trial Integration](#)

By **Former Member** , Nov 30, 2017

[Calling Mulesoft API from ABAP](#)

By **Tanmoy Mondal** , Dec 08, 2016

[ABAP Push Channel is not supported in the selected project](#)

By **Former Member** , Nov 03, 2016

## 26 Comments

You must be [Logged on](#) to comment or reply to a post.

---



**abilash n**

[March 31, 2014 at 3:39 pm](#)

Thanks Masoud for wonderful blog..... Great .. 😊 😊

Like (0)



**Former Member**

[April 11, 2014 at 10:12 pm](#)

I think this is a very important technology improvement to the ABAP platform and for our platform provides a much more compelling case to upgrade to SP05 than any HANA aspects. Thanks for the blog!.

Like (0)



Former Member

November 10, 2014 at 4:35 pm

So all of you who are planning to use AMC on NW 7.40 SP8, there is a bug in Standard SAP which would not allow you to add any Custom Function Group to a channel. It gives you an error that the Function is not a valid ABAP program. I raised a message with SAP and they have fixed this bug and would be available in SAP note 2092735 once it is released.

Like (0)



Ekansh Saxena

December 23, 2014 at 12:03 pm

Hi Masoud,

Can you please explain in detail about channel extension ID. I searched on SCN and help portal but could not find anything which details about channel extension ID.

Like (0)



Masoud Aghadavoodi Jolfaei | Post author

December 23, 2014 at 12:30 pm

Hi Ekansh,

the main goal of channel extension ID is to build a group of receivers. The channel extension extends the channel for exchanging of messages. For instance with a client-specific channel, i.e. AMC application id & channel id with Activity Scope = Client, all receivers residing on the same client as the sender session can receive a message. With the specification of a channel extension ID only those receivers with the same channel extension ID will be able to receive the published messages. You can check this with the reports RS\_AMC\_SEND\_MESSAGE and RS\_AMC\_RECEIVE\_MESSAGE with default parameter settings and with and without specification of a "Channel Extension ID".

Cheers,

Masoud

Like (0)



Former Member

March 16, 2015 at 3:28 pm

Hi Ekansh,

As Masoud mentioned using the channel extension id you could trigger multiple instances of the AMC with different extension ID during runtime. What that means really is using a single AMC you can create multiple channels at runtime which would not interact with each other based on their extension ID's.

Regards,

Manish

Like (0)



Former Member

February 18, 2015 at 8:59 pm

Hi Masoud,

A similar question asked to you by Sergiu Popa on your 1st Blog on WebSockets. Do we have any profile parameters which would control the number of active Message channels in the system? Any recommendations on how many concurrent Message Channels we could have before system performs slowly?

Regards,

Manish

Like (0)



Former Member

February 18, 2015 at 9:17 pm

Hi Masoud,

I have got a profile parameter `rdisp/max_amc_receiver_entries` which defines the number of amc running parallel at a time. Apart from this any other factors which would affect the performance of a system with regards to AMC?

Regards,

Manish

Like (0)



Masoud Aghadavoodi Jolfaei | Post author

February 19, 2015 at 10:42 am

Hi Manish,

the parameter `rdisp/max_amc_receiver_entries` defines the maximum number of AMC registrations per application server. The AMC receiver table per application server has a fixed size given by this parameter. For a higher value, the server requires slightly more memory. As this receiver table is used for dispatching AMC messages on an application server, with heavily increased number of registrations the evaluation of the table for the identification of the receivers will take longer.

Best regards,

Masoud

Like (0)



Former Member

February 26, 2015 at 10:20 pm

So now I am eagerly waiting for your next blog on how to use PCP Protocol for Channels. 😊😊😊

Like (0)



Former Member

May 19, 2015 at 4:31 am

Hi Masoud,

Thanks a lot for this wonderful Blog. I Wanted to know how to integrate APC with Fiori Applications since we have the Odata gateway between the Fiori Ui and the Abap Backend System.

