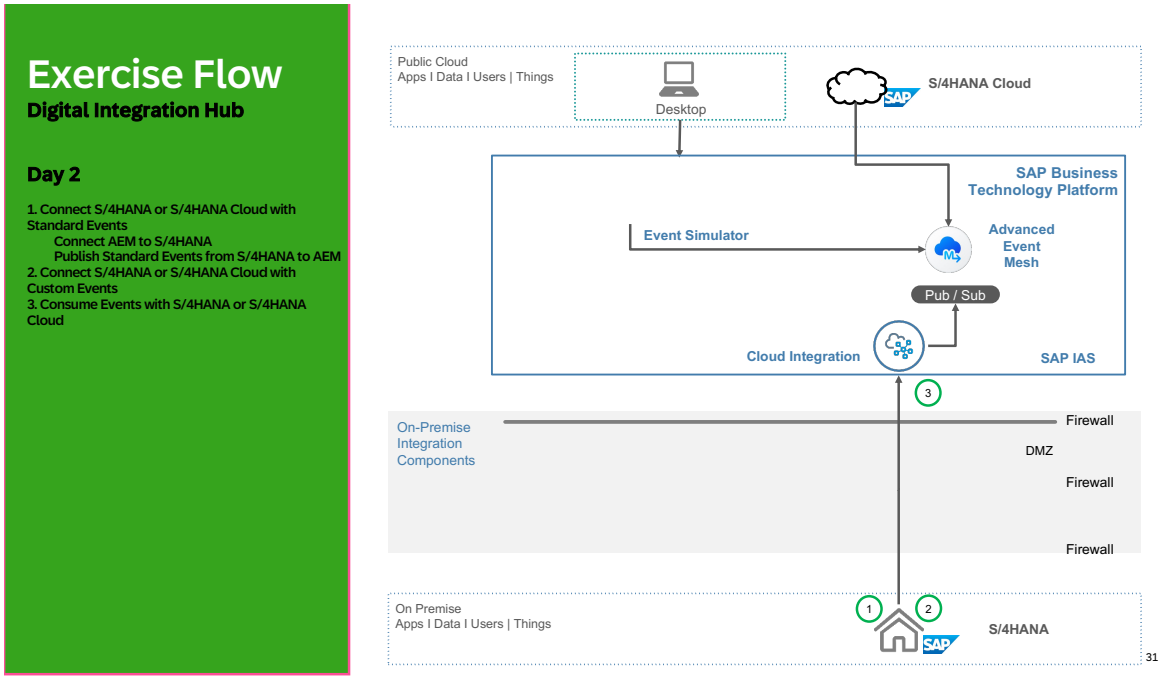
Topic 2

1. Connect S/4HANA with Standard Events
    a. Connect AEM to S/4HANA
    b. Publish Standard Events from S/4HANA to AEM
2. Connect S/4HANA with Custom Events
    a. Creating custom events
    b. Creating custom events with deep payloads
    c. Customize standard events using derived events
    d. Filtering derived events using CDS views
    e. Creating topic filters
3. Consuming events
    a. Creating local event handlers
    b. Receiving events with S/4HANA



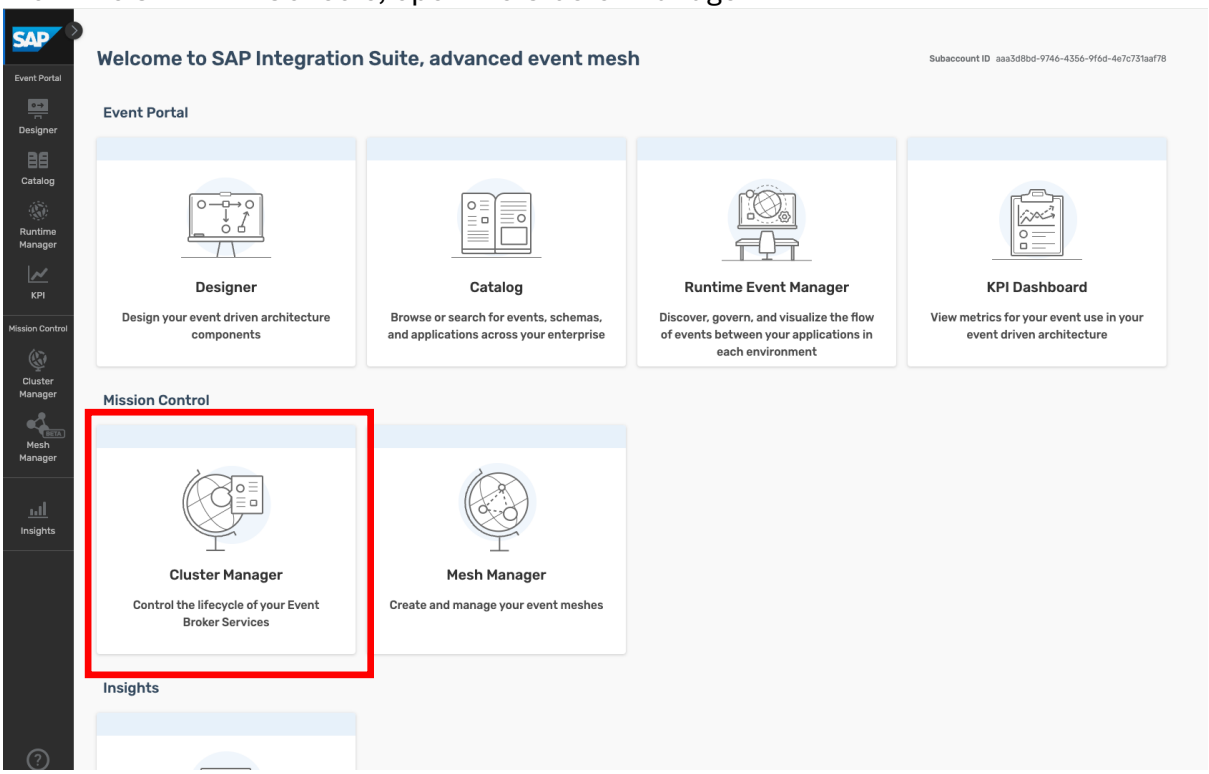
Connect S/4HANA with Standard Events

In this task we'll be connecting the Advanced Event Mesh to the S/4HANA system and configuring the Business Partner Changed event as a standard event to be sent to the AEM. In doing so we'll be creating two destinations within transaction *SM59*. One for the AEM broker and one for the VMR. While the destination for the AEM broker has to be

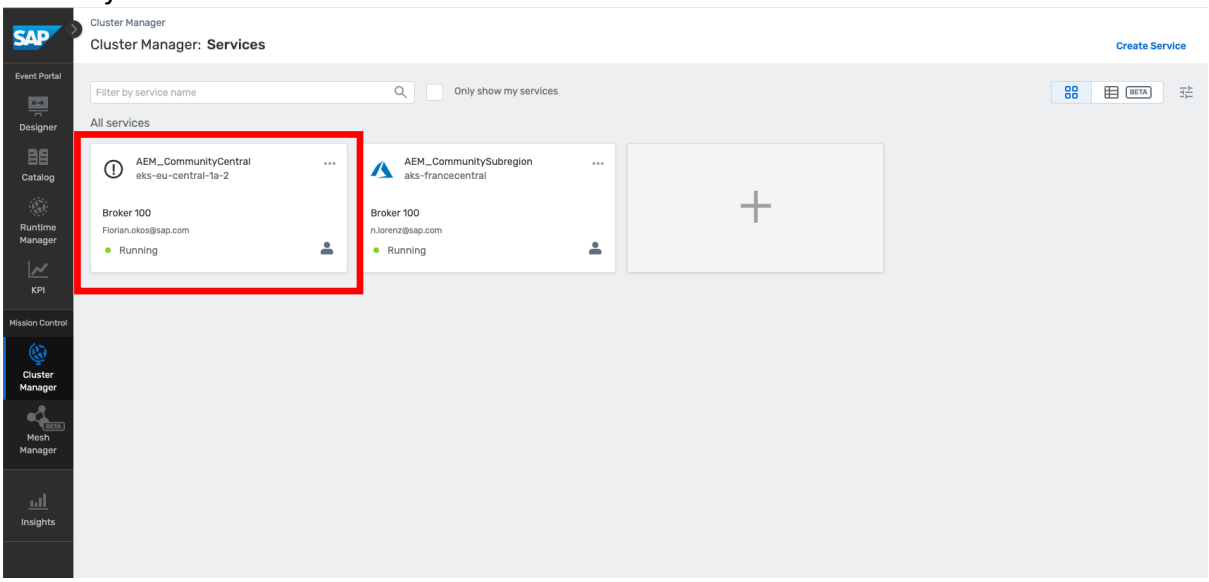
setup manually, the destination for VMR can be setup automatically via the service key from the BTP.

Connect AEM to S/4HANA

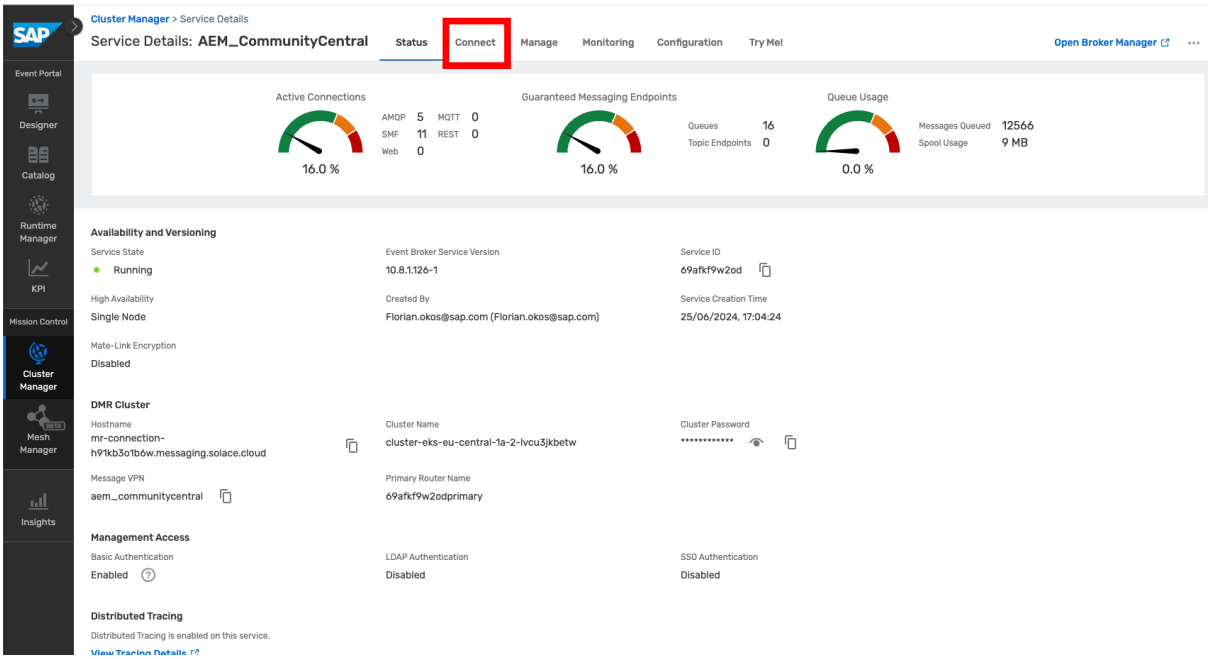
1. From the SAP AEM Console, open the Cluster Manager.



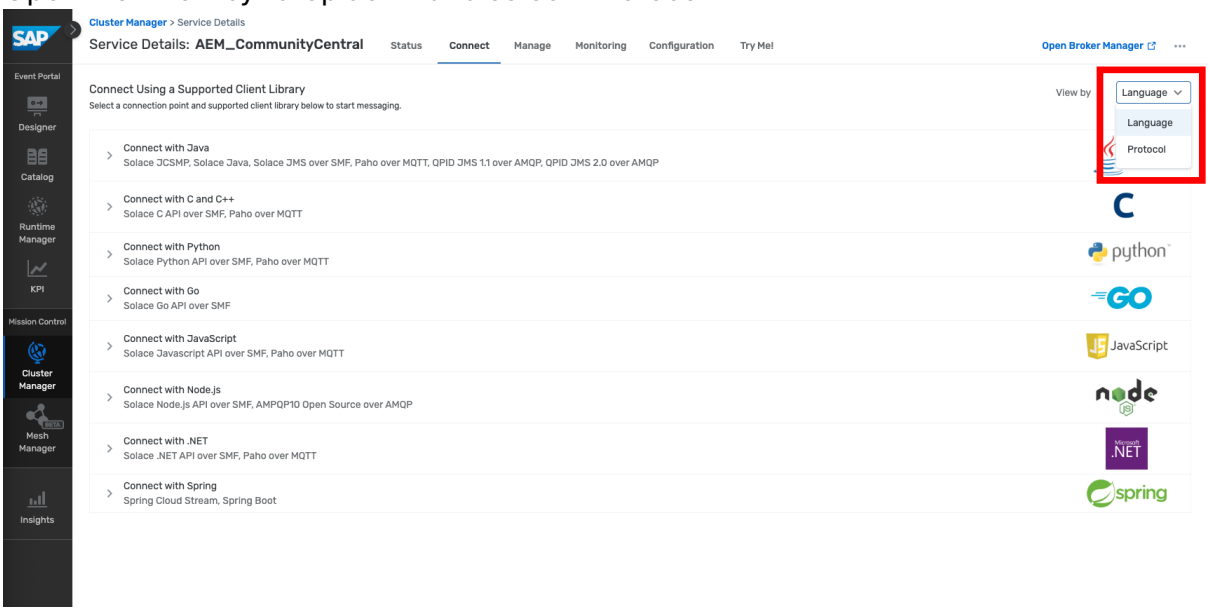
2. Select your broker



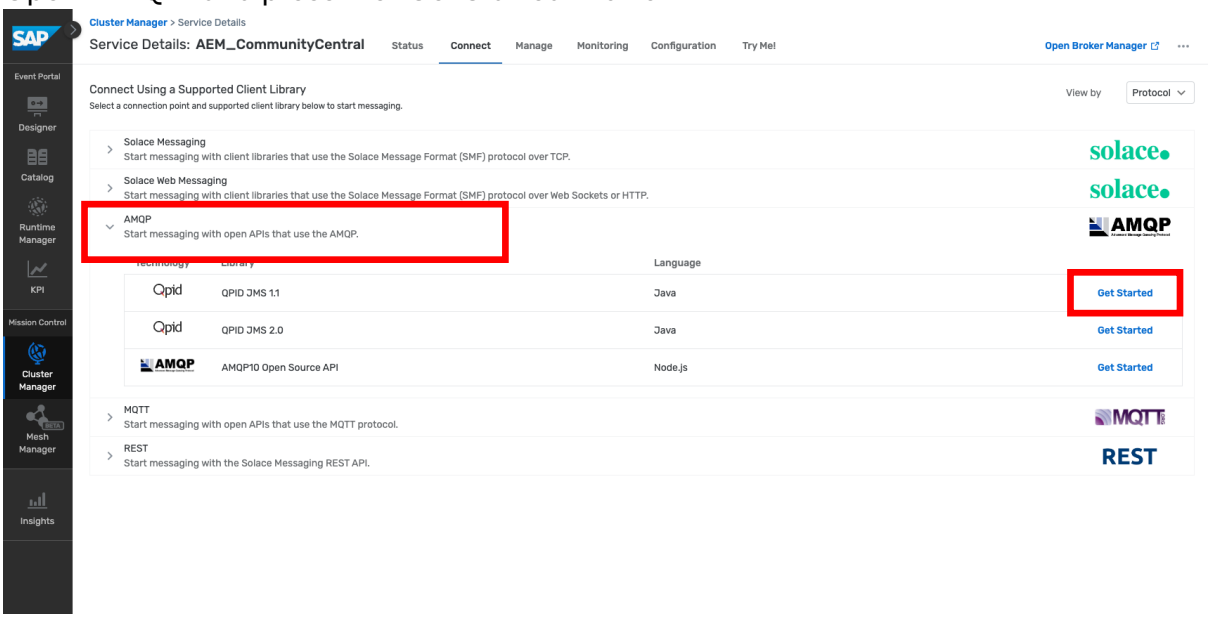
3. Select the “Connect” tab



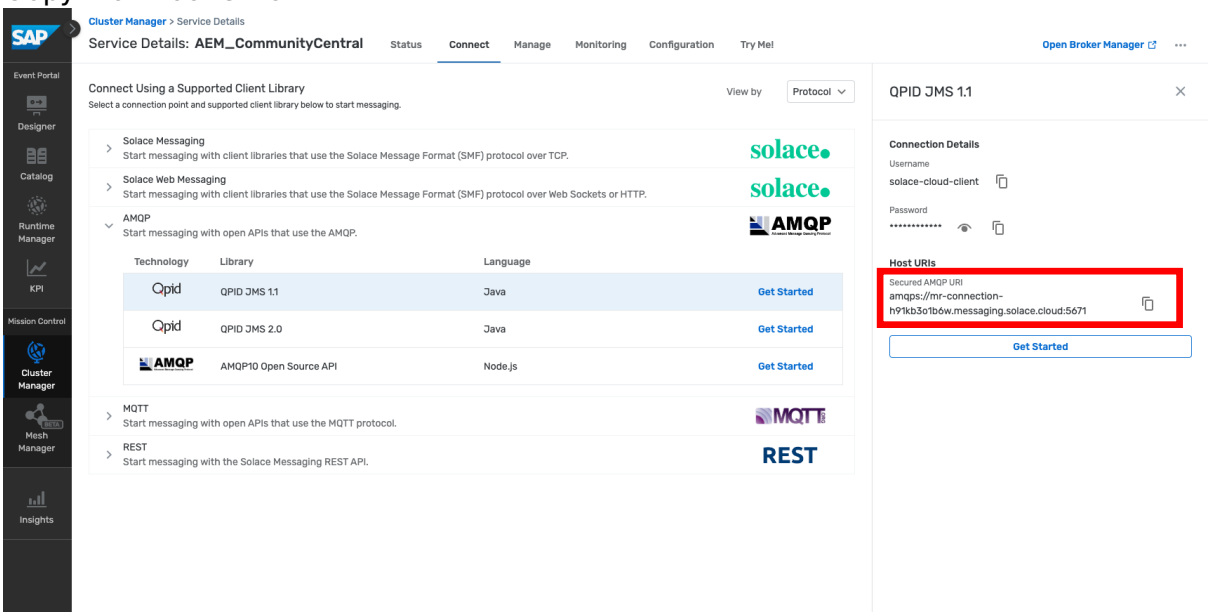
4. Open the “View by” drop down and select “Protocol”



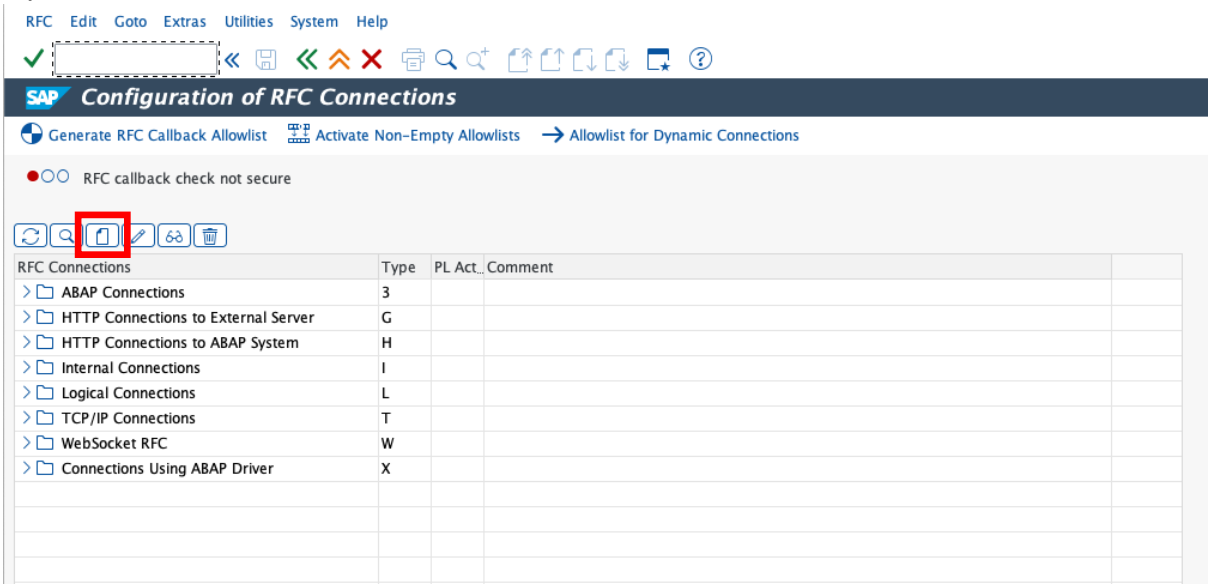
5. Open “AMQP” and press the “Get Started” Button



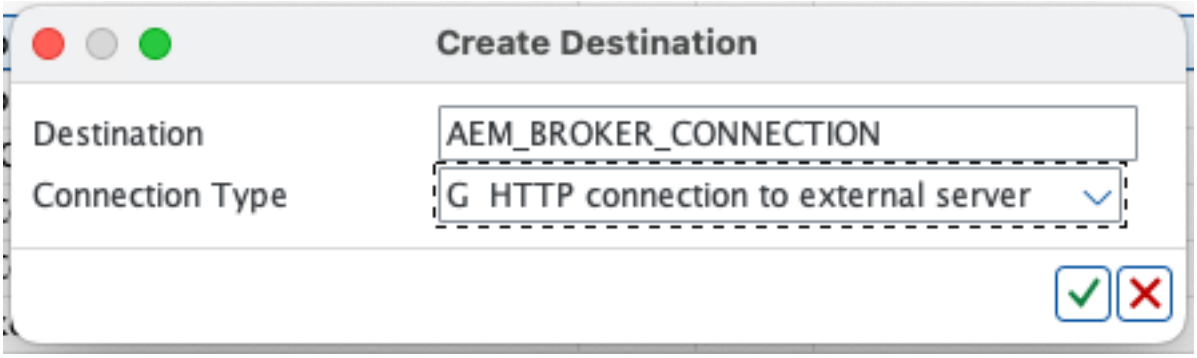
6. Copy the “Host URIs”



7. Open transaction *SM59* and create a new destination

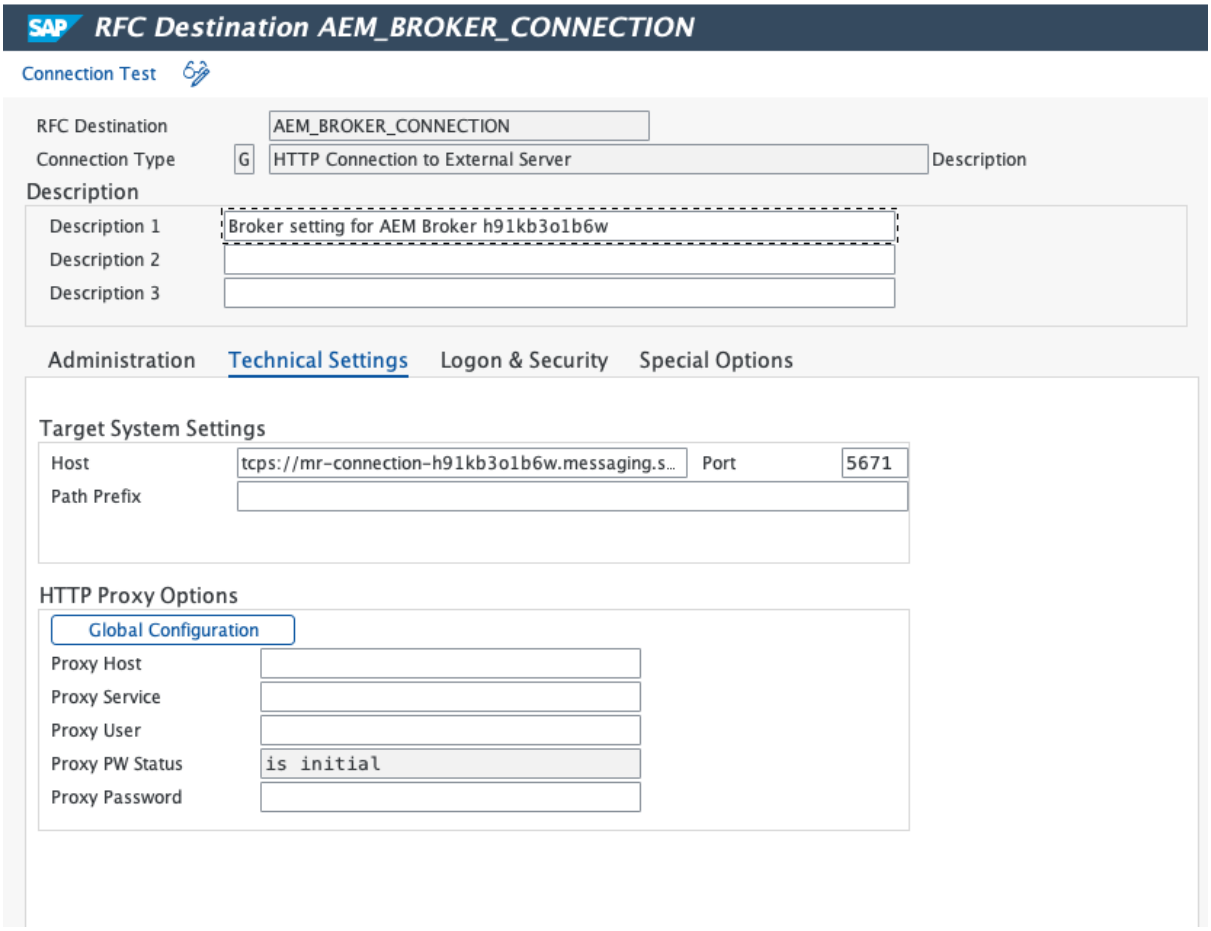


8. Name the destination “AEM_BROKER_CONNECTION” and select Connection Type “G”

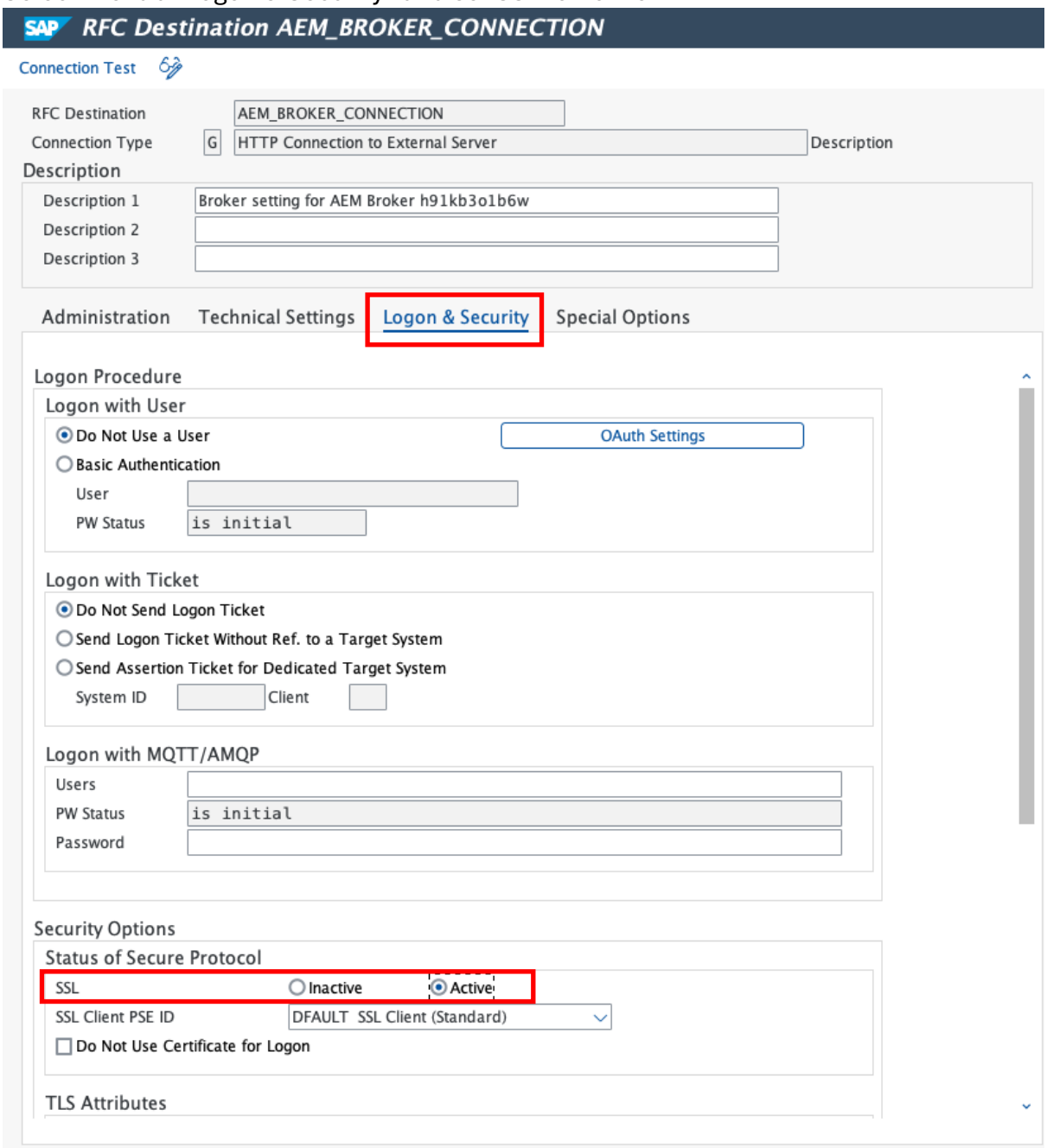


9. Select Continue
10. Enter “Broker setting for AEM Broker” as description and under Target System Settings enter the previously copied host uri as host. Replace the starting

protocol “amqps” from the host with the protocol “tcps”