Like (0)



Former Member

July 24, 2015 at 5:50 am

Hi Masoud,

Is there a best practice defined to use ABAP Push Channels integrated with standard SAP transactions?

I am in the process of creating a Sales dashboard and want to integrate it with the Billing transaction (VF01). Every Billing or Rejection done will be reflected in the corresponding metrics in the Sales dashboard. Do we go for invoking the message channel using userexits or output control? One thing is certain it should not impact the main transaction therefore should be an after SAVE activity.

regards

Nitesh

Like (0)



Masoud Aghadavoodi Jolfaei | Post author

July 24, 2015 at 6:08 am

Hi Nitesh,

unfortunately not and have a long list of “should-write” blogs. As an AMC send does not trigger any database commit, you can trigger the event whenever it makes sense. In this csae after the SAVE (COMMIT WORK) and using one of those extensions (userexists or the SD output control).

Best regards,

Masoud

Like (0)



Former Member

July 29, 2015 at 5:23 am

Hi Masoud,

Any update on your next blog on how to use PCP Protocol for Channels?

regards

Nitesh

Like (0)



Masoud Aghadavoodi Jolfaei | Post author

July 29, 2015 at 6:36 am

Hi Nitesh,

I have described in the blog Specification of the Push Channel Protocol (PCP) the message structure of a Push Channel Protocol (PCP). I recommend to use the message type PCP in the ABAP Channels projects. The basic idea is – whenever it is possible – to share a single WebSocket connection for multiplexing of messages belonging to different channels and consumed by a (UI) client. Just have a look into the blog “Introduction to ABAP Channels” and section General recommendations for usage of ABAP Channels.

Cheers,

Masoud

Like (0)



Titus Thomas

August 31, 2015 at 11:23 am

Hi Masoud,

I have a doubt when executing the report "YRS\_AMC\_RECEIVE\_MESSAGE\_TEXT". Since the statement "WAIT FOR MESSAGING CHANNELS" is executed, the process flow is interrupted and actively waits for a message. What if i want to show a report in an ALV grid in the GUI screen, and only if any change happens, i need to refresh data. Otherwise, if no messages are received, i would like to continue displaying the report and for user to make changes to the GUI, just like how we do in UI5 or Web Dynpro.

Regards,

Titus

Like (0)



Titus Thomas

September 1, 2015 at 9:45 am

Hi,

Are there any other alternatives to WAIT FOR MESSAGING CHANNELS, that can be used in GUI programs. Because, with this statement, the WAIT statement locks my work process in wait mode, and the busy indicator is shown. I wont be able to do anything on the screen.

My requirement was to show a GUI report, and if any change happens in the backend, to auto-update/refresh the GUI screen, without the WAIT statement.

Regards,

Titus

Like (0)



Masoud Aghadavoodi Jolfaei | Post author

September 1, 2015 at 10:43 am

Hi Titus,

just have a look into the "Comments" section of the blog "[Say Goodbye to Polling: Real-time eventing in ABAP with ABAP Channels](#)". There, e.g. my reply on Nov 28, 2014 4:13 PM , you may find some interesting information regarding AMC message handling in ABAP engine.

Cheers,

Masoud

Like (0)



Titus Thomas

September 2, 2015 at 9:31 am



Thanks Masoud for the quick information. That was exactly what i wanted to know. Thanks for the elaborate and detailed description and also the limited functionalities as of now. Are you able to let us know when this will be available for productive use ?

Like (0)



Former Member

May 24, 2016 at 9:51 am

Hi Masoud,

Thanks for this blog. it is more than 2 year old but still informative.

I am planning to put receiver as background Job, and sending message from sender from user exit. As soon as I receive message from sender I will lets say submit report or call proxy and again will wait for next message in background.

My question is..Keeping receiver as background job for day long will have any load on system?

Thanks!