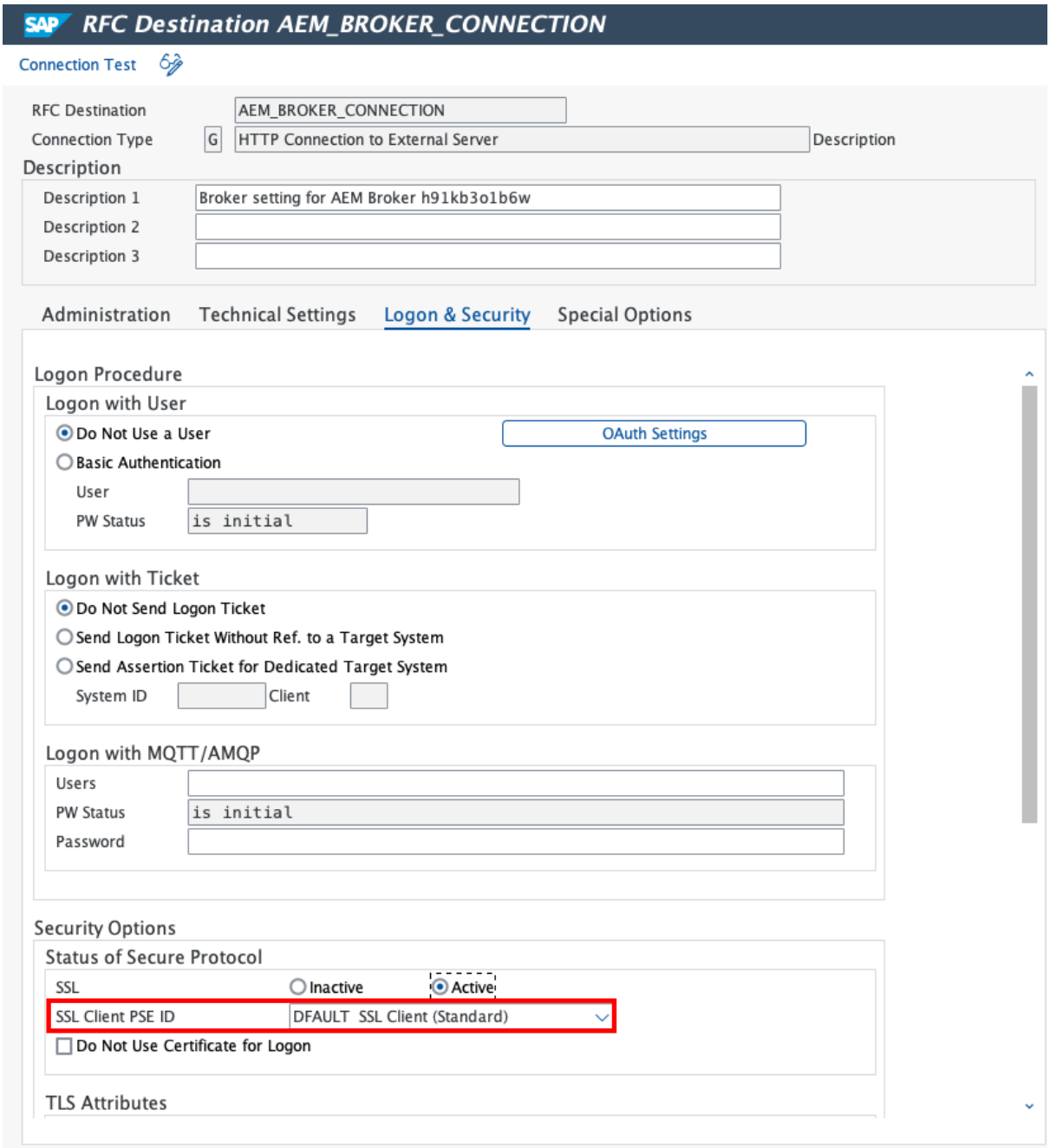
11. Select the tab “Logon & Security” and set SSL to Active



12. Hit Save

The Enterprise Event Enablement integration within S/4HANA uses Client Certificate Authentication to connect to AEM. Therefore the next step is to export the necessary certificates from S/4HANA and import them to AEM.

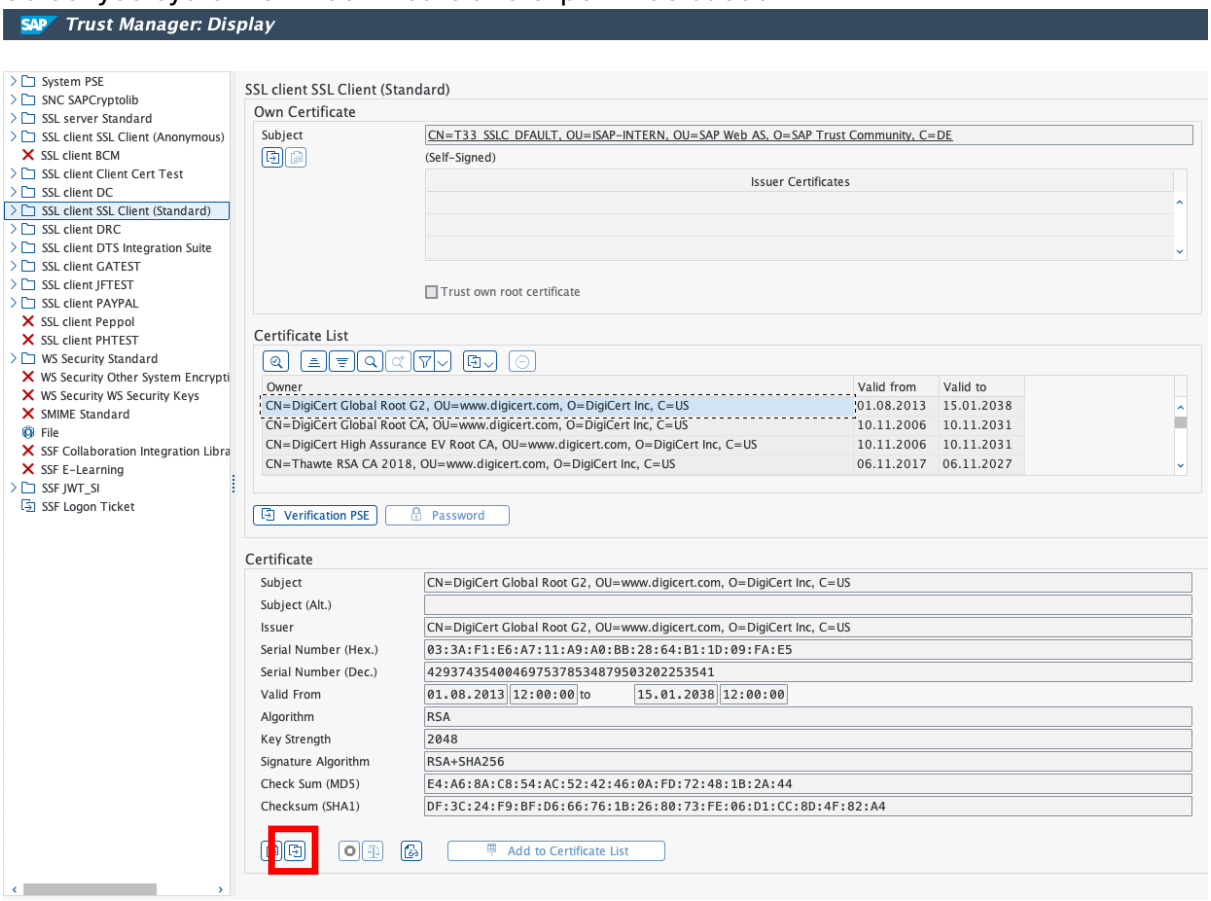
13. Remember the “SSL Client PSE ID”



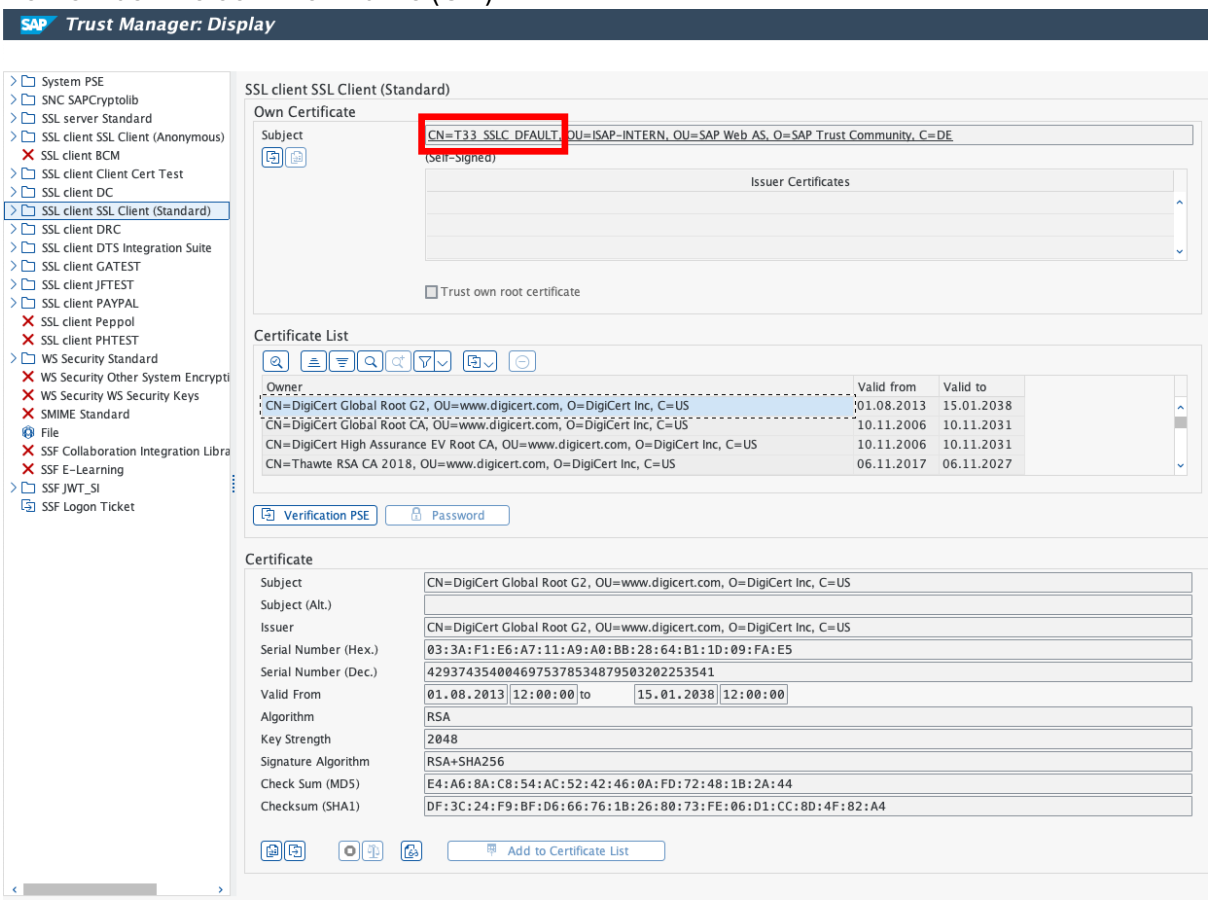
That completes the setup of the first destination. Now lets configure authentication for the Advanced Event Mesh

14. Open the transaction *STRUST*
15. Select the SSL Client node given in the destination setup

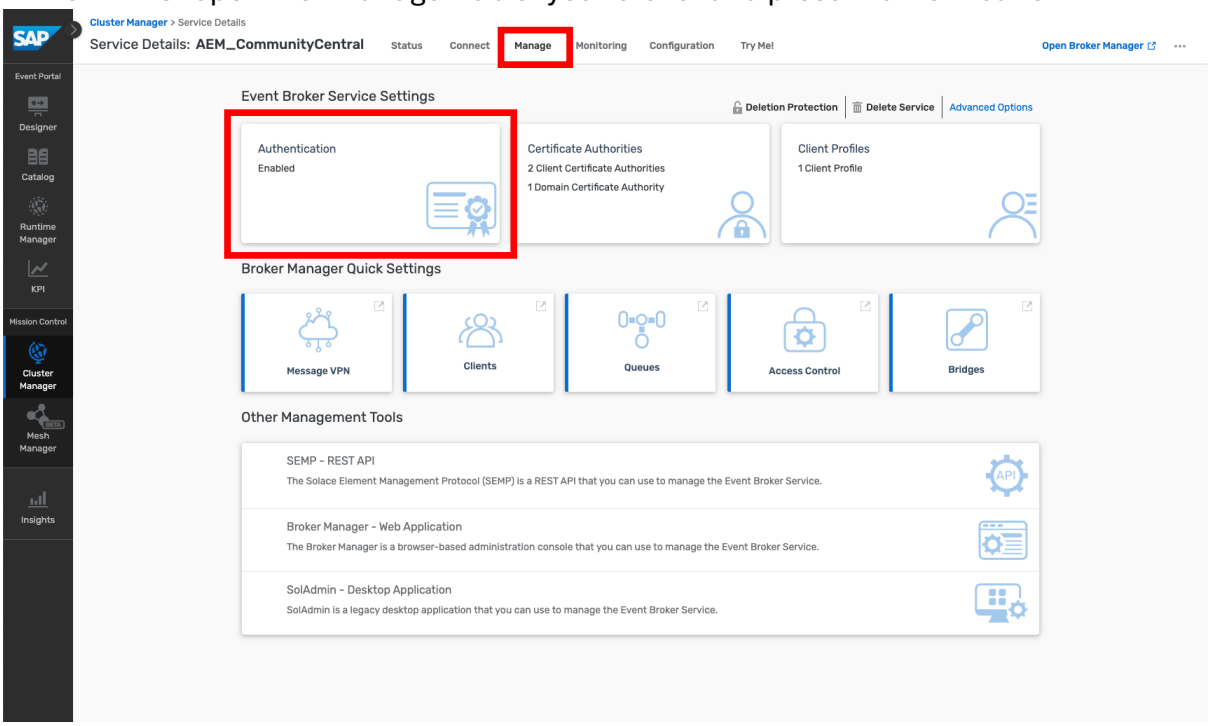
16. Select you system own certificate and export it as base64



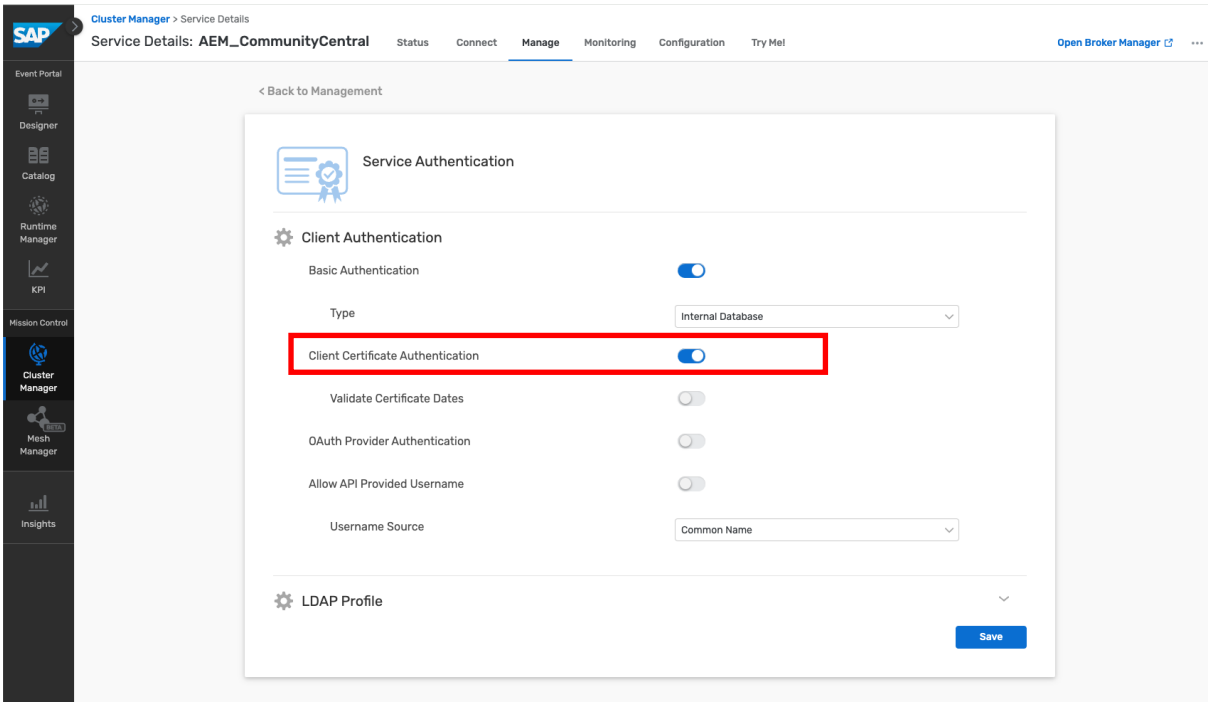
17. Remember the common name (CN)



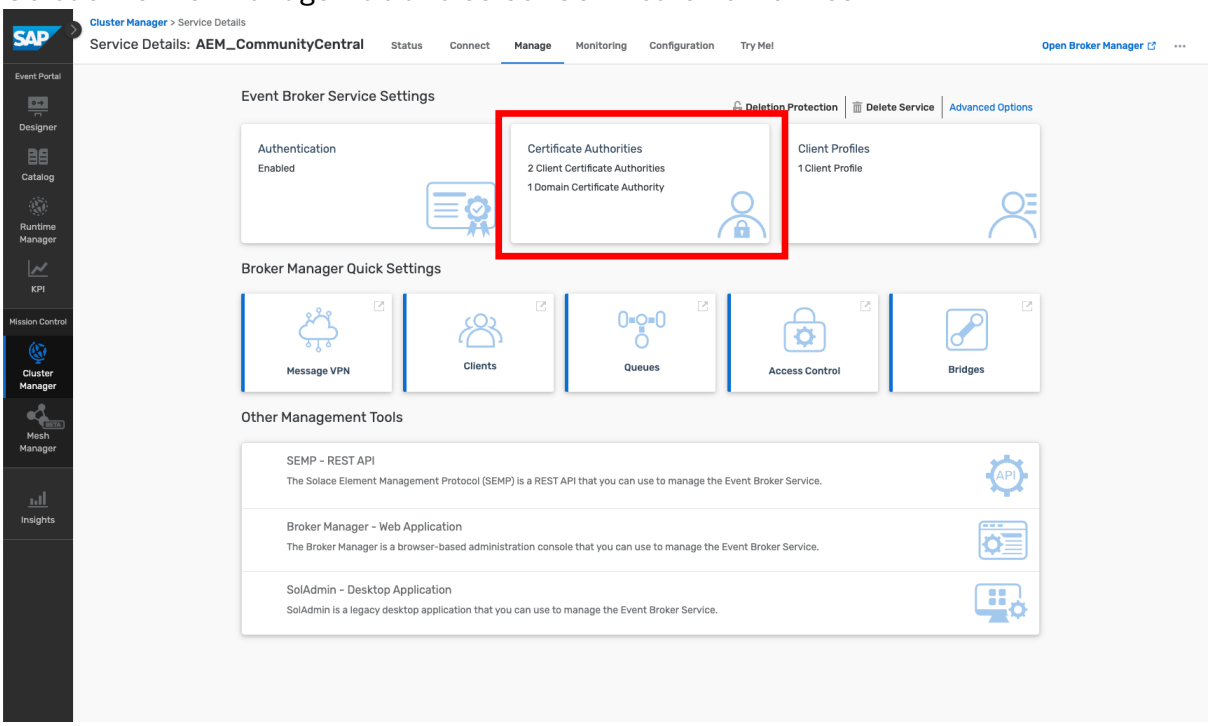
18. In the AEM UI open the “Manage” tab of your broker and press “Authentication”



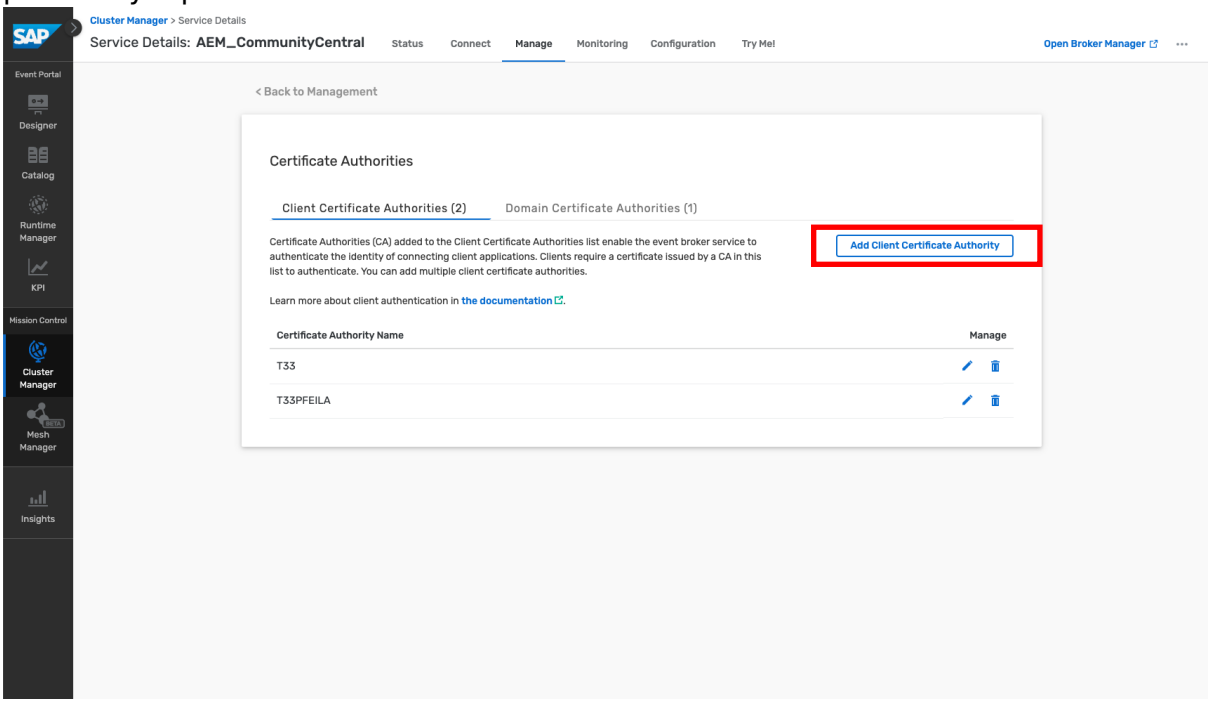
19. Enable “Client Certificate Authentication”



20. Go back to the “Manage” tab and select Certificate Authorities”

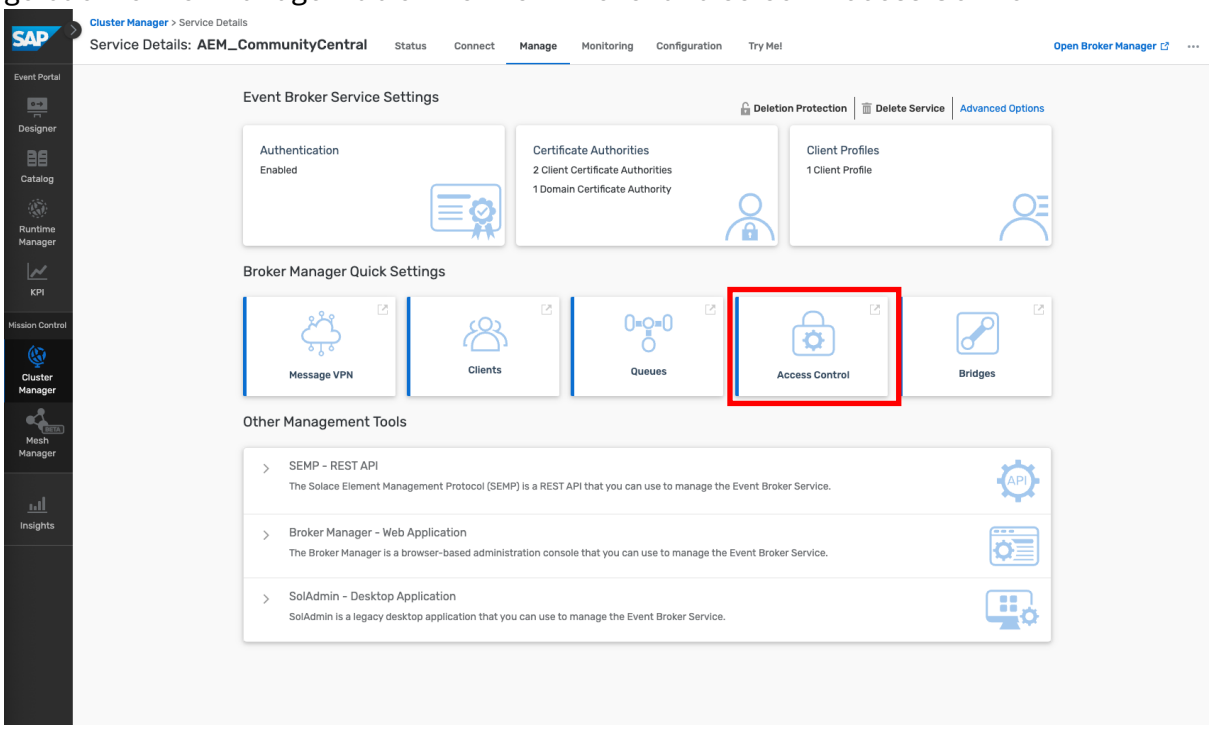


21. Click “Add Client Certificate Authority”, give the certificate authority name “S/4HANA” (You can also give more meaningfull names) and upload the previously exported certificate.