Viral Shah

Like (0)



Masoud Aghadavoodi Jolfaei | Post author

May 24, 2016 at 10:16 am

Hi Viral,

I assume you are starting the “background job/daemon” as an asynchronous RFC session (without response) running in a dialog process and not as a batch job. In this case occupies only the allocated memory (check it in SMO4 transaction) the server resources which is neglectable.

Cheers,

Masoud

Like (0)



Former Member

May 24, 2016 at 10:49 am

Hi Masoud,

Thanks for quick input.

Basically I am just running example program given in the blog “YRS\_AMC\_RECEIVE\_MESSAGE\_TEXT” from Se38 -> Run in Background -> Immediately.

And yes I checked in SM04 and it is not consuming much resource. This will help in proposing solution.

Is there any do and don't for AMC? like do not run receiving program in background?

Again Thanks for the information.

Thanks!

Viral Shah

Like (0)



Former Member

July 15, 2016 at 9:47 am

Hy all,

that's a nice blog and the demos are working.

I have to admit, now comes the “but”.

I don't get how this makes a gui-appliaction more responsive.

All this demos still are blocking the users interaction. So, what's the beenfit in use for a Ui Scenario?

The javascript examples are working nicely ( using the right browser ).Is there any component that wraps this functions of listening without blocking user interaction at least for webdynpro ?

Thanks ,

Ulrich Becker

Like (0)



Former Member

August 22, 2016 at 6:15 am

Hi,

May you know when it could be released for productive usage? This is a weapon to fix some heavy performance issue.

Like (0)



Deepak Tewari

July 26, 2017 at 11:47 am

Hi Masoud,

That's a very informative blog for starting with IOT. Thanks a lot for your effort.

I have a problem. When I execute the receiver ABAP program it gives me the error:

Expected receiver interface IF\_AMC\_MESSAGE\_RECEIVER\_PCP is not implemented for application ID ZZAMC\_TEST\_485 and channel /ping.

and when I execute the sender program it is terminated with a Dump:

**Runtime Error - Description of Exception**

Long Text Debugger

Category	ABAP programming error
Runtime Errors	MOVE_CAST_ERROR
Except.	CX_SY_MOVE_CAST_ERROR
ABAP Program	ZZAMC_SENDER_485
Application Component	Not assigned
Date and Time	26.07.2017 04:46:13

The current ABAP program "ZZAMC\_SENDER\_485" had to be terminated because it found a statement that could not be executed.

**Error analysis**

An exception has occurred which is explained in more detail below. The exception, which is assigned to class 'CX\_SY\_MOVE\_CAST\_ERROR' was not caught and therefore caused a runtime error. The reason for the exception is:  
During the 'CAST' operation ('?=' or 'MOVE TO') an attempt was made to assign a reference to a reference variable.  
The current content of the source variable is not compatible with the target variable however.  
Source type: \CLASS=CL\_AMC\_MESSAGE\_TYPE\_PCP  
Target type: "\INTERFACE=IF\_AMC\_MESSAGE\_PRODUCER\_TEXT"

**Missing handling of system exception**

Program	ZZAMC_SENDER_485
---------	------------------

Please help me out on this. I am eager to try this application.

Regards

Deepak

Like (0)



Masoud Aghadavoodi Jolfaei | Post author

January 6, 2018 at 11:07 am

Hi Deepak,

sorry for the late response. According to the attached ABAP runtime error a casting error during execution of the report occurs. At the moment I assume this is because of mismatch of usage of APC client interfaces if\_apc\_wsp\_messag vs if\_apc\_wsp\_messages\_pcp or providing improper subprotocol during creation phase using cl\_apc\_wsp\_client factory class and assignment of value for i\_protocol parameter. This is 'V10.PCP.SAP.COM' for WebSocket target application of type PCP vs. none for no subprotocol.

Just for tests have a look into the test report RS\_APC\_WSP\_EXCHANGE\_MESSAGE in the ABAP system.

Cheers,

Masoud

Like (0)

## Share & Follow

Privacy

Terms of Use

Legal Disclosure

Copyright

Trademark

Cookie Preferences

Sitemap

Newsletter