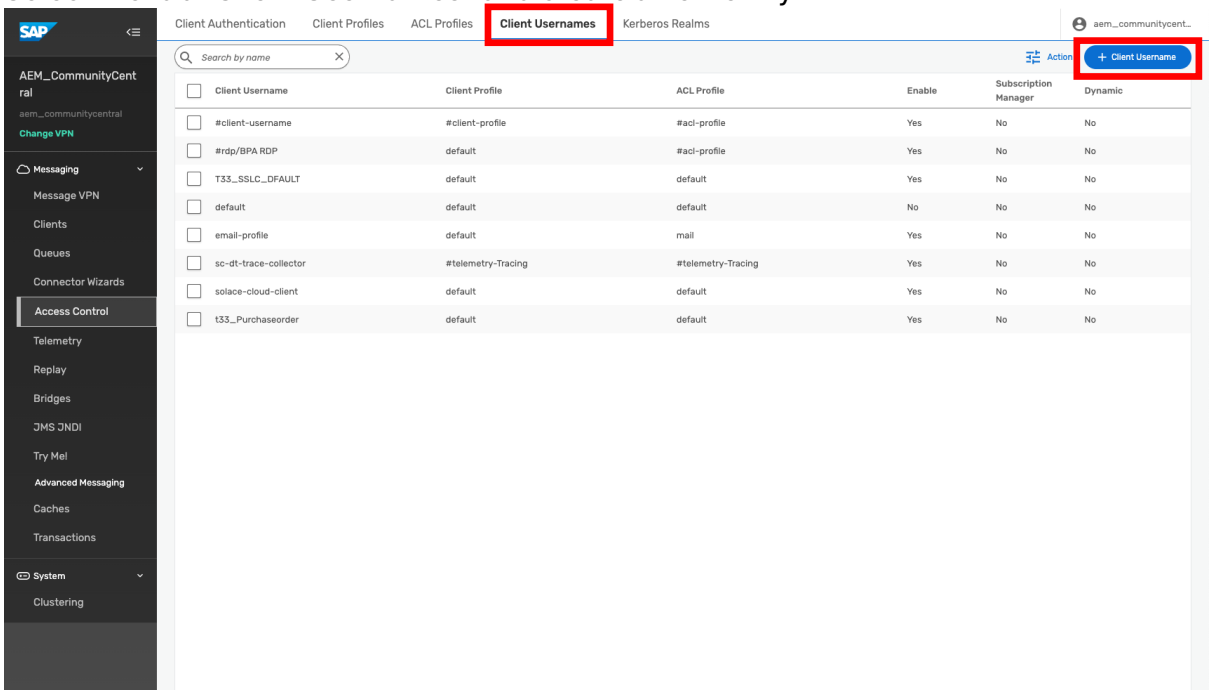


22. Save the Certificate Authority

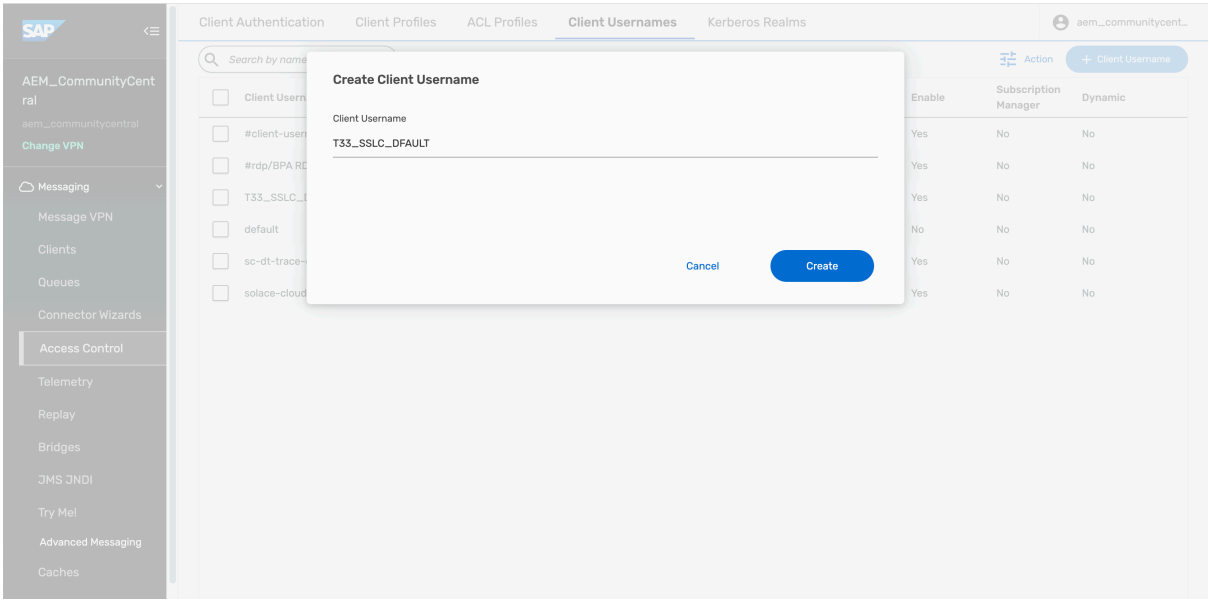
23. go back to the “*Manage*” tab of the Event Broker and select “*Access Control*”



24. Select the tab “Client Usernames” and create a new entry

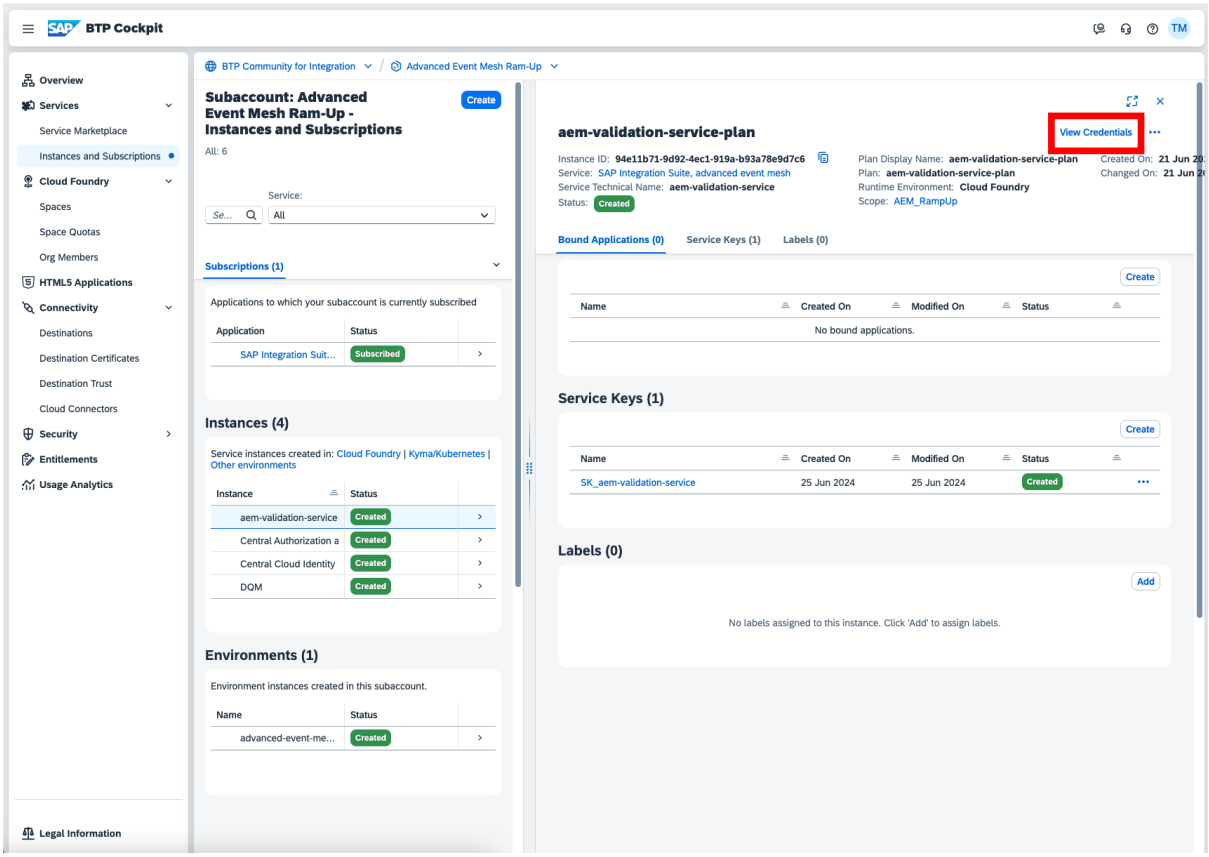


25. Enter the in step 17 mentioned common name (CN) (here T33_SSLC_DFAULT) as Client Username and hit “Create”. Don’t forget to enable the username.



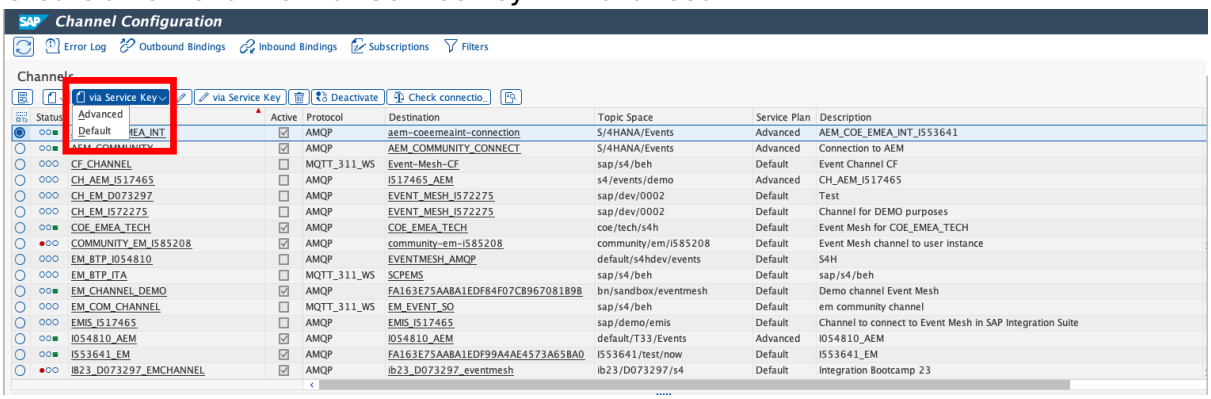
With the authentication configuration for the Advanced Event Mesh let’s create the VMR destination within the S/4. Start by going into the BTP Cockpit of your suaccount with the AEM instance in it

26. Open “Instances and Subscriptions” and select the instance of your AEM
27. Press “View Credentials” and copy the service key

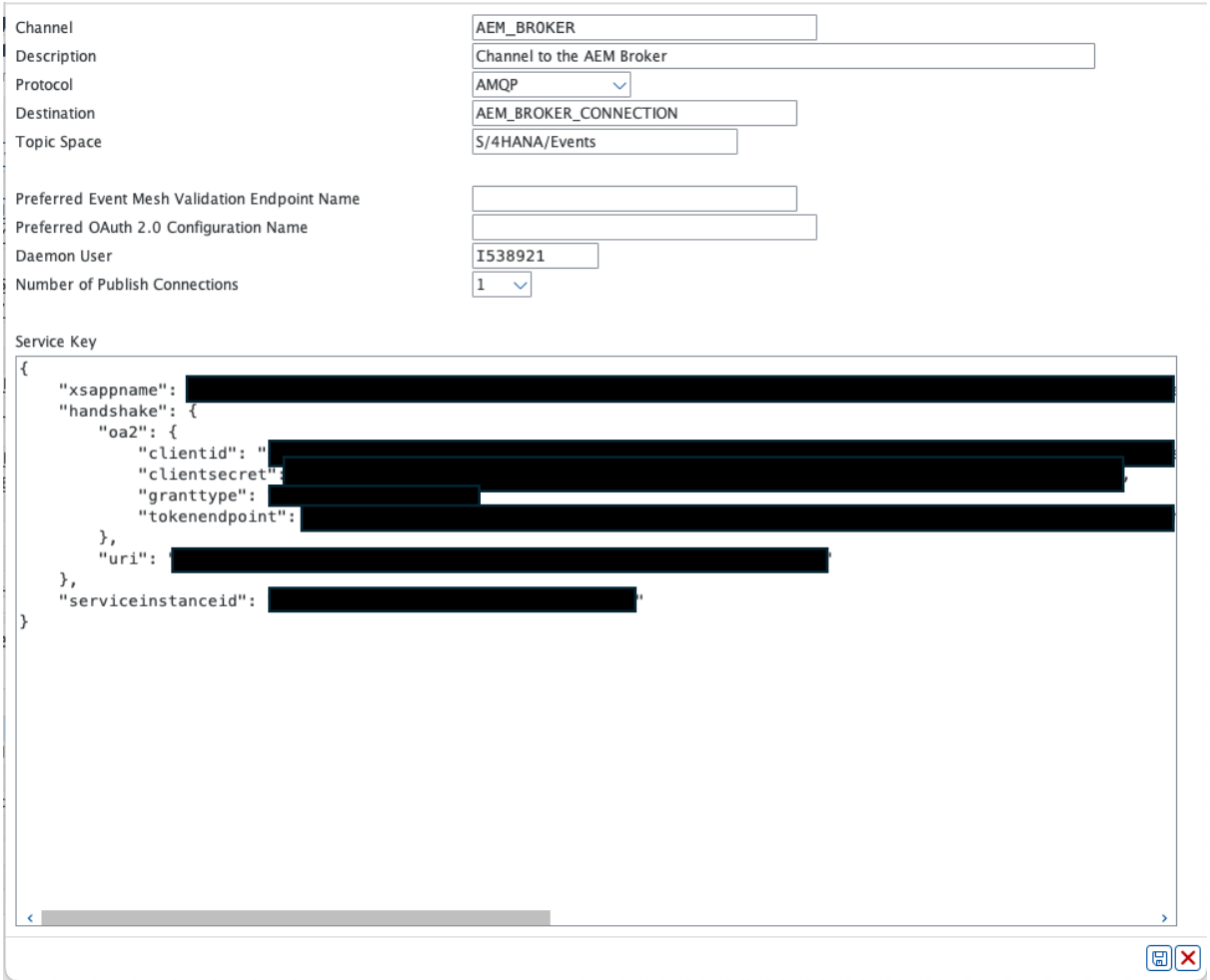


28. Go back to SAPGUI and open transaction */IWXBE/CONFIG*.

29. Create a new channel via “Service Key” > “Advanced”.



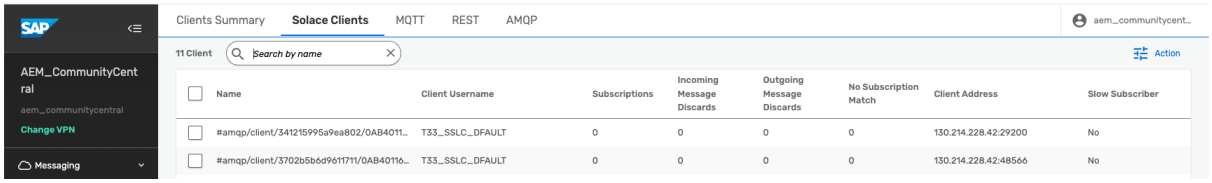
30. Enter the following data and paste the previously copied service key
    Channel: AEM_BROKER
    Description: Channel to the AEM Broker
    Destination: AEM_BROKER_CONNECTION (from step 8)



31. Press save
32. If not already, activate the channel

With that the connection between the AEM and S/4HANA system should be setup. You can verify the setup by checking the AMQP connections within the AEM dashboard. Simply open “Clients” under the “Manage” tab of the event broker. There should be two

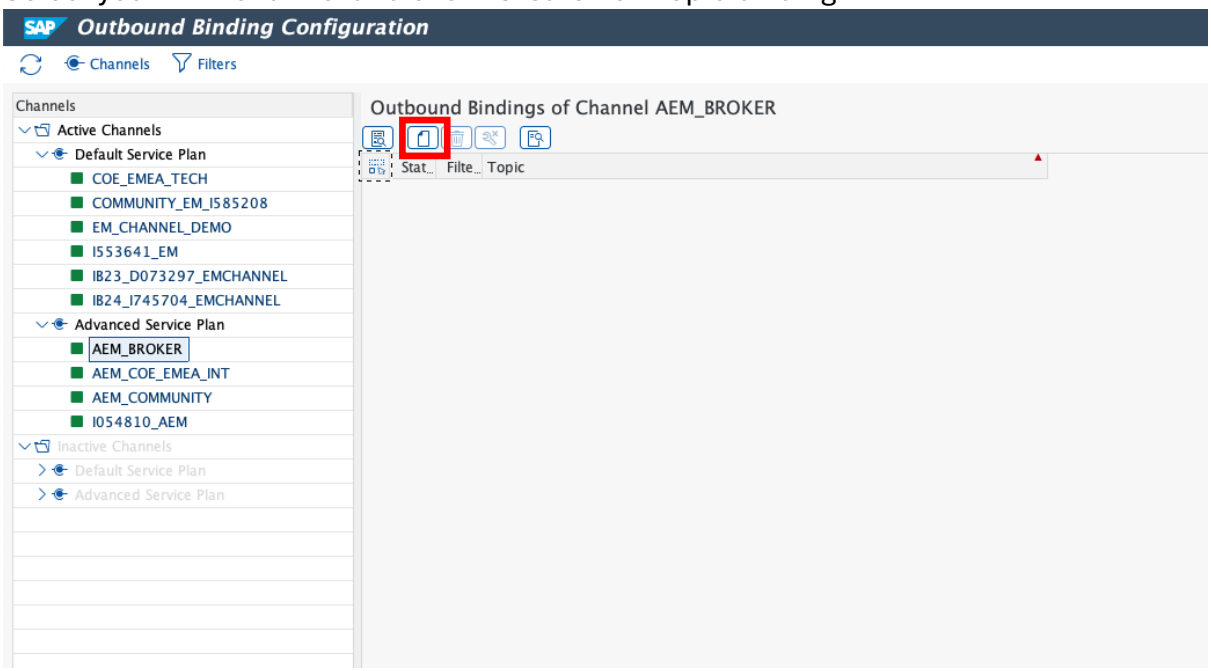
new connections. One for publishing and one for receiving events



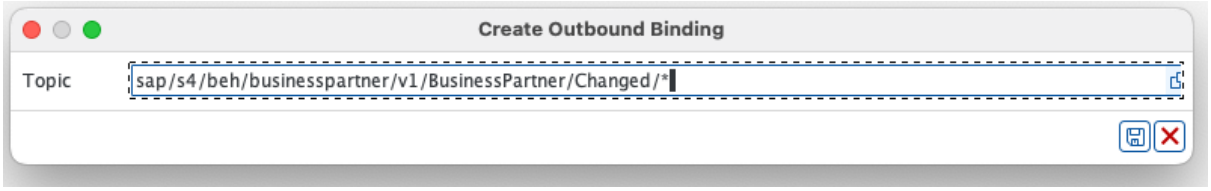
Publish Standard Events from S/4HANA to AEM

To publish a standard event to the AEM you will need to create a topic binding on the channel created in the previous section. In this exercise we will create a binding for the standard Business Partner Changed event.

1. Open transaction */IWXBE/OUTBOUND_CFG*
2. Select your AEM channel and click “Create new topic binding”

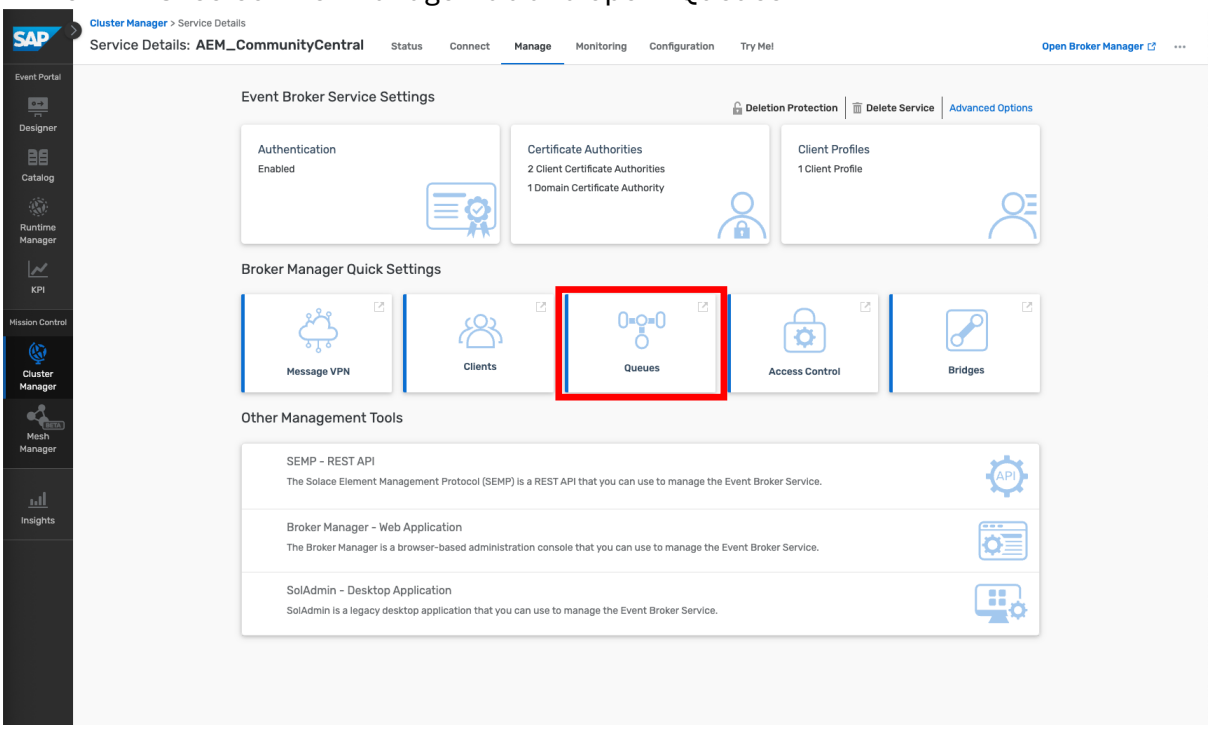


3. Enter the topic “*sap/s4/beh/businesspartner/v1/BusinessPartner/Changed/**”and hit save

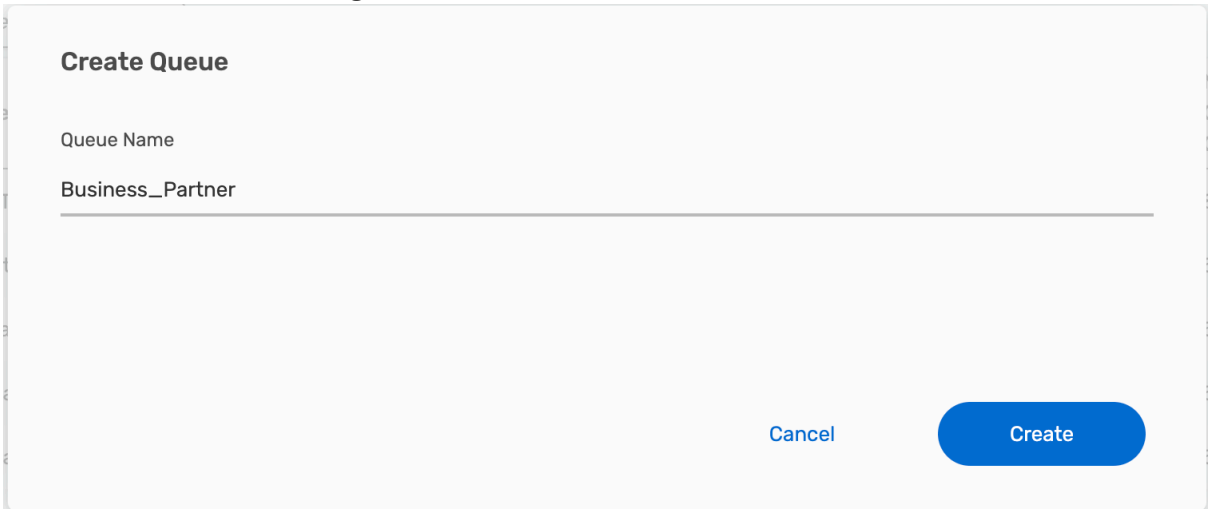


The names of all standard business events can be found in the SAP Business Accelerator Hub. Simply remove the leading “ce/” from the event name for the topic binding.

4. In the AEM UI select the “Manage” tab and open “Queues”

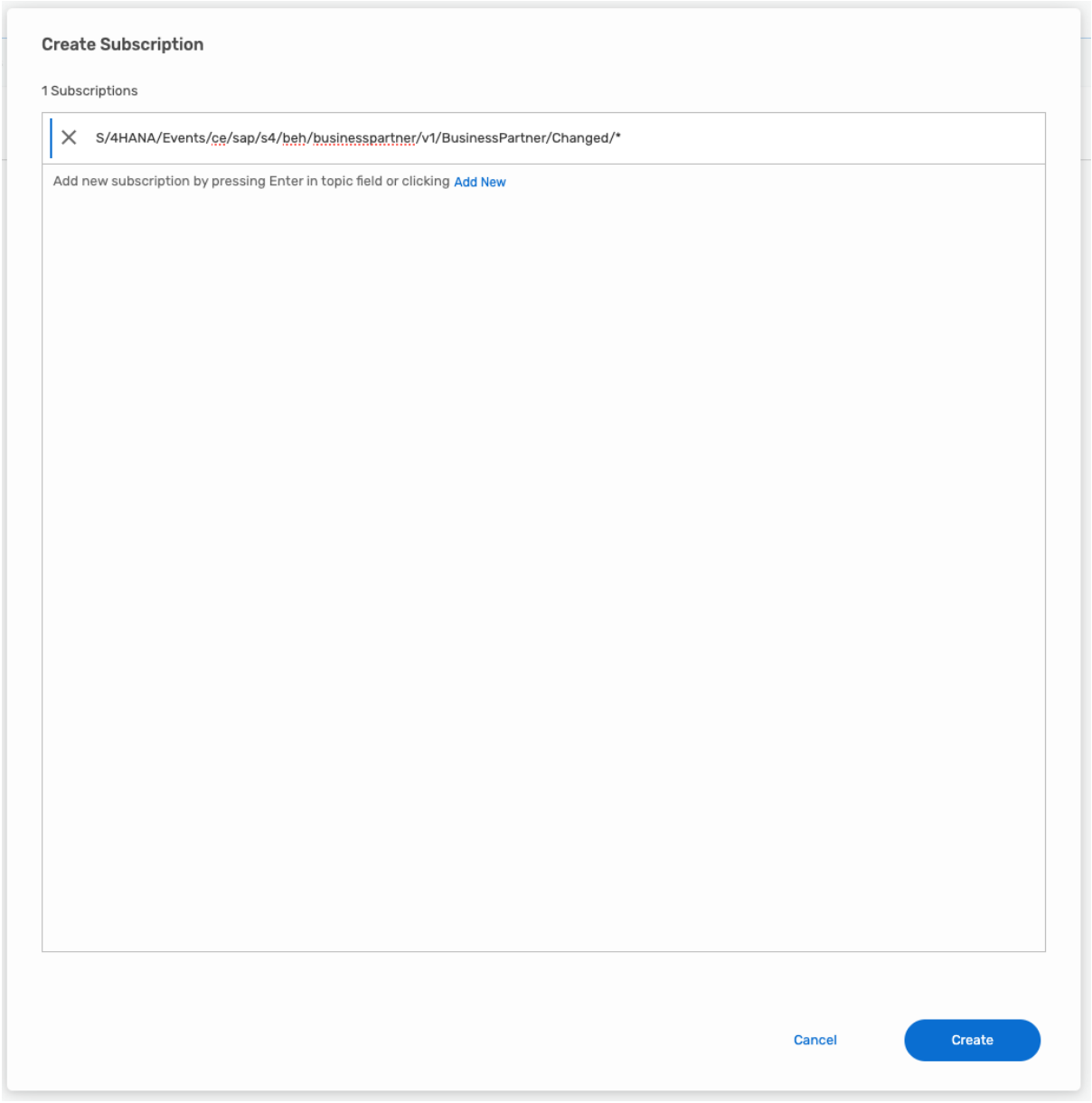


5. Create a new Queue and give it the name “Business_Partner”



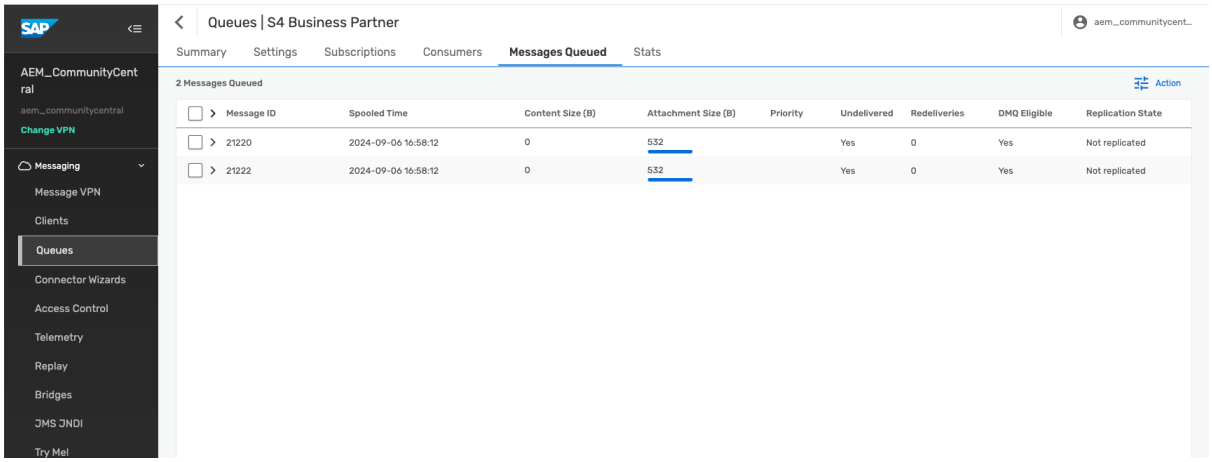
6. Select the queue and navigate to “Subscriptions”.

7. Create a new Subscription with topic "*S/4HANA/Events/ce/sap/s4/beh/businesspartner/v1/BusinessPartner/Changed/**"



*Note: Standard events published by S/4HANA will always include the namespace that was given during channel creation in the topic. By default this is "*S/4HANA/Events*". "/ce" is this time included to indicate that the event is Cloud.Event.io conform.

Making a change to a business partner within transaction *BP* will now trigger an event that is being bound by the created channel. The Enterprise Event Enablement will then perform a handshake with the VMR/validation service to confirm that the AEM broker is a valid SAP broker and afterwards push the event. Since the created queue is subscribed to the topic of business partner changes the event will be queued. This can be verified by looking at the "*Messages Queued*" within the queue.



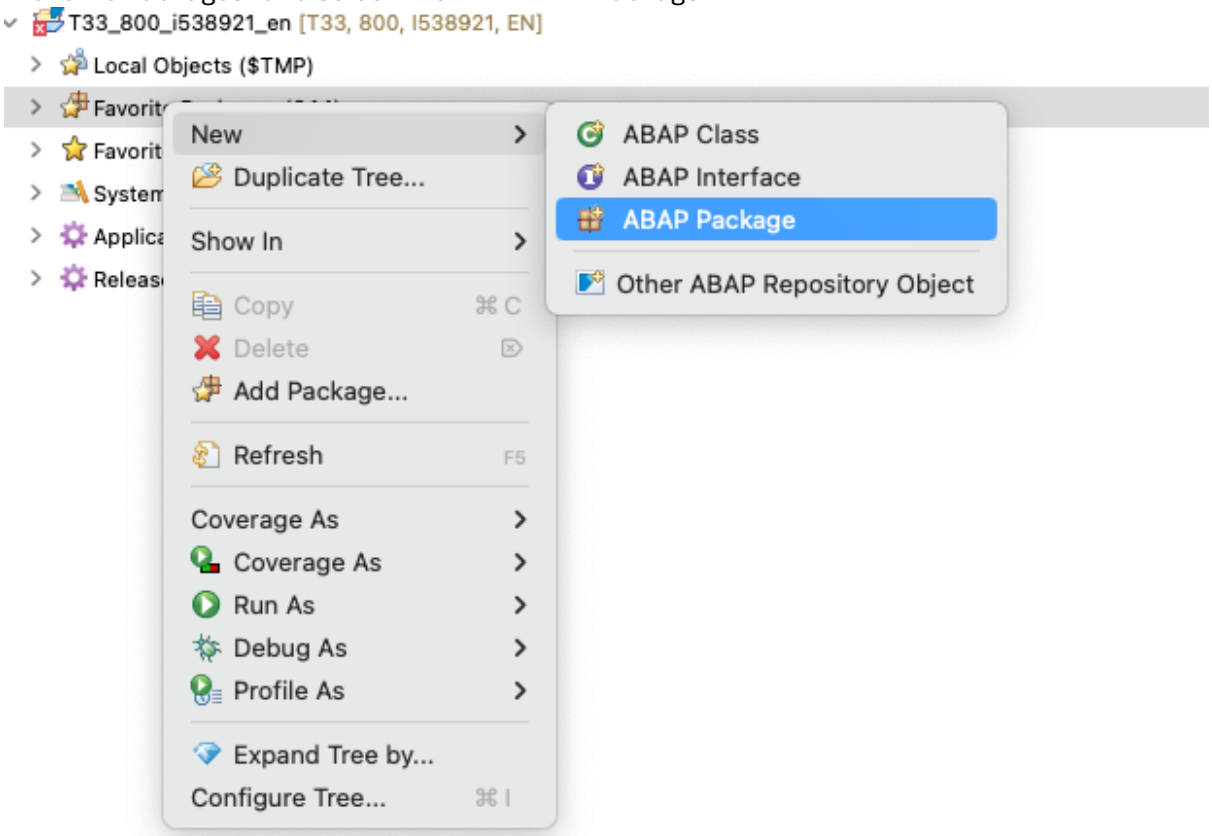
Connect S/4HANA or S/4HANA Cloud with Custom Events

In this next task, we're going to look at creating custom business events from scratch and publishing them to the Advanced Event Mesh. To be precise we'll be creating a new custom online shop application and defining a items ordered event to indicate that a new order has been placed.

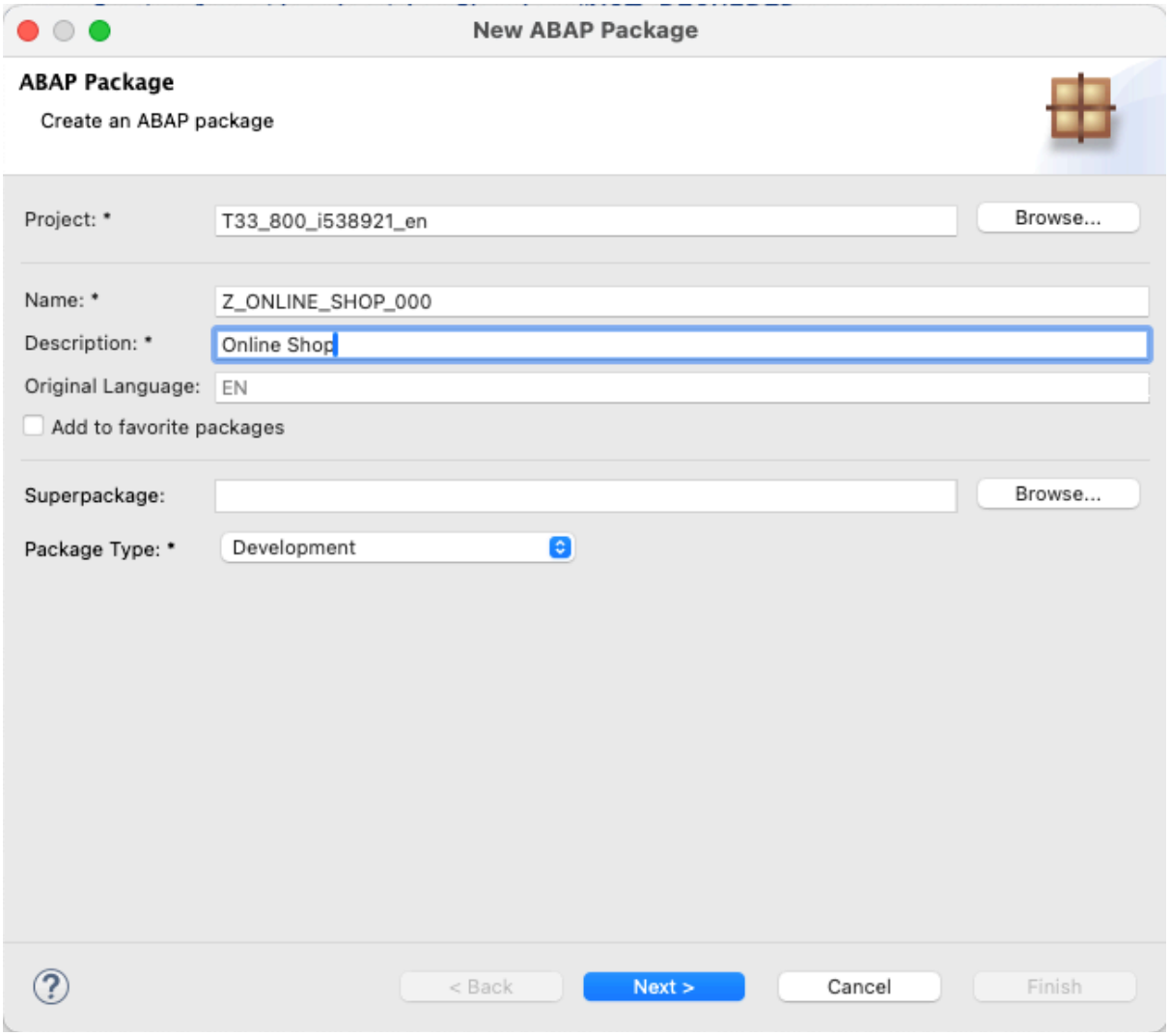
To start lets create the basic online shop application:

1. Create a new package "Z_ONLINE_SHOP_000". The "000" can be replaced by any dedicated number if multiple users are trying to follow this tutorial. Right click on

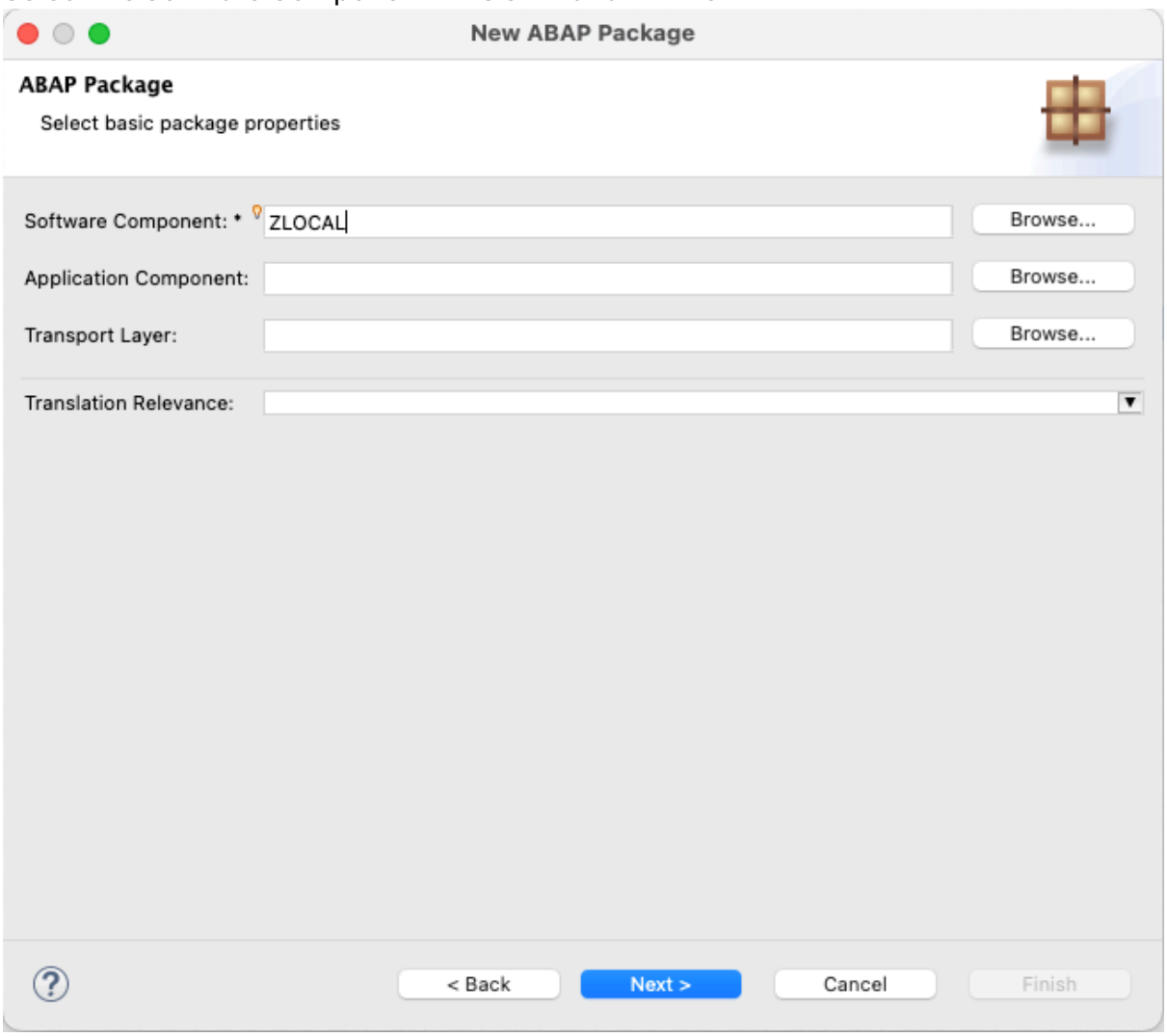
“Favorite Packages” and select “New” > “ABAP Package”



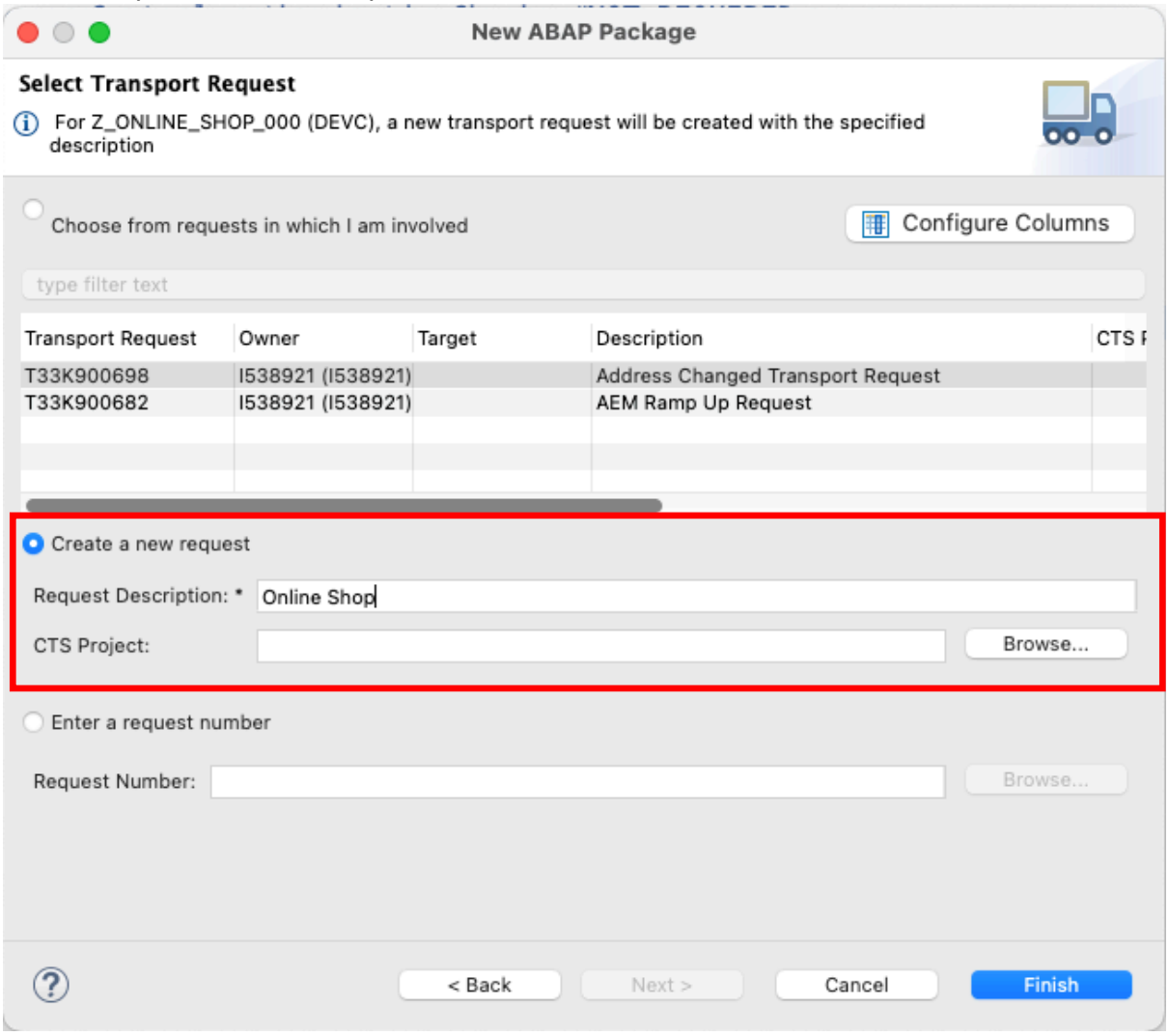
2. Give the name “Z_ONLINE_SHOP_000”, the description “Online Shop” and hit “Next”



3. Select the Software Component “ZLOCAL” and hit “Next”



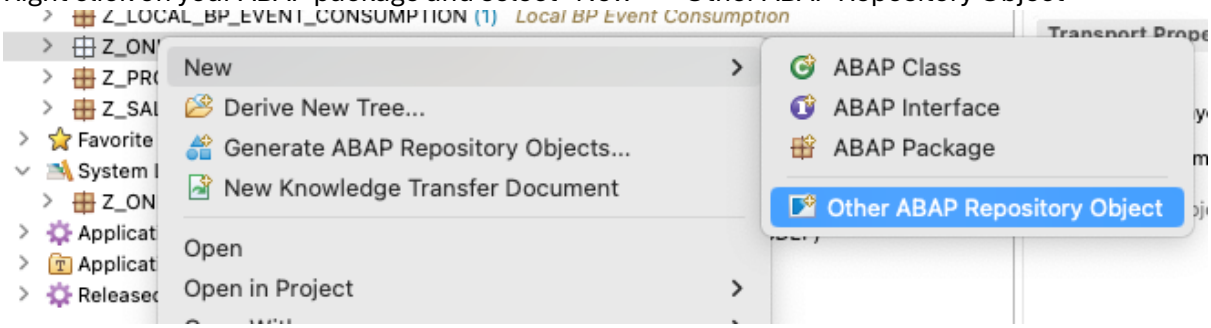
4. Create a new transport request by selecting “Create a new request” and giving the description “Online Shop”



5. Press “Finish”.

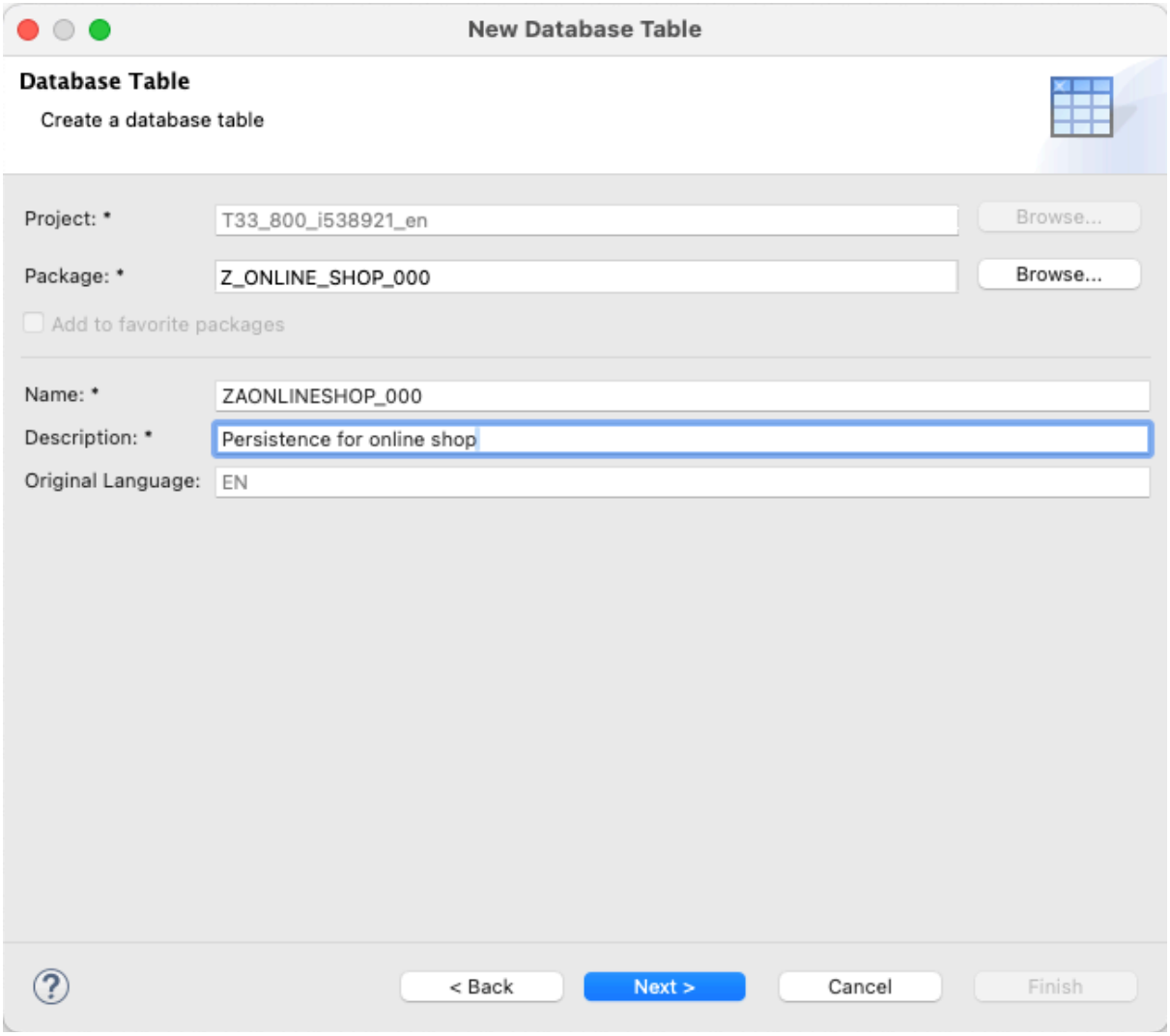
Lets create a new database table

6. Right click on your ABAP package and select “New” > “Other ABAP Repository Object”



7. Search for “Database Table”, select it and click “Next”.

8. Give the name “ZAONLINESHOP_000” and the description “Persistence for online shop”



9. Select “Next”
10. Select the transport request and click “Finish”
11. Replace the default code with the code snippet provided below

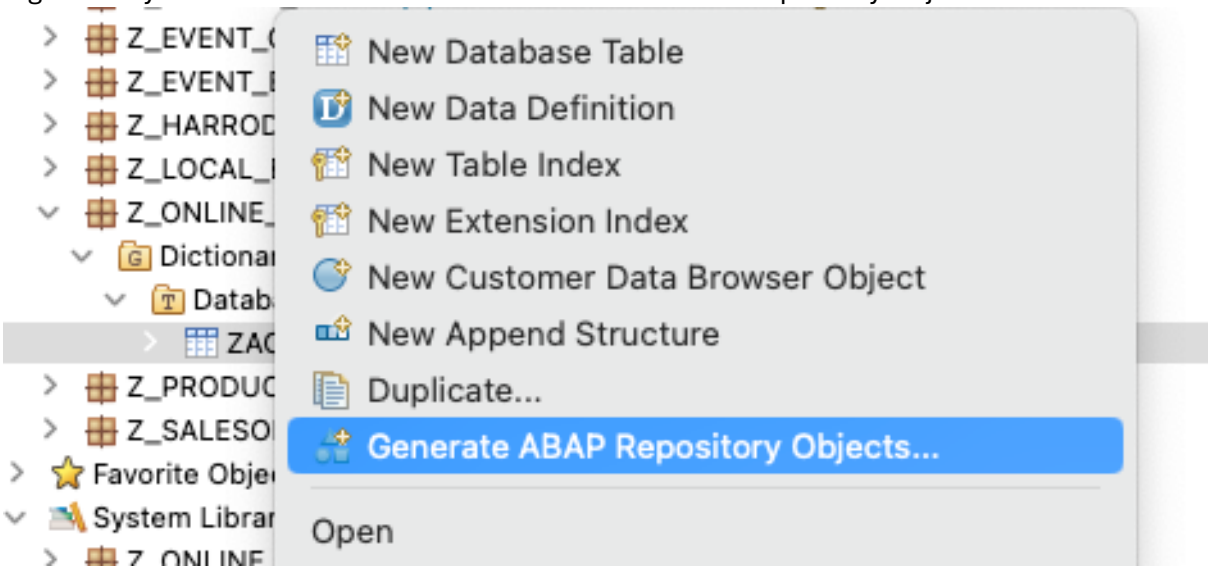
```
@EndUserText.label : 'Persistence for online shop'
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table zaonlineshop_000 {

    key client    : abap.clnt not null;
    key order_uuid : sysuuid_x16 not null;
    order_id      : abap.char(10) not null;
    ordereditem   : abap.char(10) not null;
    deliverydate  : abap.dats;
    creationdate  : abap.dats;
    local_last_changed : abp_locinst_lastchange_tstmpl;
    last_changed      : abp_lastchange_tstmpl;
```

}

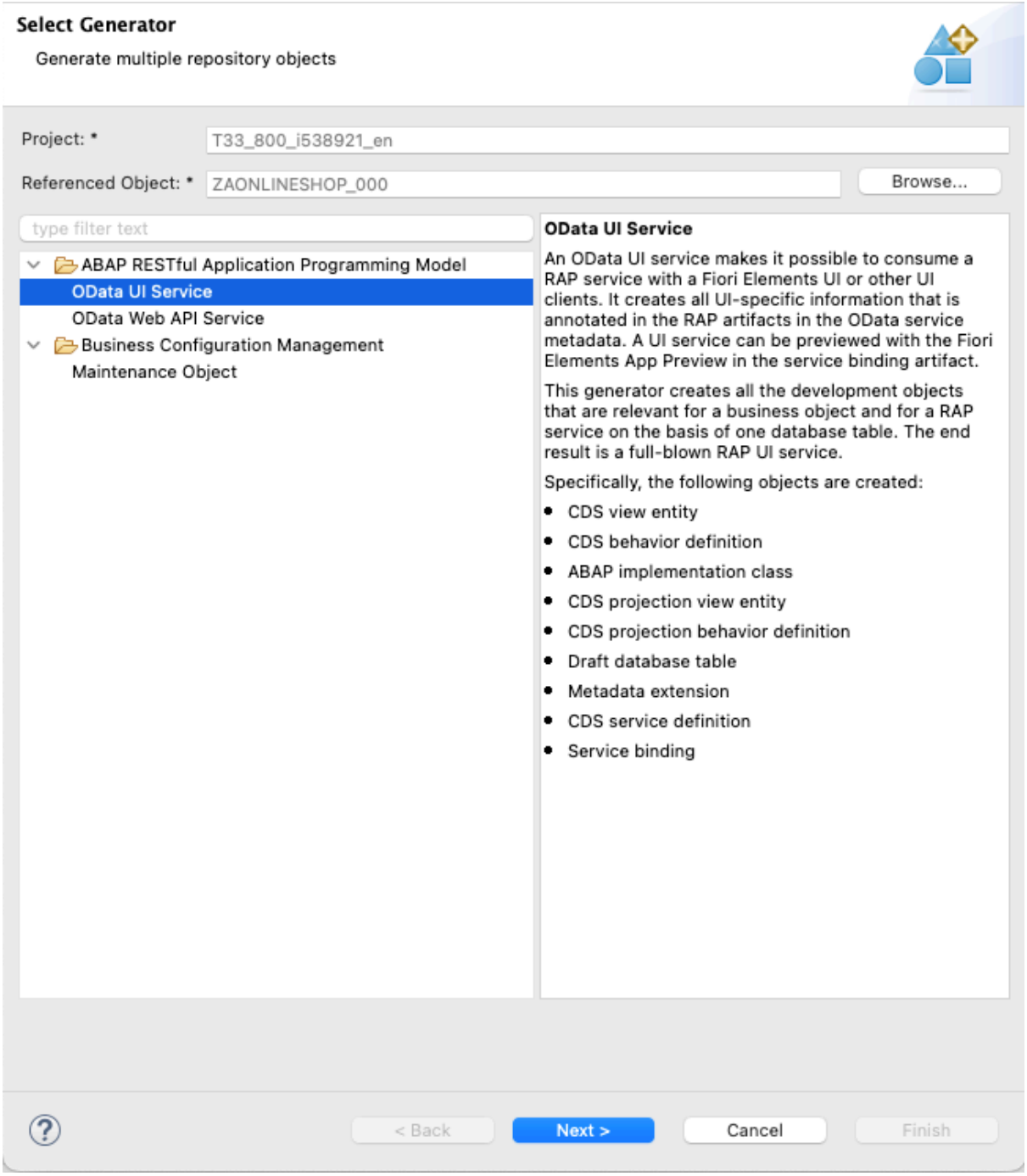
12. Save and activate the changes

Next lets generate the transactional UI services

13. Right-click your database table and select “Generate ABAP Repository Objects.



14. Select “Odata UI Service” and hit “Next”



15. Enter the previously create package “Z_ONLINE_SHOP_000” and hit “Next”
16. Navigate through the wizard tree and maintain the artefact names provided in the table below, and press “Next”.

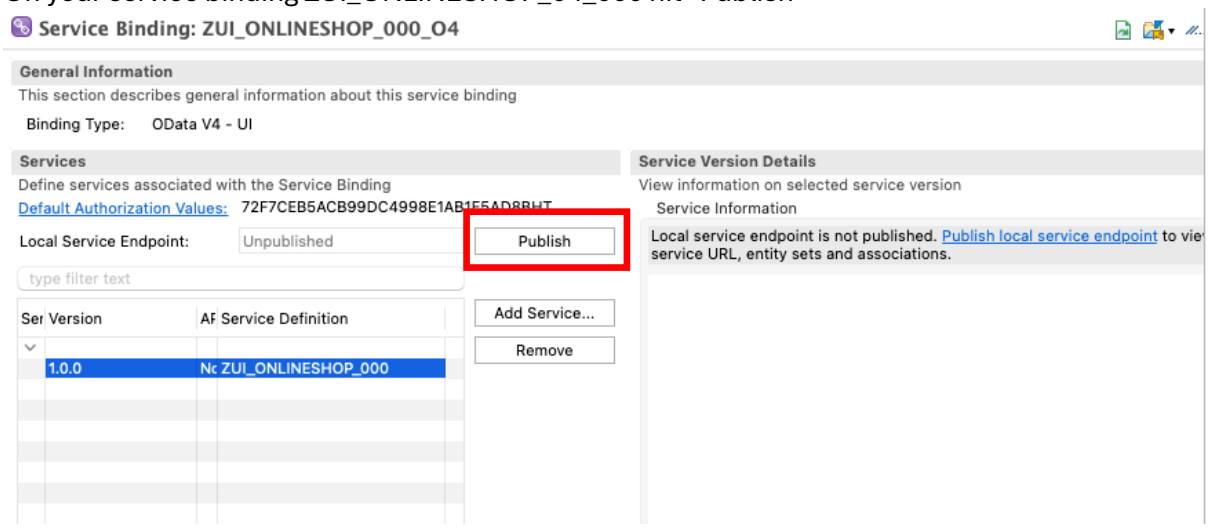
RAP Layer	Artefacts	Artefact Names
Business Object		
	Data Model	Data Definition Name: ZR_ONLINESHOP_000
		Alias Name: OnlineShop
	Behavior	Implementation Class: ZBP_R_ONLINESHOP_000

		Draft Table Name: ZDONLINESHOP_000
Service Projection (BO Projection)		Name: ZC_ONLINESHOP_000
Business Services		
	Service Definition	Name: ZUI_ONLINESHOP_000
	Service Binding	Name: ZUI_ONLINESHOP_04_000
		Binding Type: OData V4 - UI

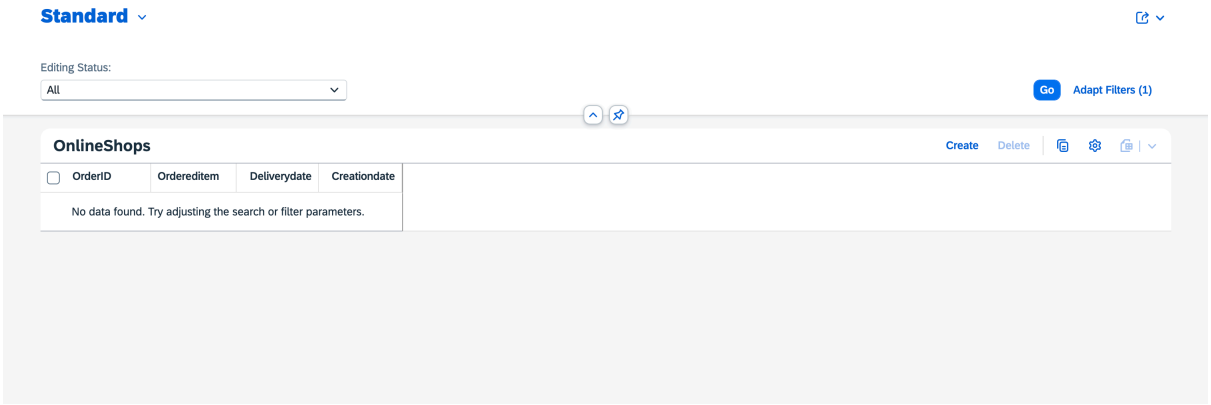
17. Press “Finish”

With the RAP objects being created we can preview the application

18. On your service binding ZUI_ONLINESHOP_04_000 hit “Publish”



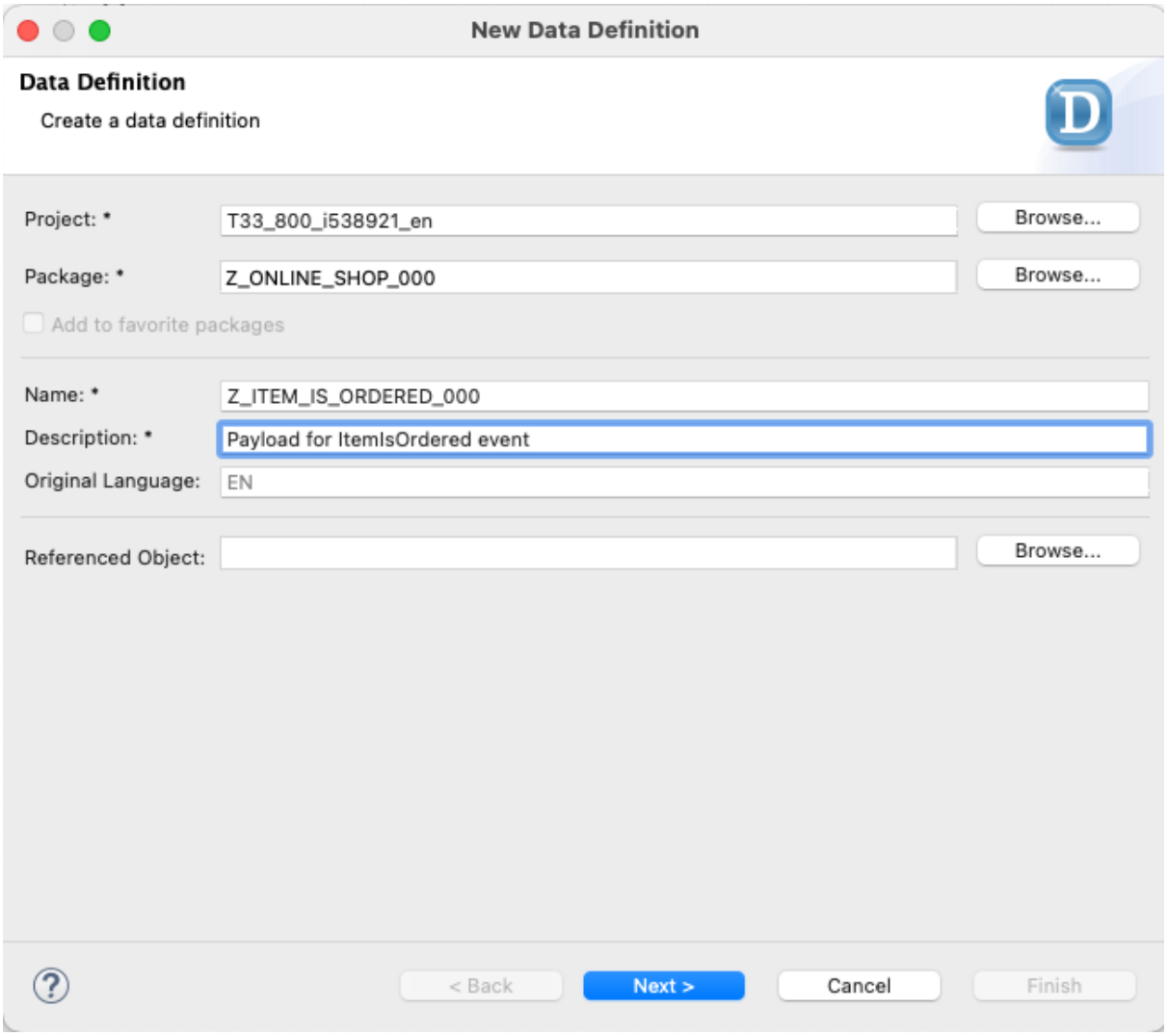
19. Double-click on the entity “OnlineShop” in the “Entity Set and Association” section to open the Fiori elements App Preview.
20. Use your S/4 login credentials to authenticate



With the application itself being created lets define an event “ItemIsOrdered” that is triggered when ever a new order is placed. The event should always contain the name of the ordered item.

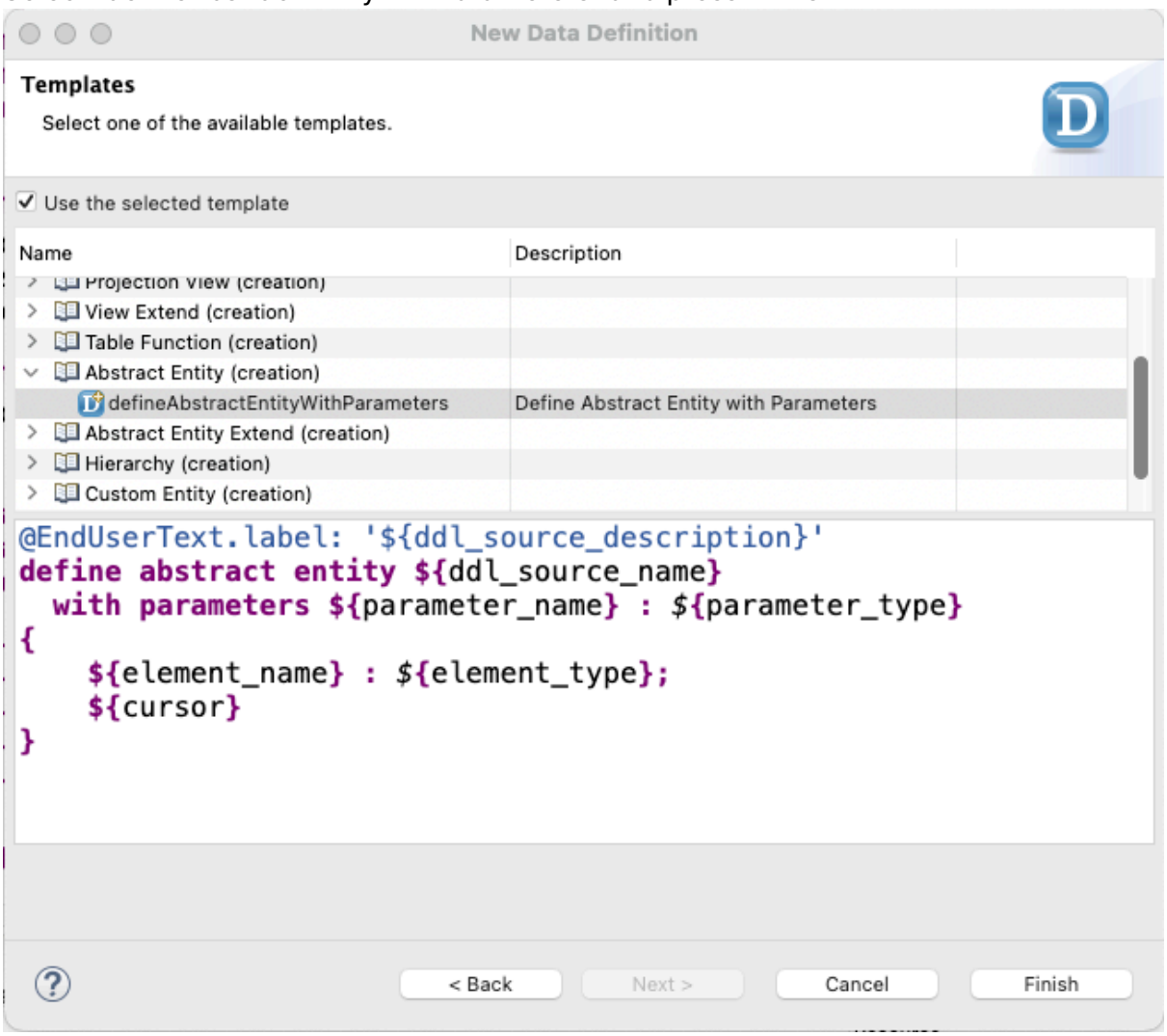
21. Right click your package Z_ONLINE_SHOP_000 and select “New” > “Other ABAP Repository Object”

22. Search for “Data Definition” and select “Next”
23. Give the name “Z_ITEM_IS_ORDERED” and the description “Payload for ItemIsOrdered event” and hit “Next”



24. Select a transport request and press “Next”

25. Select “defineAbstractEntityWithParameters” and press “Finish”



26. Replace the code with the below provided code snippet

```
@EndUserText.label: 'Payload for ItemIsOrdered event'
define root abstract entity Z_ITEM_IS_ORDERED_000
{
 ItemName : abap.char(25);
}
```

27. Save and activate the abstract entity
28. Open the behavior definition ZR_ONLINESHOP_000

29. Enter the line “event ItemIsOrdered parameter Z_ITEM_IS_ORDERED”

*[T33] ZR_ONLINESHOP_000

```
managed implementation in class ZBP_R_ONLINESHOP_000 unique;
strict ( 2 );
with draft;

define behavior for ZR_ONLINESHOP_000 alias OnlineShop
persistent table zaonlineshop_000
draft table ZDONLINESHOP_000
etag master LocalLastChanged
lock master total etag LastChanged
authorization master( global )

{
  field ( readonly )
   OrderUUID,
   LastChanged,
   LocalLastChanged;

  field ( numbering : managed )
   OrderUUID;


  create;
  update;
  delete;

  event ItemIsOrdered parameter Z_ITEM_IS_ORDERED;

  draft action Edit;
  draft action Activate optimized;
  draft action Discard;
  draft action Resume;
  draft determine action Prepare;

```

With that the data event is defined and can be used. So lets trigger it when ever a new order is created and send it to the AEM.

30. Add the keywords “with additional save” to the first line of the behavior definition ZR_ONLINESHOP_000

```
[T33] ZR_ONLINESHOP_000
 1 managed with additional save implementation in class ZBP_R_ONLINESHOP_000 unique;
 2 strict ( 2 );
 3 with draft;
 4
 5 define behavior for ZR_ONLINESHOP_000 alias OnlineShop
 6 persistent table zaonlineshop_000
 7 draft table ZDONLINESHOP_000
 8 etag master LocalLastChanged
 9 lock master total etag LastChanged
10 authorization master( global )
11
12 {
13   field ( readonly )
14    OrderUUID,
15    LastChanged,
16    LocalLastChanged;
17
18   field ( numbering : managed )
19    OrderUUID;
20
21
22   create;
23   update;
24   delete;
25
26   event ItemIsOrdered parameter Z_ITEM_IS_ORDERED_000;
27
28   draft action Edit;
29   draft action Activate optimized;
30   draft action Discard;
31   draft action Resume;
32   draft determine action Prepare;
33
```

31. Save and activate the behavior definition
32. Place your coursor on the keyword additional (where the warning is displayed) and press Ctrl + 1 and double click the provided quick assist. This should add the save_modified method to the local implementation class.

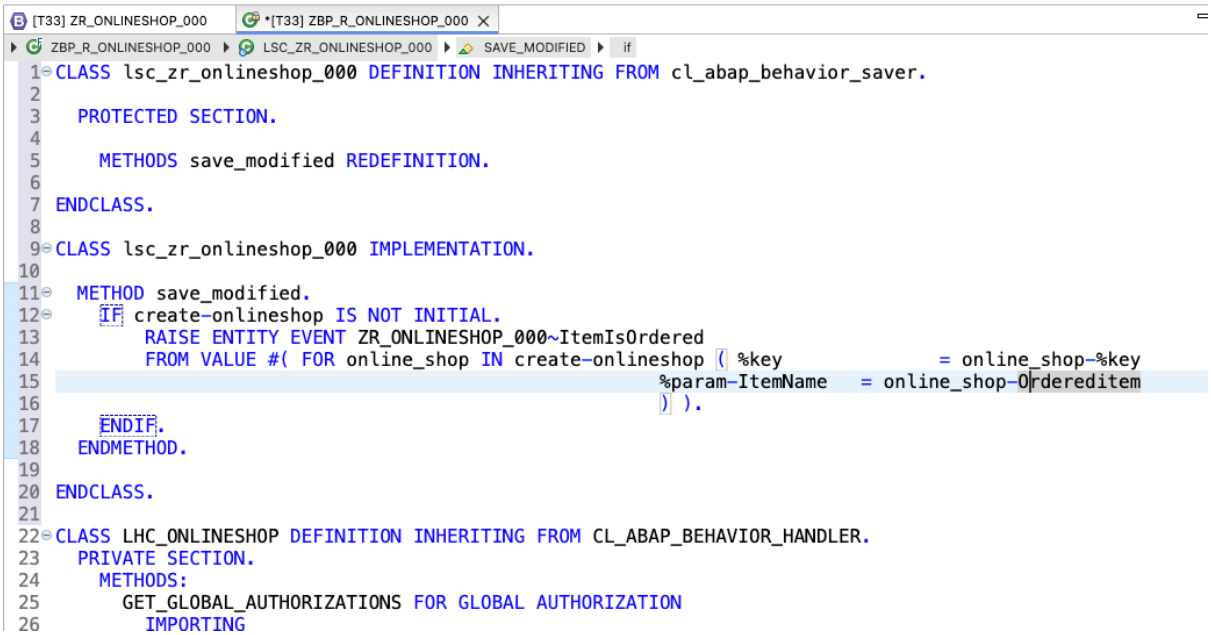
```
[T33] ZR_ONLINESHOP_000   [T33] ZBP_R_ONLINESHOP_000
 1 managed with additional save implementation in class ZBP_R_ONLINESHOP_000 unique;
 2 strict ( 2 );
 3 with draft;
 4
 5 define behavior for
 6 persistent table zao
 7 draft table ZDONLINE
 8 etag master LocalLas
 9 lock master total et
10 authorization master
11
12 {
13   field ( readonly )
14    OrderUUID,
15    LastChanged,
16    LocalLastChanged;
17
18   field ( numbering : managed )

Add required method save_modified in new local saver class
Add missing method save_modified in new local saver class inside global class zbp_r_onlineshop_000.
Press 'Ctrl+Shift+1' to show in Quick Assist View
Press 'Tab' from proposal table or click for foc
```

33. Within the local types of the ABAP Class ZBP_R_ONLINESHOP_000 replace the implementation of the method “save_modified” with the following snippet

```
METHOD save_modified.
  IF create-onlineshop IS NOT INITIAL.
```

```
    RAISE ENTITY EVENT ZR_ONLINESHOP_000~ItemIsOrdered
    FROM VALUE #( FOR online_shop IN create-onlineshop ( %key          = online_shop-
%key
                                %param-ItemName   = online_shop-Ordereditem
                                ) ).
  ENDIF.
ENDMETHOD.
```

34. The code should look something like this

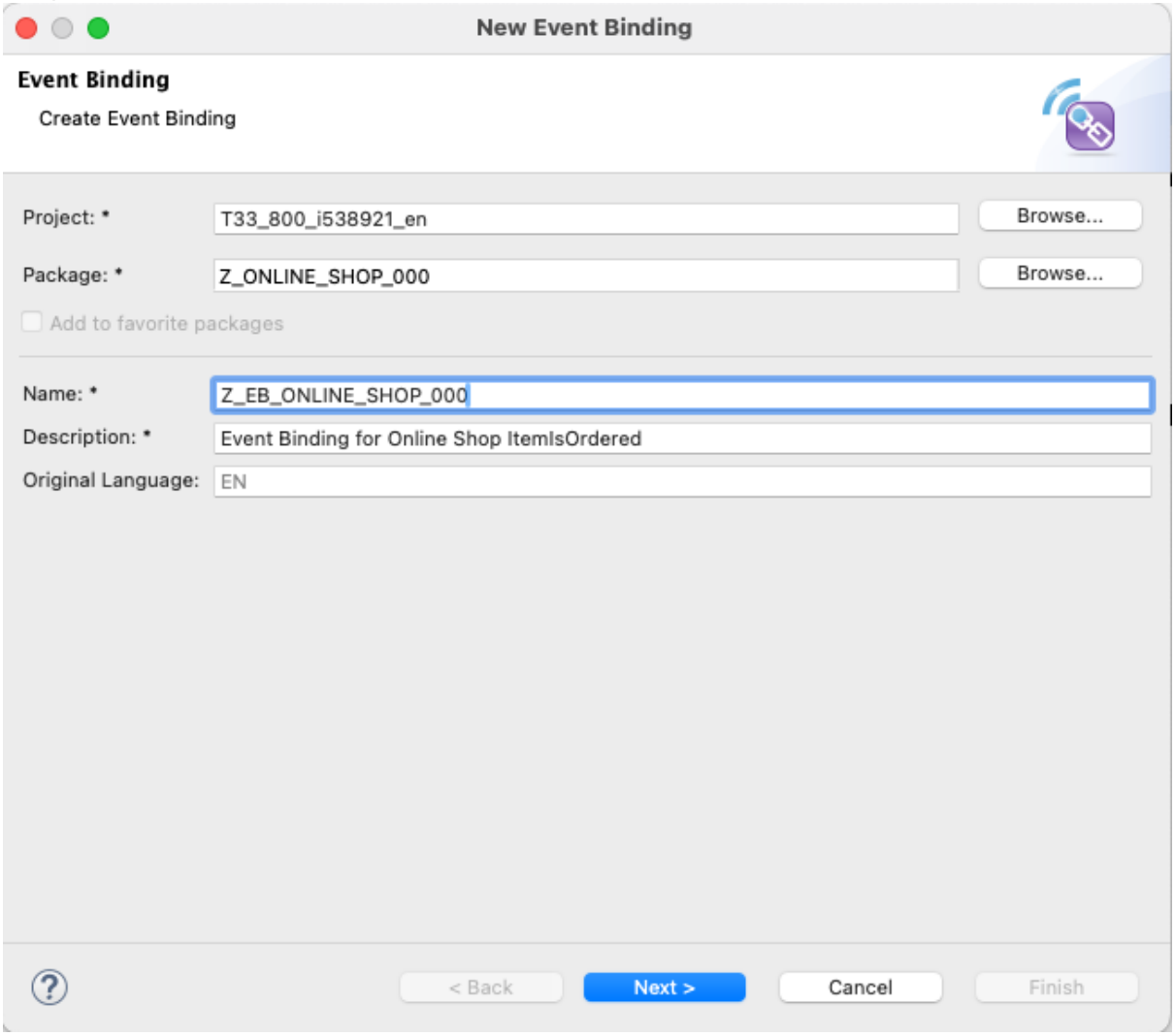


35. Save and activate

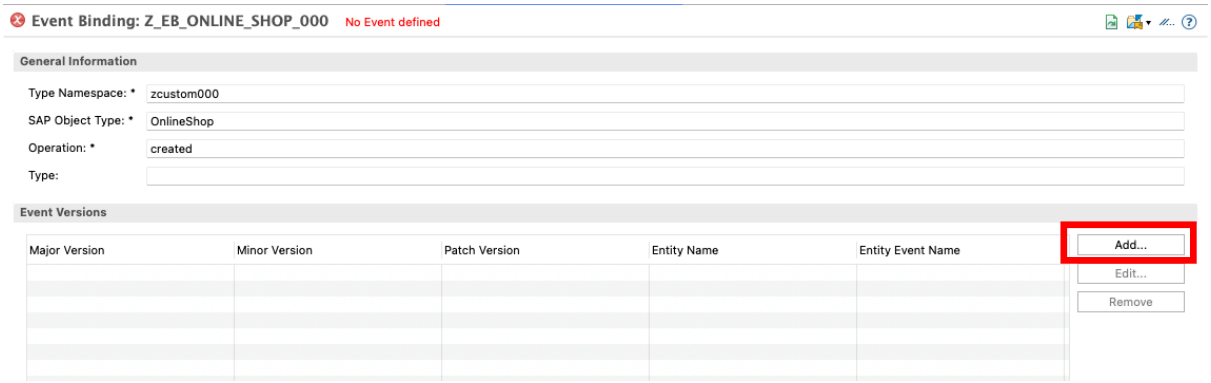
With that the event is triggered when ever a new order is placed. However it is not yet send to the Advanced Event Mesh. For that we need to create an event binding and a topic binding on the AEM channel.

36. Right click your ABAP package and select “New” > “Other ABAP Repository Object”
37. Search for “Event Binding”, select it and press “Next”

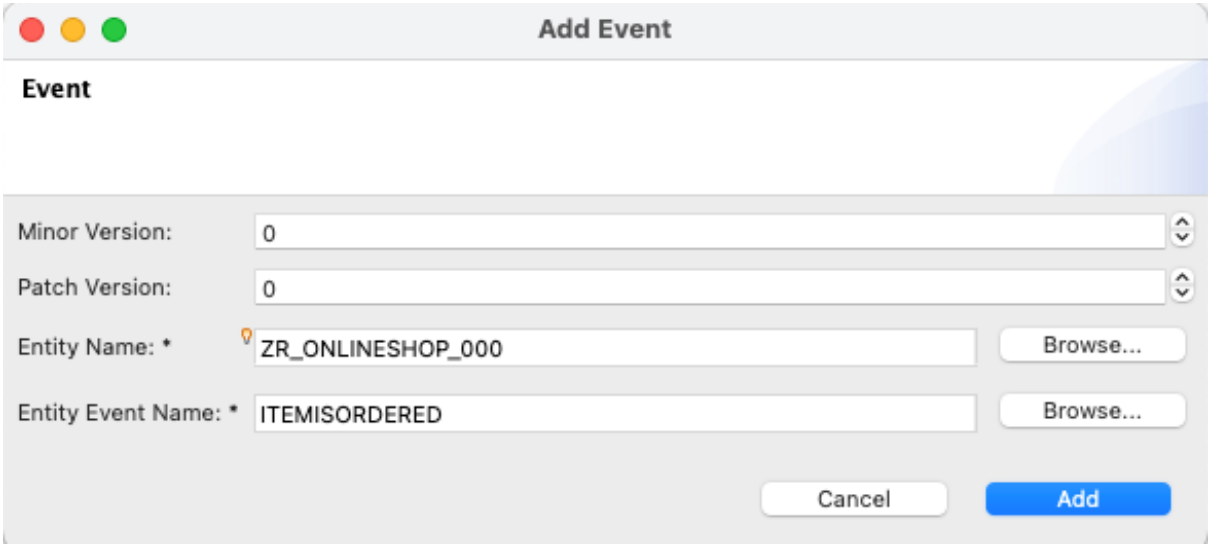
38. Give the name “Z_EB_ONLINE_SHOP” and the description “Event Binding for Online Shop ItemIsOrdered”



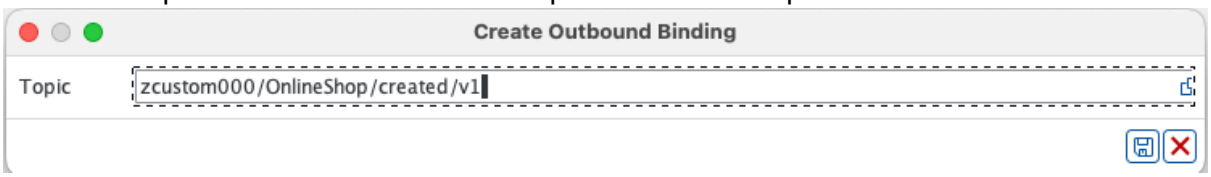
39. Select your transport request and press “Finish”
40. In the event binding you can define the topic und which the event is later published. Enter the following values:
    Type Namespace: zcustom000
    SAP Object Type: OnlineShop
    Operation: created
41. Press “Add..”



42. Enter the Entity Name “ZR_ONLINESHOP_000” and the Entity Event Name “ITEMISORDERED” and press “Add”

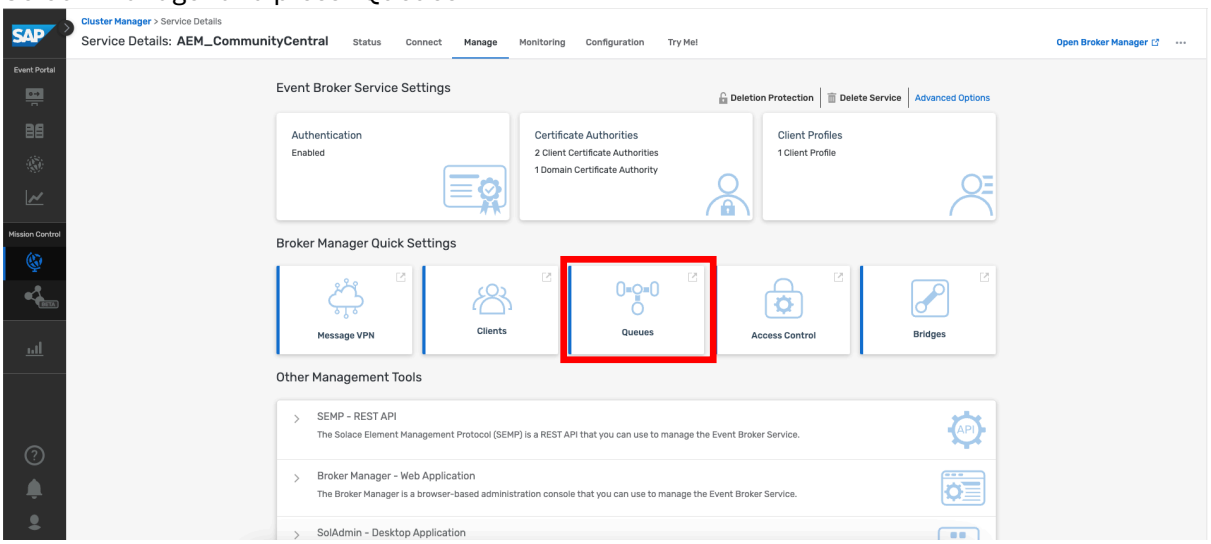


43. Activate the event binding.
44. Next go into transaction “/IWXBE/OUTBOUND_CFG” and select the channel to your AEM that has been previously set up
45. Press “Create new topic binding”
46. Enter the topic “zcustom000/OnlineShop/created/*” and press save



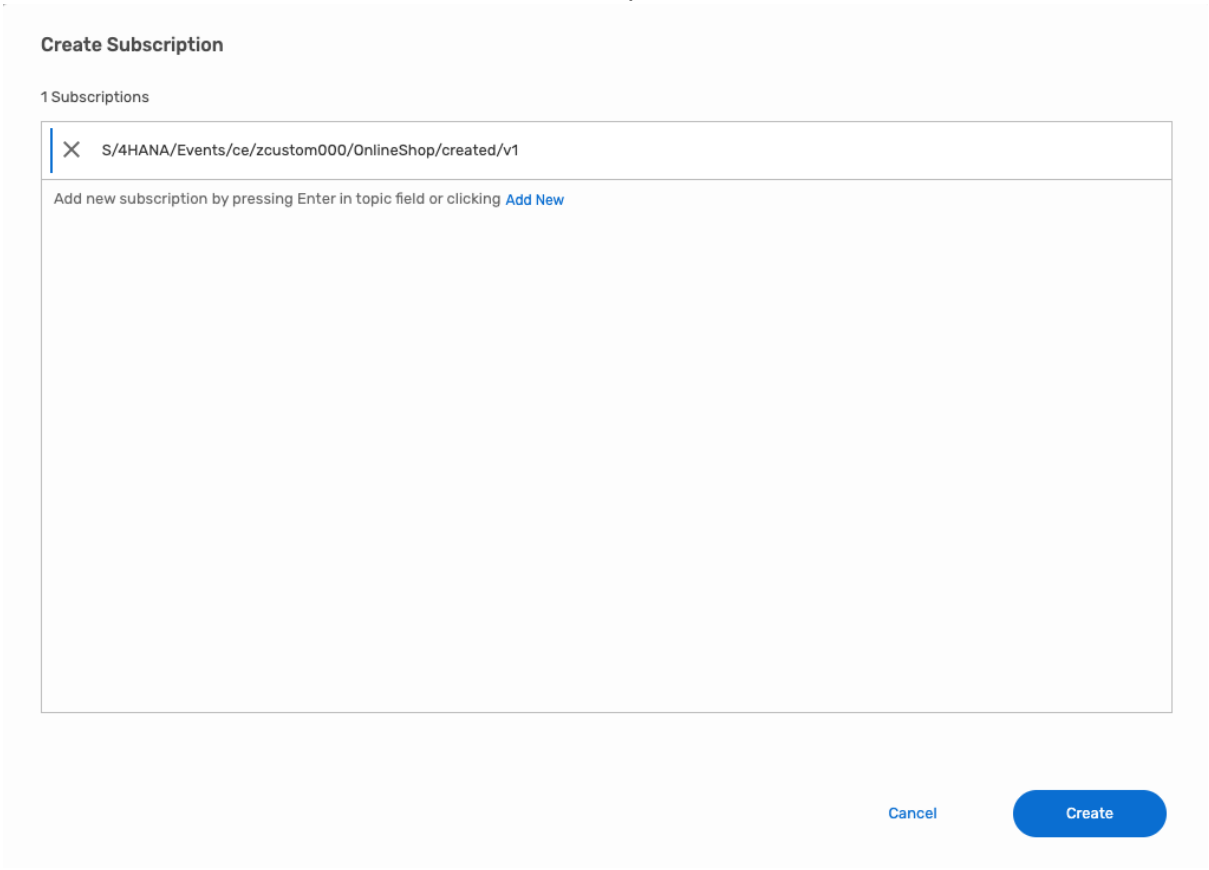
With that the event is automatically send to the AEM where it can be consumed.

47. Go into the AEM UI
48. Under Cluster Manager select your broker
49. Select “Manage” and press “Queues”



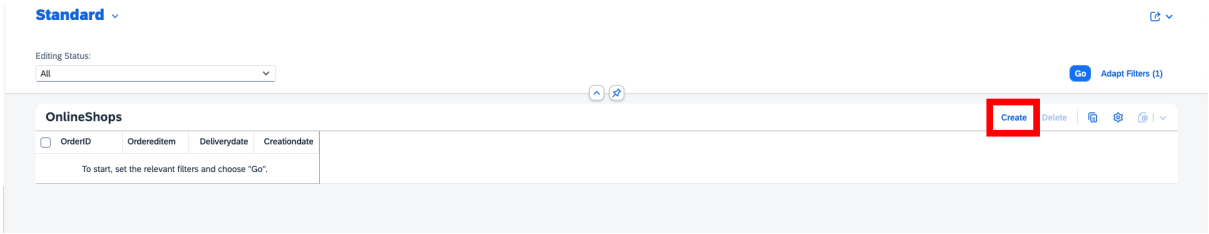
50. Create a new queue and give it the name “Online Shop”
51. Go into the queue and navigate to “Subscriptions”

52. Add a new subscription with the name “S/4HANA/Events/ce/zcustom000/OnlineShop/created/v1”

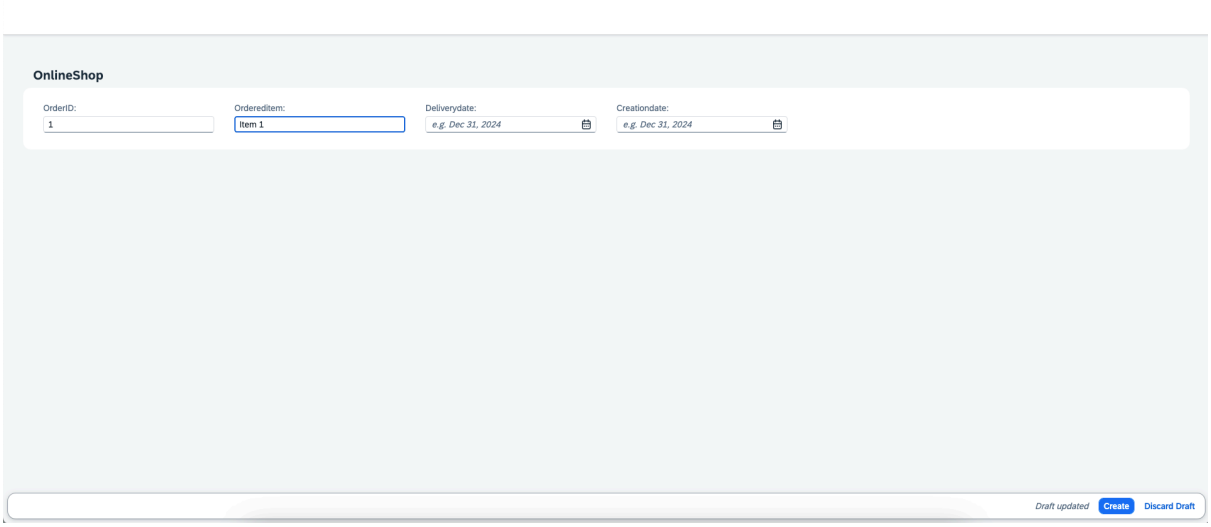


With that the event ItemIsOrdered will be triggered when ever a new order is created and send to the AEM where it is consumed by the created queue.
Let’s test that.

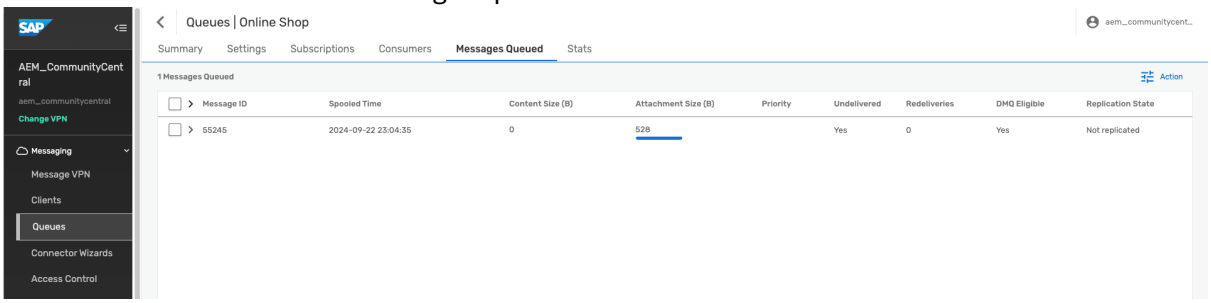
53. Within eclipse open the service binding “ZUI_ONLINESHOP_04_000” and double click the entity “OnlineShop” within the Entity Set and Association section. This should open the Fiori Preview.
54. If required log in with your S/4 credentials
55. Press create in the Fiori application



56. Give the OrderID “1” and the Ordereditem “Item 1” and press “Create”



57. With that there should be a messaged queued within the AEM



Creating custom events with deep payloads

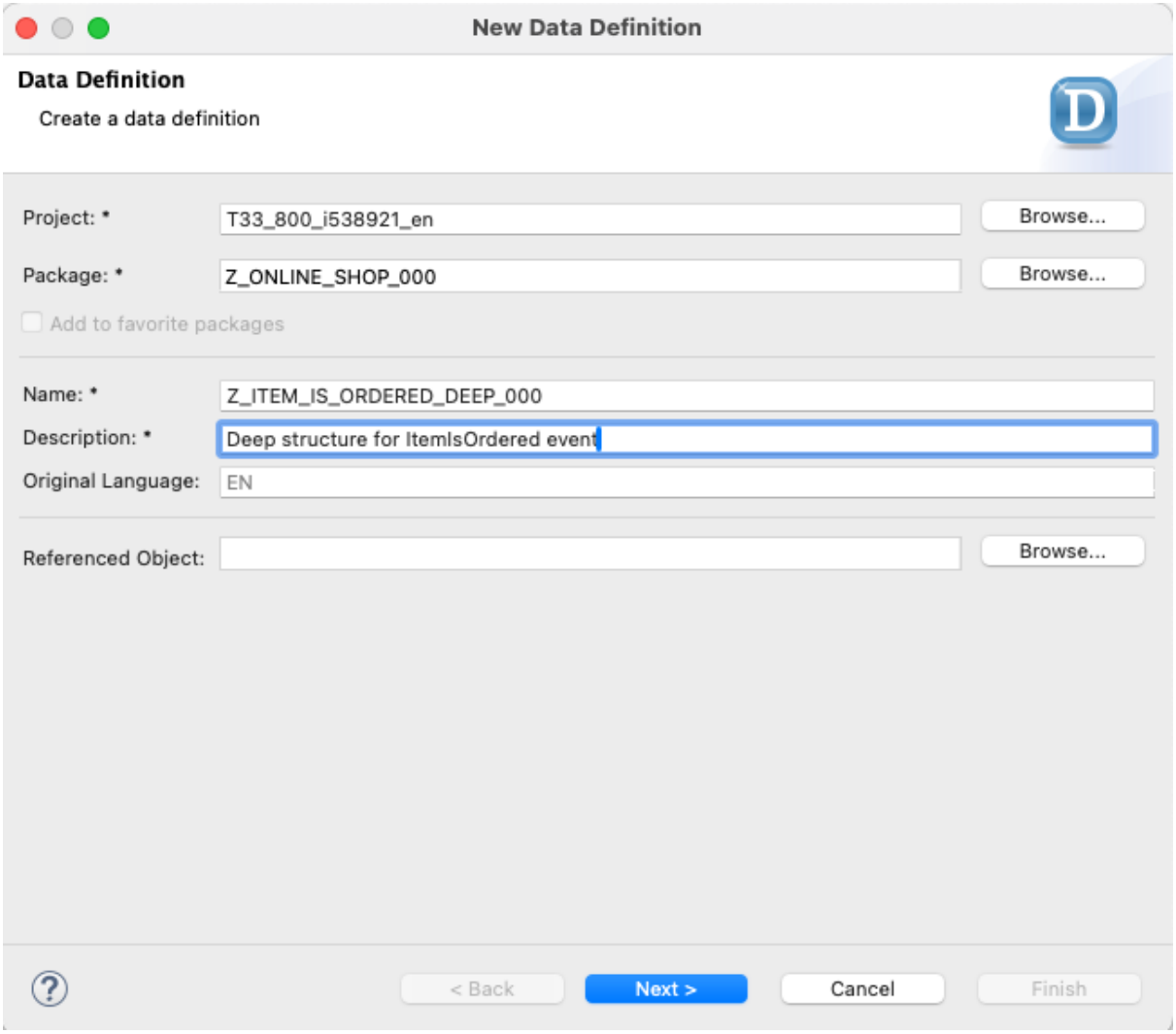
In many cases only having a flat structure as payload is not sufficient. For these cases RAP provides the possibility to have events with deep structures as payloads.
Let’s adjust the “ItemIsOrdered” event to use a deep payload structure:

1. Go into “ZR_ONLINESHOP_000” and add the “deep” keyword to the event definition (ignore the error for now

```
event ItemIsOrdered deep parameter Z_ITEM_IS_ORDERED_000;
```

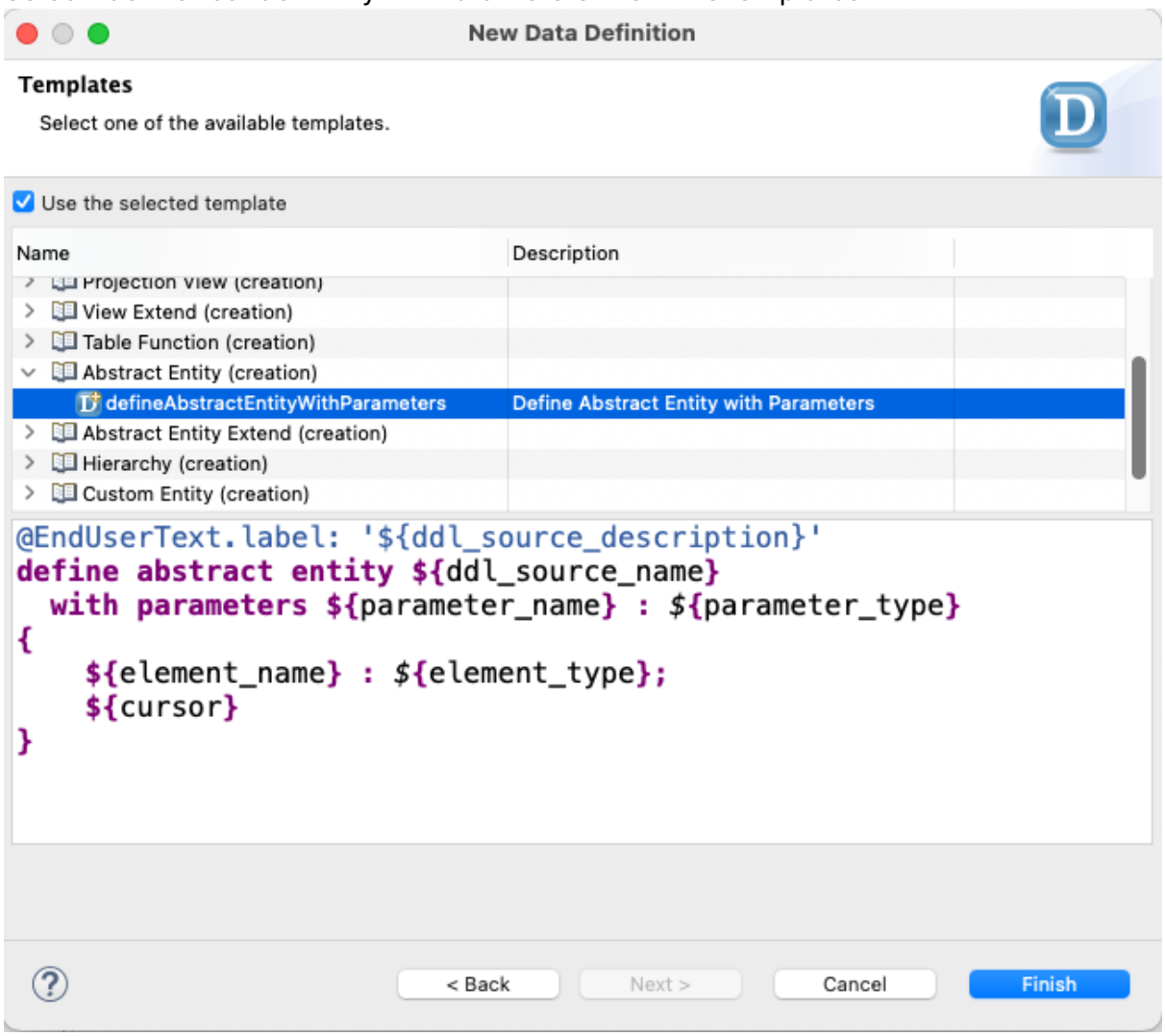
2. Right click your package “Z_ONLINE_SHOP_000” and create a new data definition

3. Give the name “Z_ITEM_IS_ORDERED_DEEP_000” and the description “Deep structure for ItemIsOrdered event”



4. Hit next, select a transport request and hit next once again.

5. Select “defineAbstractEntityWithParameters” from the templates



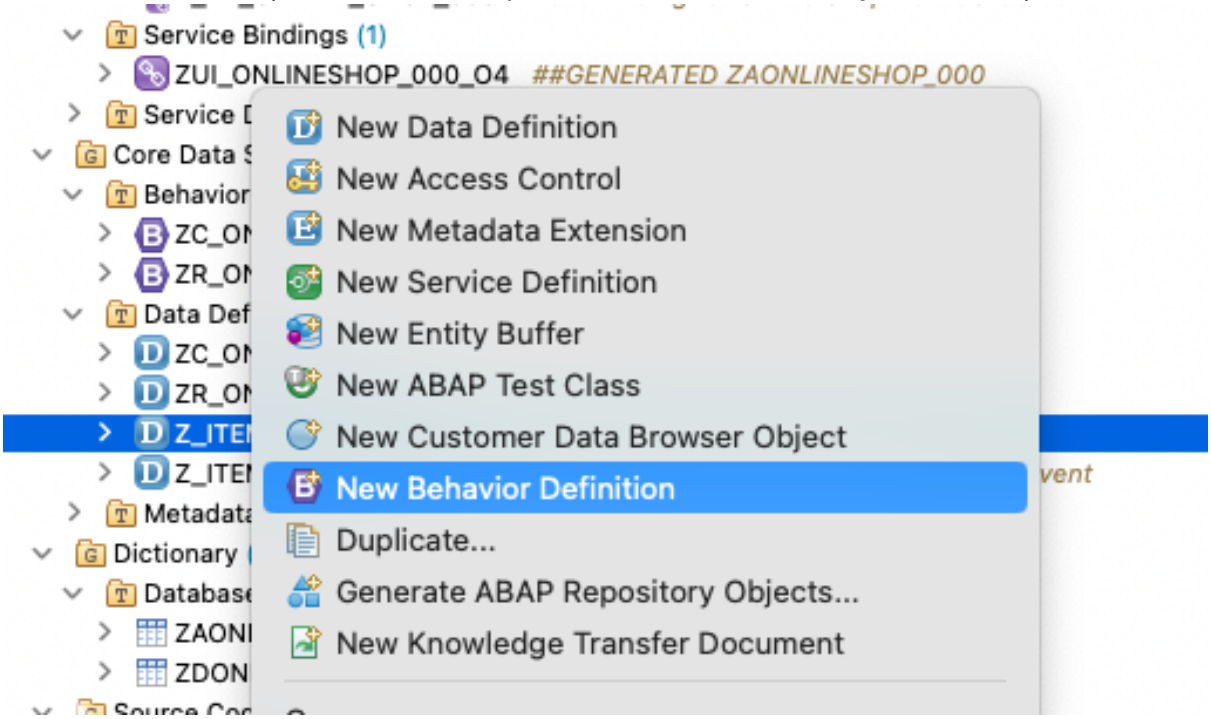
6. Adjust the data definition:

```
@EndUserText.label: 'Deep structure for ItemIsOrdered event'
define abstract entity Z_ITEM_IS_ORDERED_DEEP_000
{
  Deliverydate : abap.dats;
  Creationdate : abap.dats;
  _Item : association to parent Z_ITEM_IS_ORDERED_000;
}
```

7. Activate the data definition
8. Go into data definition “Z_ITEM_IS_ORDERED_000”
9. Add an association to “Z_ITEM_IS_ORDERED_DEEP_000”

```
1 @EndUserText.label: 'Payload for ItemIsOrdered event'
2 define root abstract entity Z_ITEM_IS_ORDERED_000
3 {
4   ItemName : abap.char(25);
5   dates : composition [0..1] of Z_ITEM_IS_ORDERED_DEEP_000;
6 }
7
```

10. Activate the data definition
11. Right click your data definition “Z_ITEM_IS_ORDERED_000” and create a new behavior definition (name and description should automatically be derived)



12. Adjust the behavior definition to follow this structure:

```
abstract;
with hierarchy;

define behavior for Z_ITEM_IS_ORDERED_000 //alias <alias_name>
{
}

define behavior for Z_ITEM_IS_ORDERED_DEEP_000
{
}
```

13. Activate the behavior definition
14. Go into “ZR_ONLINESHOP_000” and activate it

With that the event has been defined to have a deep structure as its payload.
To fill that deep structure lets adjust the “RAISE” statement.

15. Go into ABAP class “ZBP_R_ONLINESHOP_000”
16. Adjust the implementation of the “save_modified” method

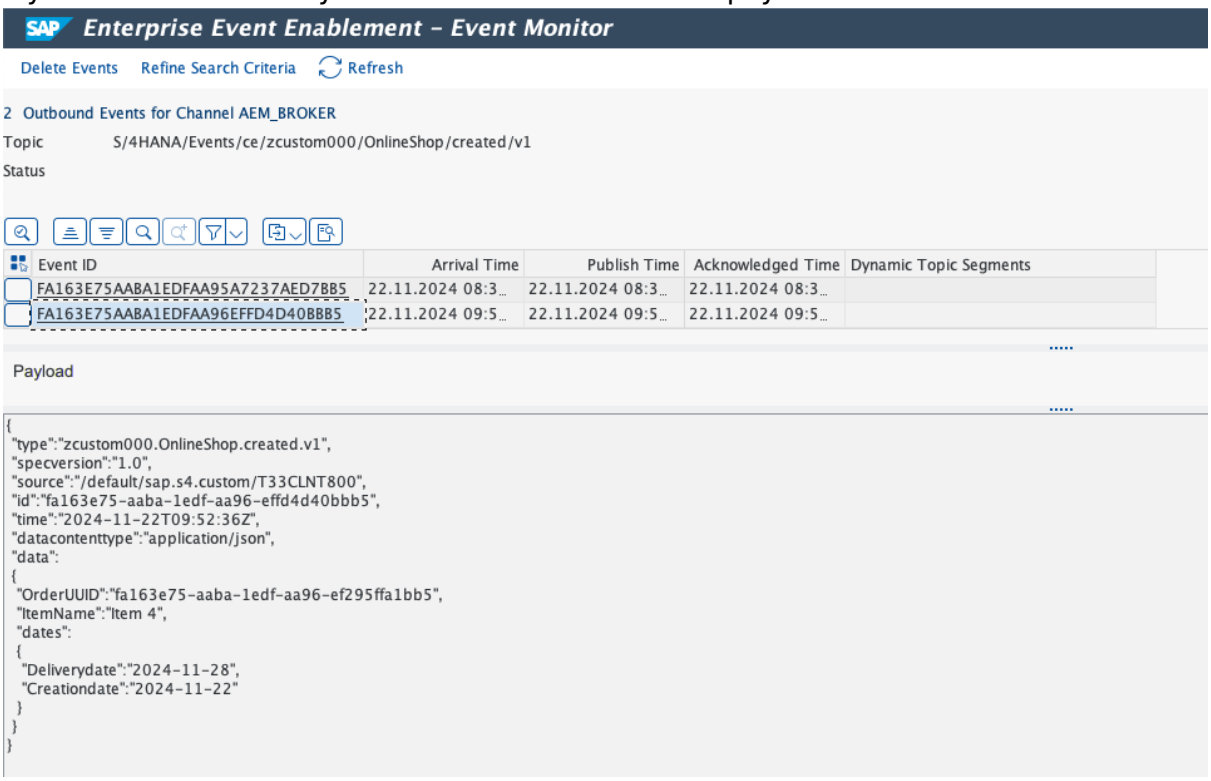
```
METHOD save_modified.
  IF create-onlineshop IS NOT INITIAL.
    RAISE ENTITY EVENT ZR_ONLINESHOP_000~ItemIsOrdered
```

```
      FROM VALUE #( FOR online_shop IN create-onlineshop ( %key          =
online_shop-%key
                                %param-ItemName   = online_shop-Ordereditem
                                %param-dates = VALUE #(
                                  Creationdate = online_shop-Creationdate
                                  Deliverydate = online_shop-Deliverydate
                                )
                                ) ).
    ENDIF.
  ENDMETHOD.
```

17. Save and activate the ABAP class

Not let's test the adjusted event

18. Open your service binding "ZUI_ONLINESHOP_000_O4" and start the preview
19. Create a new Order Item with an ItemName, Deliverydate and Creationdate
20. Next go into transaction "/IWXBE/EVENT_MONITOR", select your channel and navigate to the latest event
21. If you select that event you should be able to view its payload



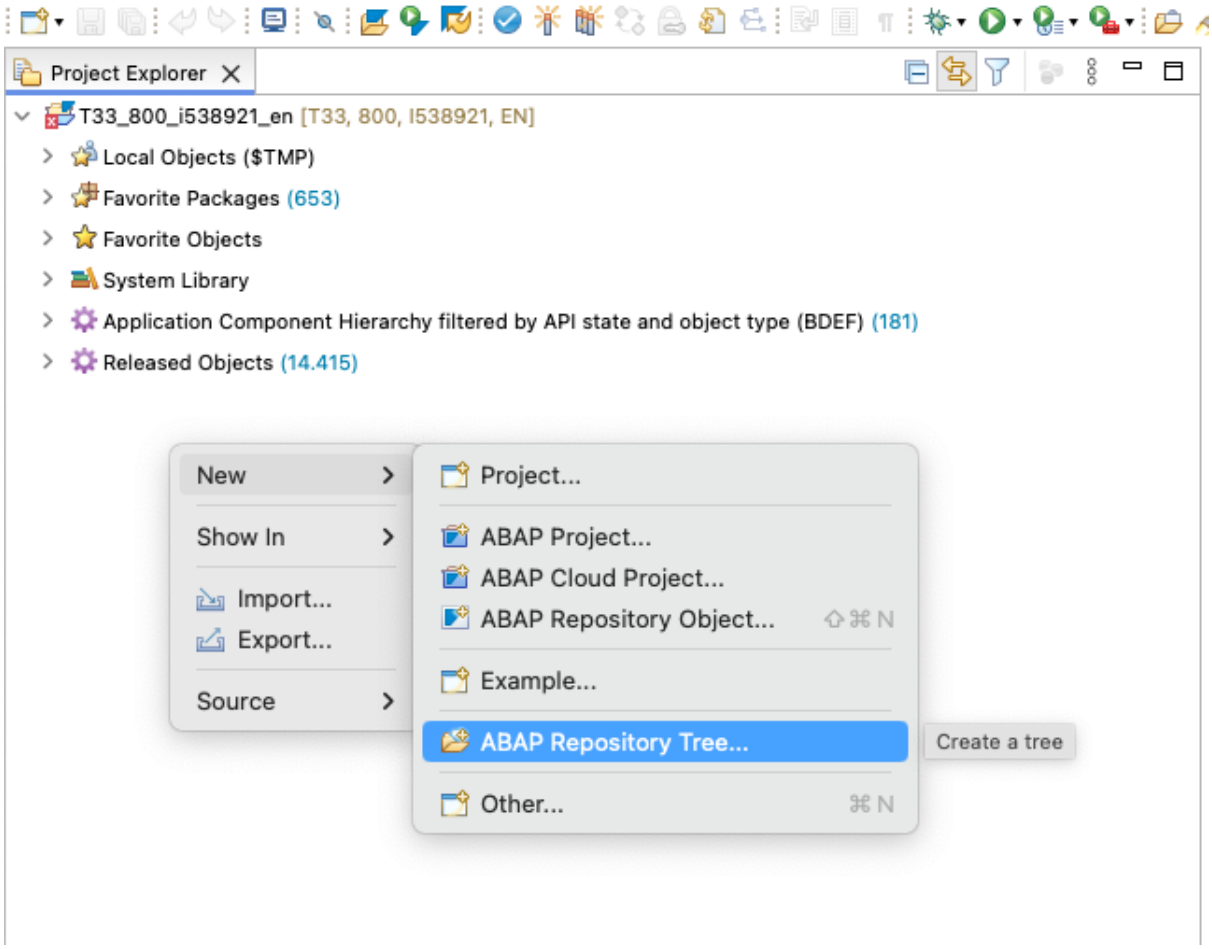
Customizing standard events using derived events

Standard events are often useful notifiers for any changes in the system. In many cases there is however the requirement to customize these events. For these cases so called derived events can be used. Derived events are new custom events which are bound to a standard event or any other already existing event. Whenever the already existing event is triggered, the derived event is triggered automatically alongside it. In addition to that derived events also give the possibility to pass along a CDS view which can be automatically filled with data allowing for a completely custom payload.

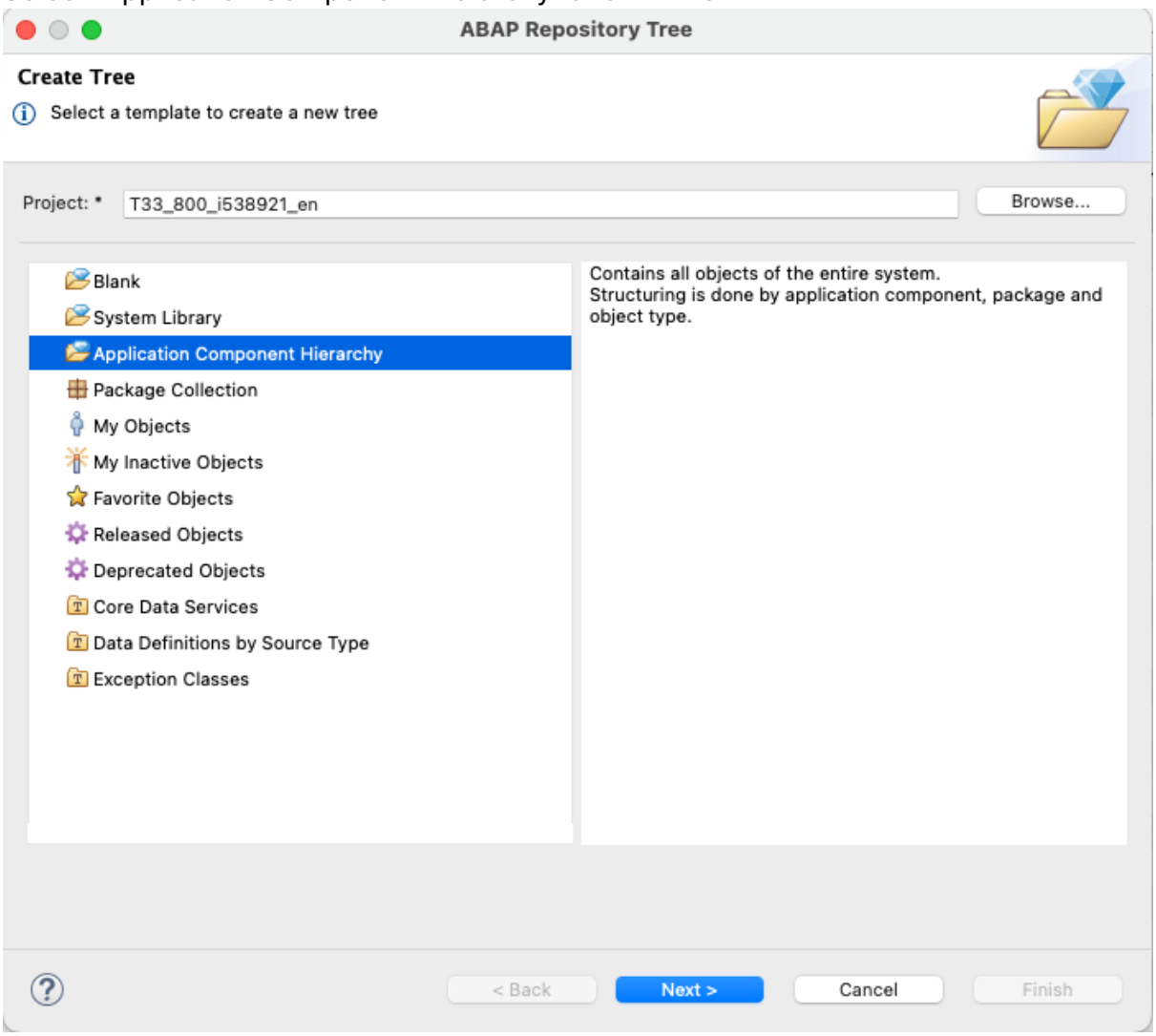
Let's look at how such a derived event can be created for the Business Partner Changed event.

Derived events always need to be defined on the same behavior definition as the base event. As such we first need to find out which behavior definition the business partner changed event belongs to. Normally the events are always defined on the business object itself. If you however don't know the exact name of the business object you can use an ABAP Repository Tree for help:

1. Right click into your project explorer and select "New > ABAP Repository Tree"

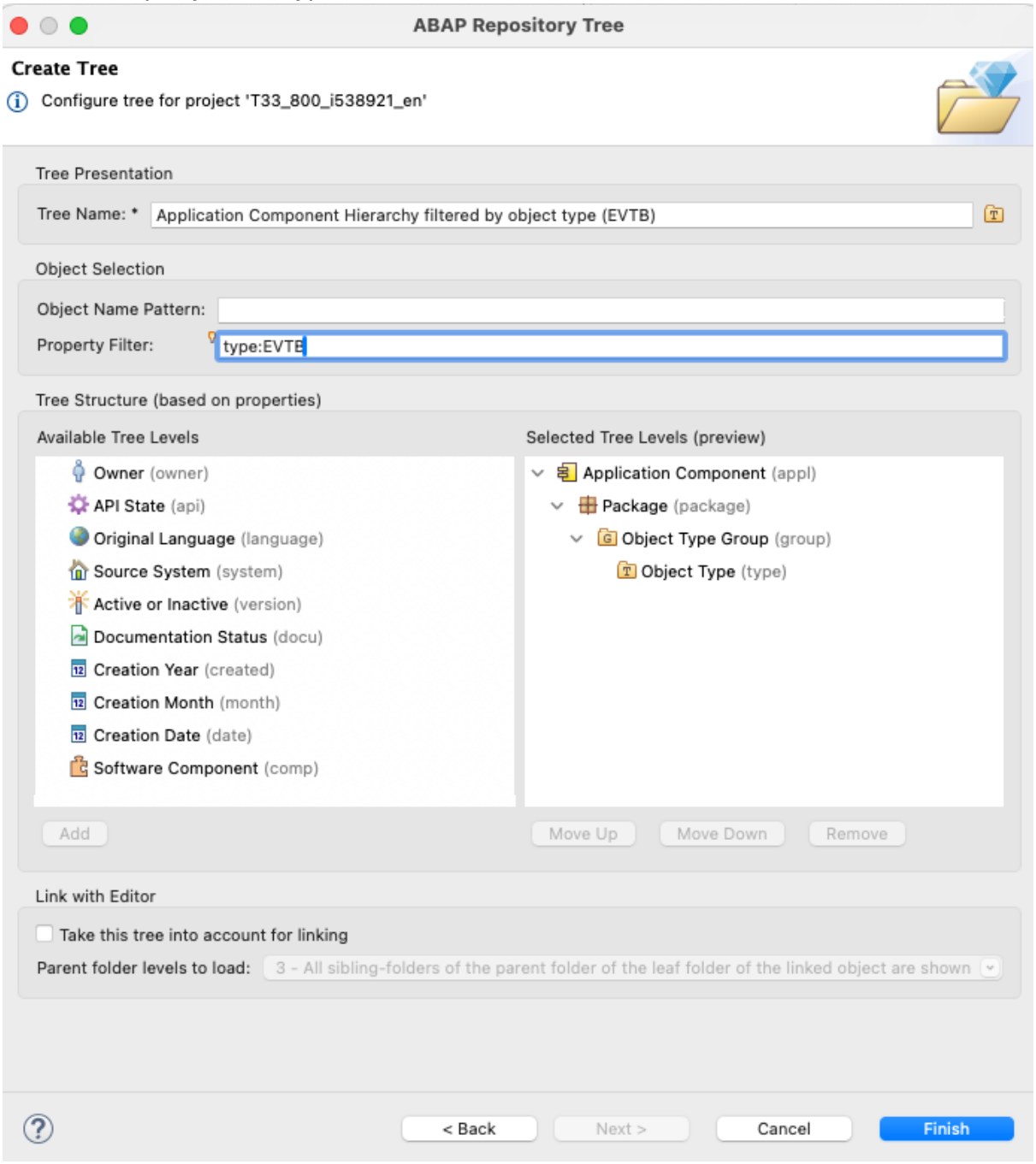


2. Select “Application Component Hierarchy” and hit “Next”



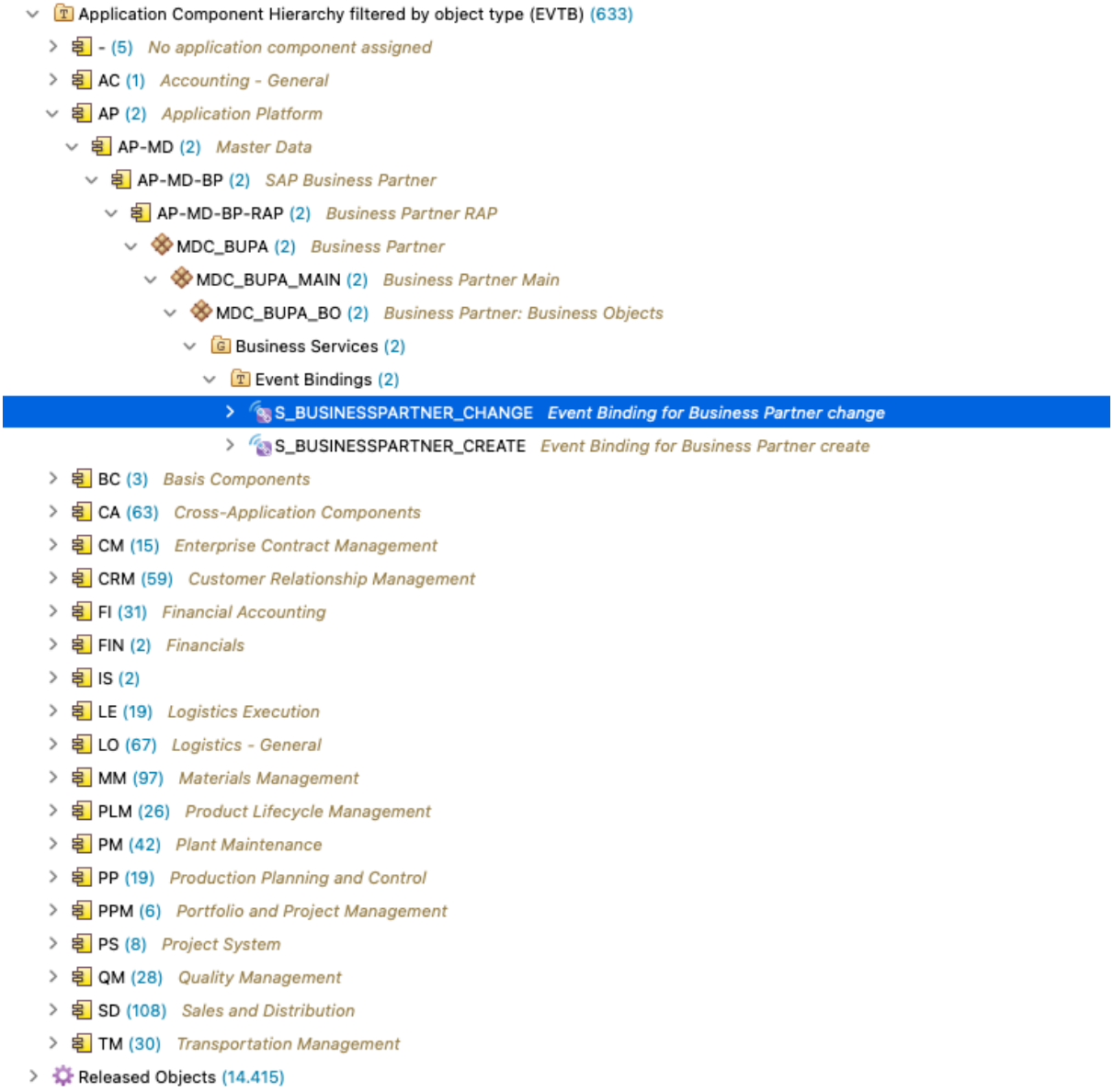
3. Name the Tree “Application Component Hierarchy filtered by object type (EVTB)”

4. Give the Property Filter “type:EVTB” and hit “Finish”

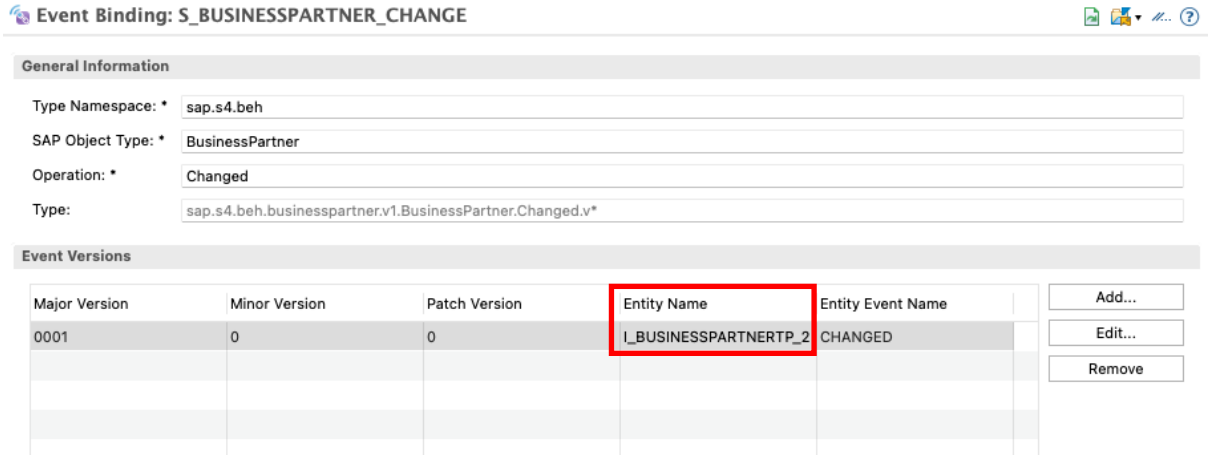


The resulting tree displays all event bindings ordered by their respective domain. It can therefore be used to more easily find the respective business events.

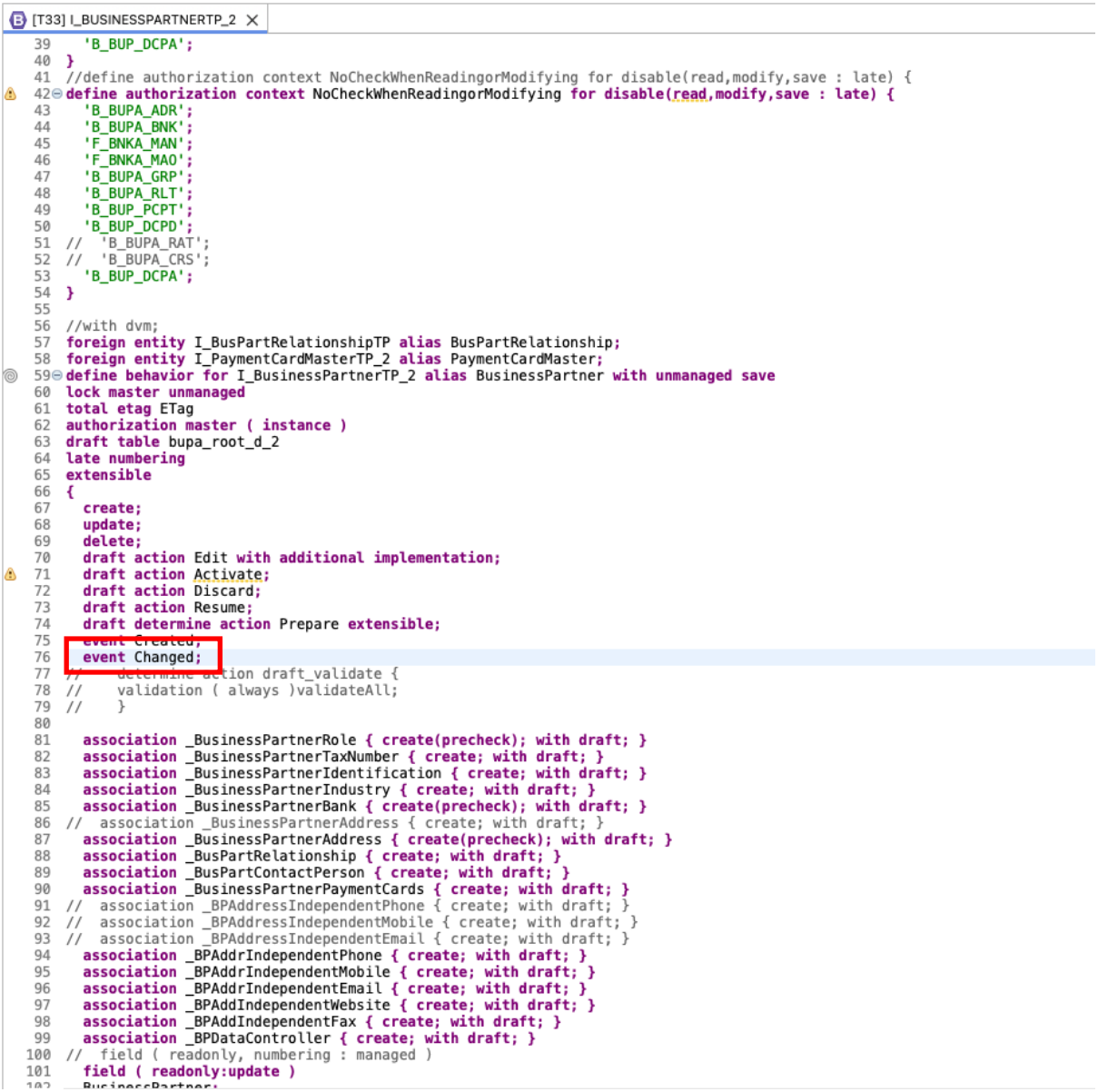
5. Open “AP-MD-BP-RAP” to view the Business Partner Changed event



6. Inspect the “Entity Name” field to see on which behavior definition the event is defined on

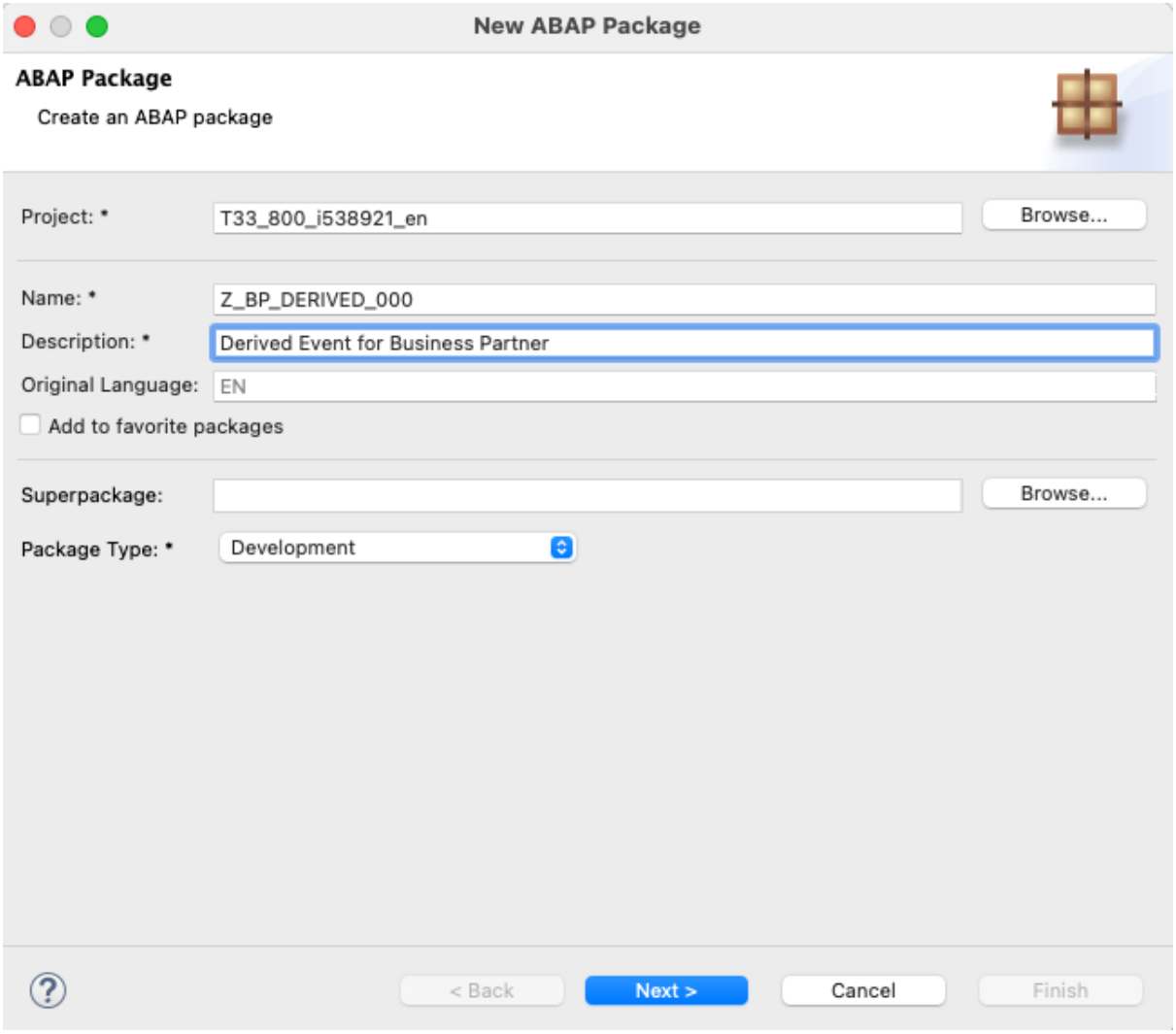


7. Open the behavior definition “I_BUSINESSPARTNERTP_2” and search for the Changed event



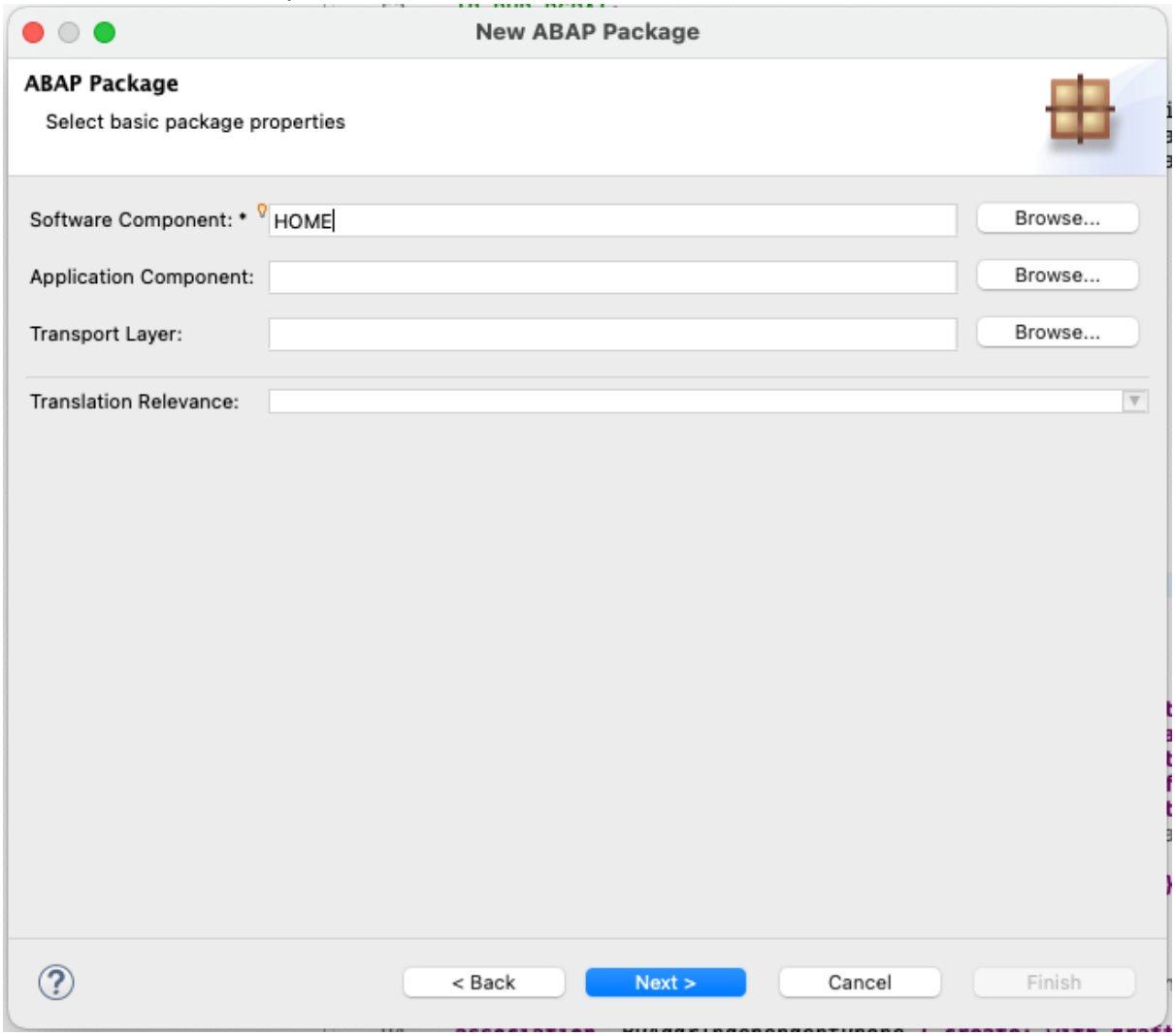
Just like that you’ve found the definition of the respective standard event.
As the standard Business Partner Changed event does not contain any data, lets create a derived event to customize this event to include a payload.
Since derived events use CDS views to define and populate the data of the payload, let’s start of by creating such a CDS view.

8. To start, create a new package with the name “Z_BP_DERIVED_000” and the description “Derived Event for Business Partner”



9. Click “Next”

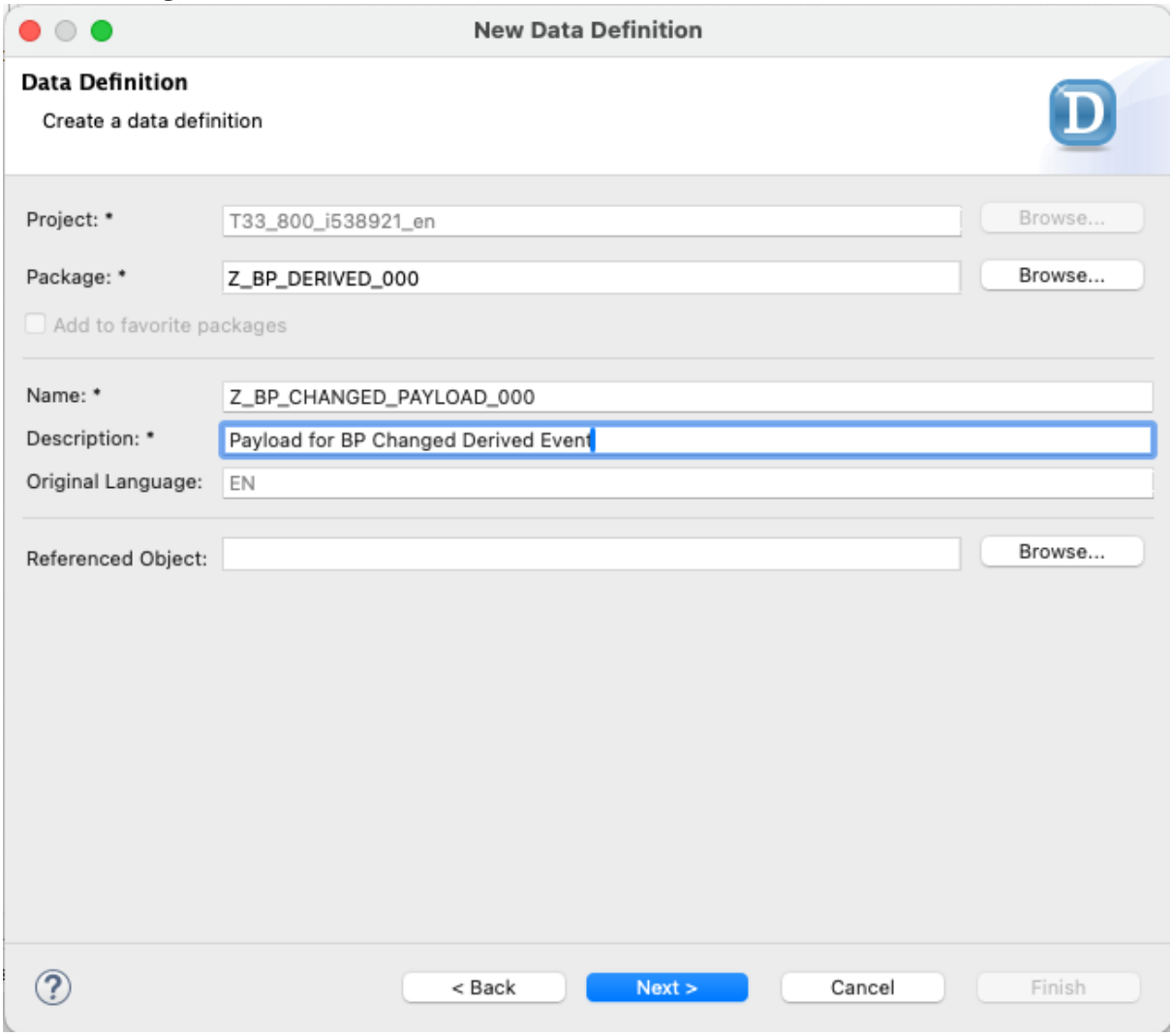
10. Select Software Component “HOME” and click “Next”



Software Component “HOME” is used to ensure that the ABAP Language Version “Standard ABAP” is selected by default. This is required because “ABAP for Cloud Development” can’t be used for derived events (Private Cloud and OnPremise) since the events currently aren’t accessible through released interfaces.

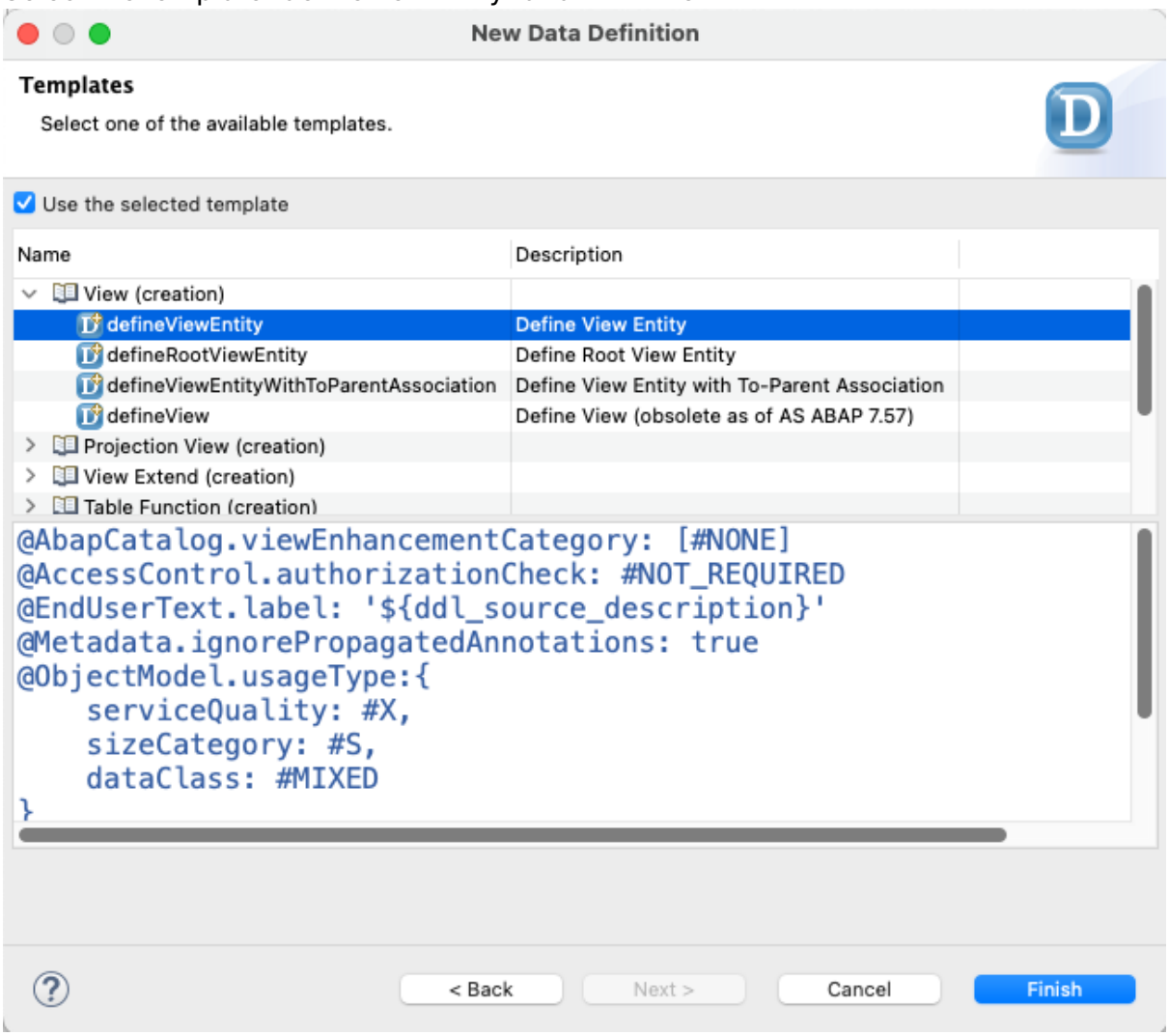
11. Select a transport request and hit “Finish”
12. Right click the package and add it to your favorite packages
13. Right click your package again and select “New”>”Other ABAP Repository Object”
14. Search for “Data Definition”, select it and hit “Next”

15. Give the name “Z_BP_CHANGED_PAYLOADD_000” and the description “Payload for BP Changed Derived Event”



16. Hit “Next”, select a transport request and hit “Next” again

17. Select the template “defineViewEntity” and hit “Finish”



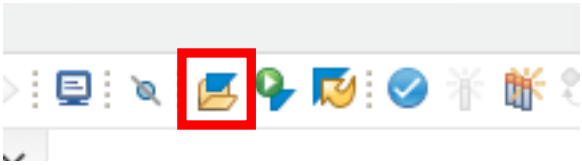
18. Enter the interface “I_BusinessPartner” as data source and add some respective fields to populate the payload:

```
@AbapCatalog.viewEnhancementCategory: [#NONE]
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Payload for BP Changed Derived Event'
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{
  serviceQuality: #X,
  sizeCategory: #S,
  dataClass: #MIXED
}
define view entity Z_BP_CHANGED_PAYLOAD_000 as select from
I_BusinessPartner
{
  key BusinessPartner,
    BbusinessPartnerCategory,
    BusinessPartnerFullName
}
```

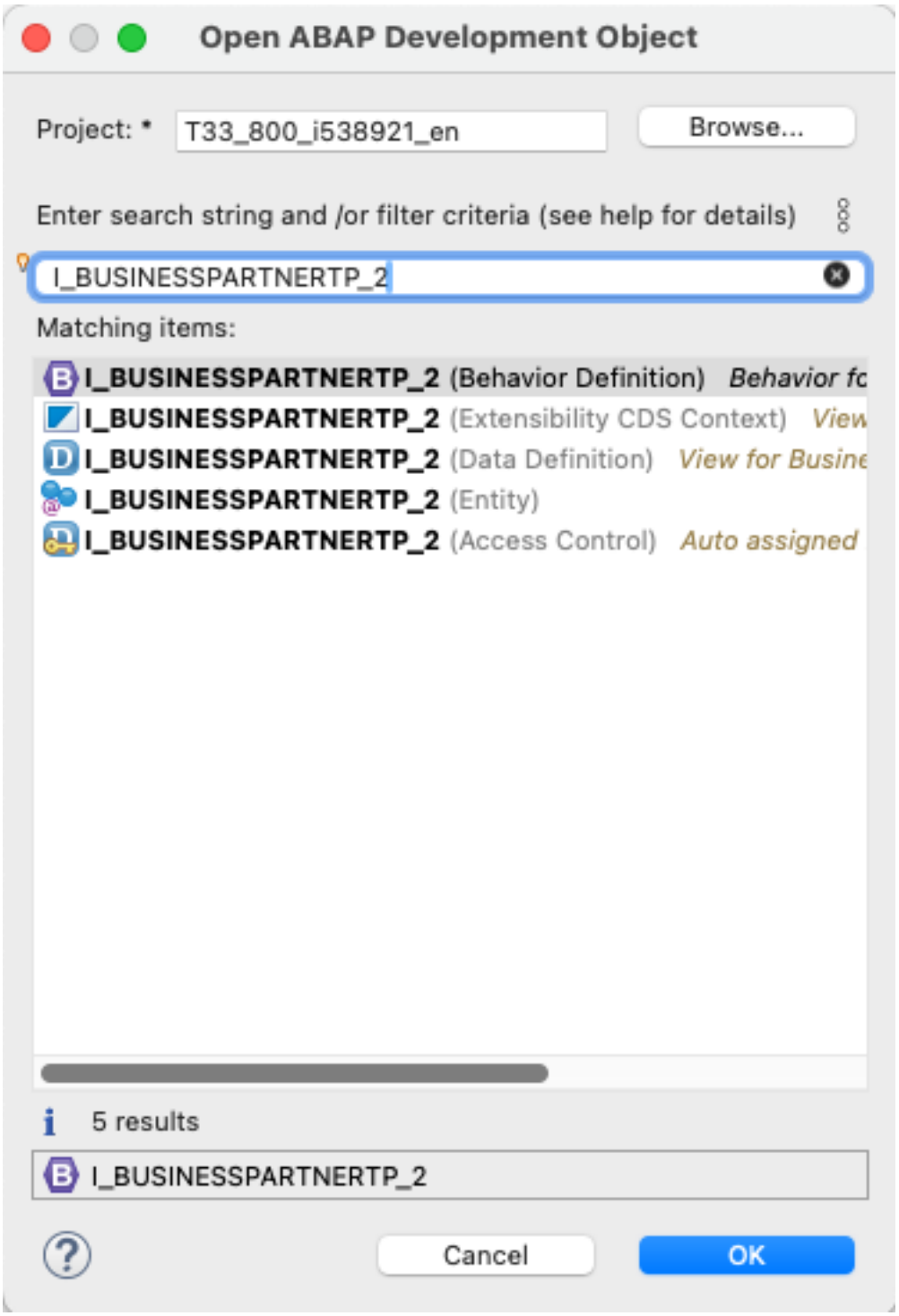
19. Save and activate the CDS view.

Next let’s define the derived event itself.

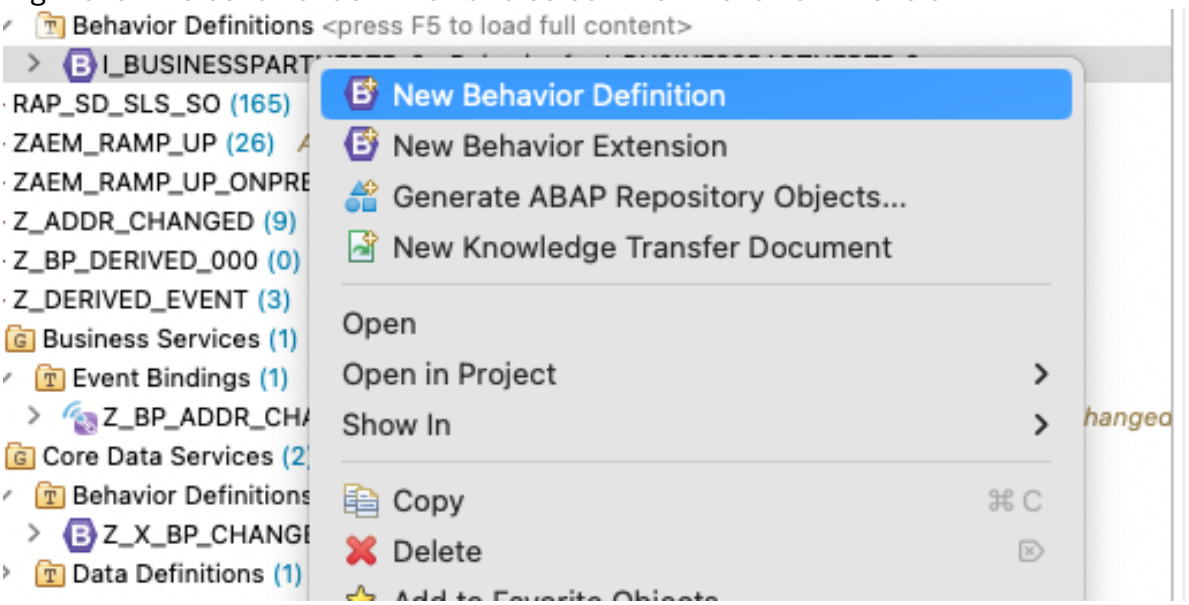
20. Open the “Open ABAP Development Object” dialog



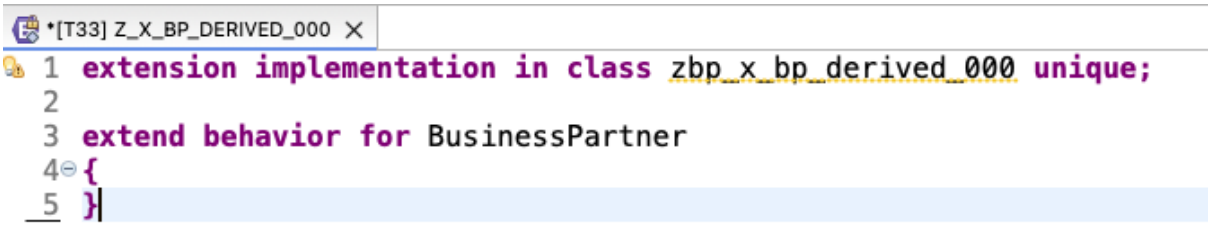
21. Search for “I_BUSINESSPARTNERTP_2” and select the behavior definition



22. Right click the behavior definition and select “New Behavior Extension”



23. Select your previously created package “Z_BP_DERIVED_000”
24. Give the name “Z_X_BP_DERIVED_000” and the description “I_BPTP_2 Extension for derived event”
25. Click “Next”, select a transport request and hit “Finish”
26. Remove all of the “extend behavior for” blocks exept the one for “BusinessPartner”



27. Add the definition of the derived event to the behavior extension

    extension implementation in class zbp_x_bp_derived_000 **unique;**

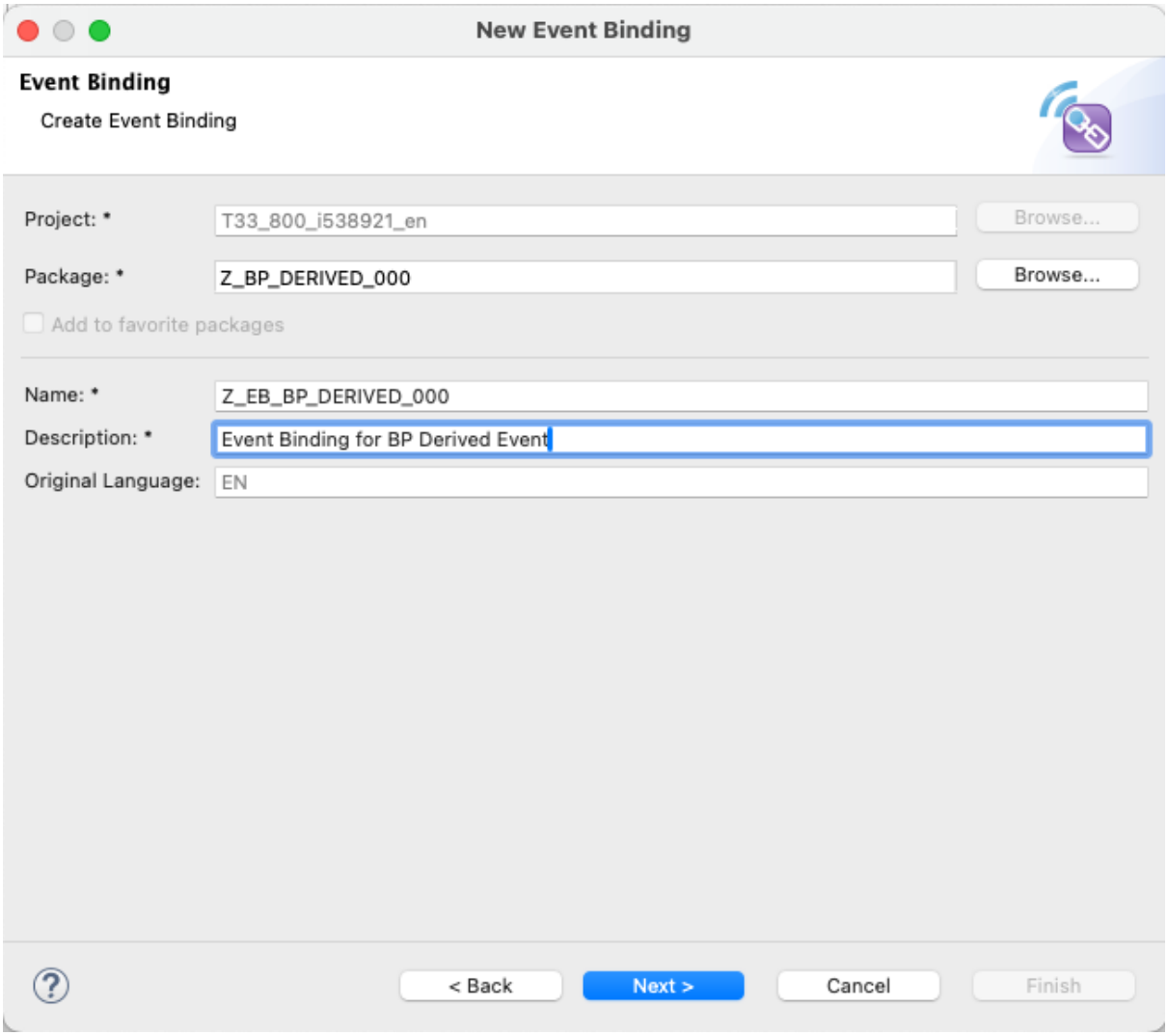
    extend behavior for BusinessPartner
    {
      managed event ChangedDerived_000 **on** Changed **parameter** Z_BP_CHANGED_PAYLOAD_000**;**
    }

28. Save and activate the behavior extension

The “managed” keyword is used to indicate that an event is a derived event. With the “on” keyword an event is given that the derived event is derived from and lastly “parameter” is used to pass the CDS view.

To publish this event to the Advanced Event Mesh once again an event binding is needed:

29. Right click your package and select “New”>”Other ABAP Repository Object”
30. Search for “Event Binding”, select it and hit “Next”
31. Give the name “Z_EB_BP_DERIVED_000” and the description “Event Binding for BP Derived Event”



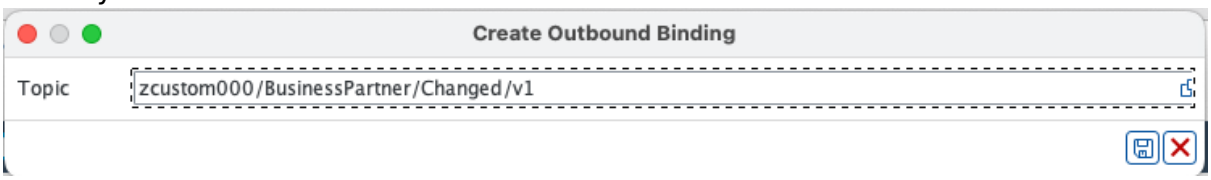
32. Hit “Next”, select a transport request and hit “Finish”
33. Enter the following general information:
    Type Namespace: zcustom000
    SAP Object Type: BusinessPartner
    Operation: changed

34. Press the “Add...” Button

35. Enter the Entity Name “I_BUSINESSPARTNERTP_2” and the Entity Event Name “CHANGEDDERIVED_000”
36. Activate the event binding

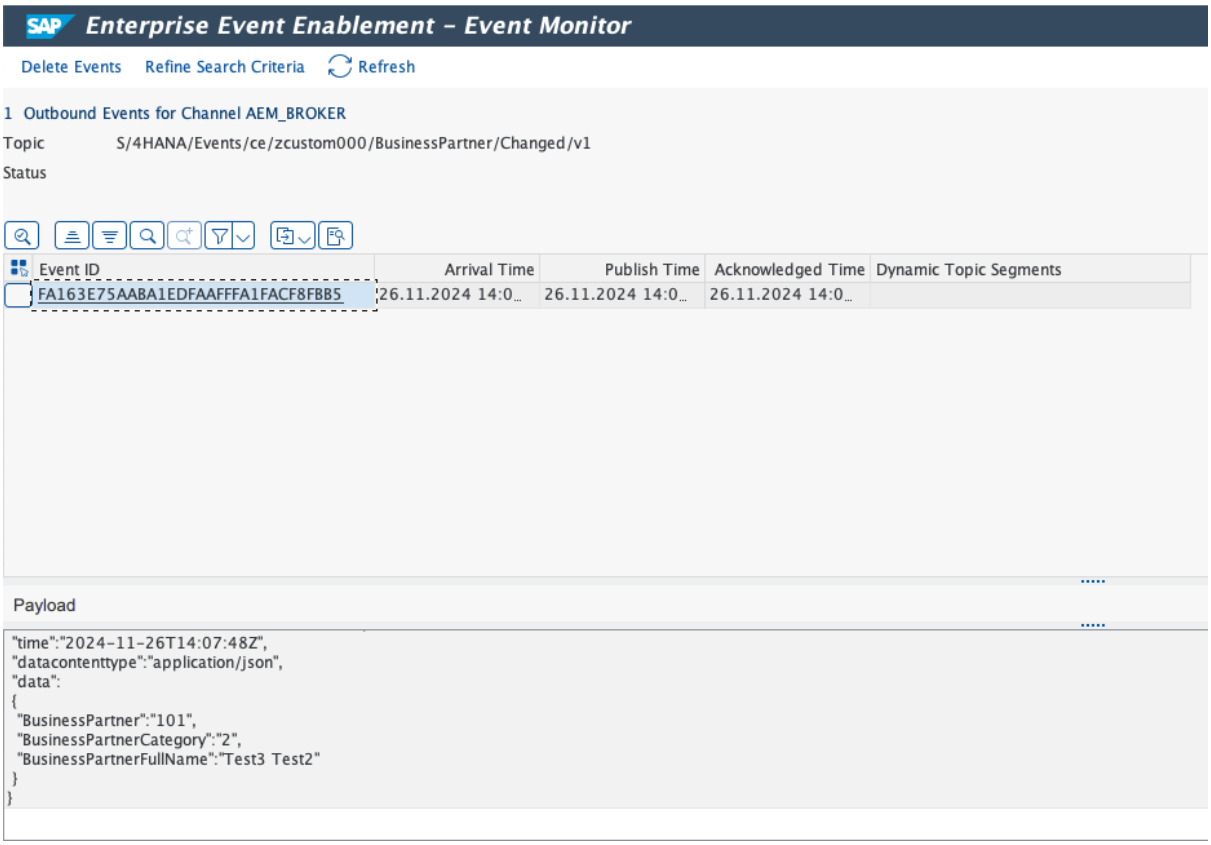
With the event binding created we can add the topic binding to send the events to the advanced event mesh.

37. Go into transaction “/IWXBE/OUTBOUND_CFG”
38. Select the channel to your AEM
39. Create a new topic binding
40. Use the search help to search for your event “zcustom000/BusinessPartner/changed/v1”
41. Select your event and hit “Save”



With that the derived event should be triggered when ever the standard event is triggered and should be send to the Advanced Event Mesh.
To test that trigger the standard event.

42. Go into transaction “BP”
43. Select a business partner, edit it and save the changes.
    This should have triggered the event.
44. Go into transaction “/IWXBE/EVENT_MONITOR”
45. Select your channel, select “Outbound Events” and select your event (depending on your S/4HANA version the steps could be different)



Filtering derived events using CDS views

One quite useful feature of derived events is the option to configure automatic filtering for events by defining conditions within the CDS view. Let's adjust the created derived event to only be triggered when the full name of a business partner exceeds a certain length.

1. Go into your CDS view "Z_BP_CHANGED_PAYLOAD_000
2. Add a "where"-clause to the CDS view

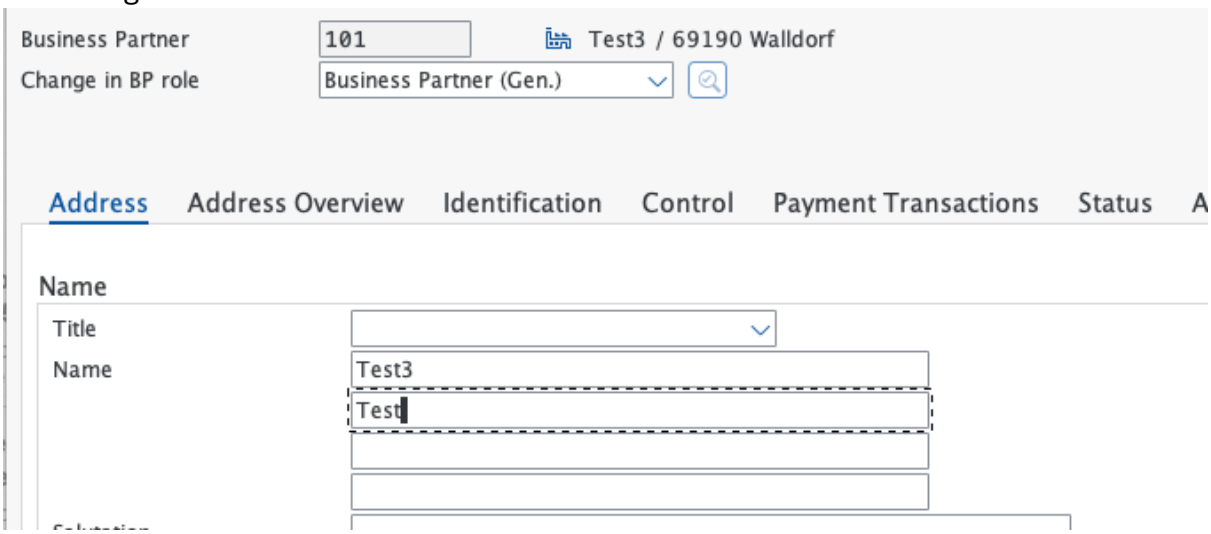
```
@AbapCatalog.viewEnhancementCategory: [#NONE]
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Payload for BP Changed Derived Event'
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{
   serviceQuality: #X,
   sizeCategory: #S,
   dataClass: #MIXED
}
define view entity Z_BP_CHANGED_PAYLOAD_000 as select from
I_BusinessPartner
{
```

```
    key BusinessPartner,
      BusinessPartnerCategory,
      BusinessPartnerFullName
  } where length(BusinessPartnerFullName) > 10
```

3. Save and activate the CDS view

This “where”-condition prevents events from being triggered when the full name of the business partner is less than 11 characters long. Let’s test that.

4. Go into transaction “BP”
5. Adjust the name fields so that the full name is less than 11 fields long and save the changes.



6. Go into transaction “/IWXBE/EVENT_MONITOR” and navigate to your event topic. There should be no new event being displayed
7. Go back transaction “BP” and adjust the name to be 11 characters or longer and save
8. Check the event monitor once again
9. Experiment with the CDS filter

10. Let’s remove the “where”-condition again from the CDS view for the next exercises

Creating topic filters

Besides filtering through the CDS view any kind of custom event can also be filtered by using context attributes. Context attributes enrich the event with additional data which allows advanced filtering and routing. This can be done by adding annotations to the fields of the payload. Let’s adjust our first event to utilize context attributes.

1. Open your CDS view “Z_BP_CHANGED_PAYLOAD_000”
2. Add a new field “_DefaultAddress._Address.HouseNumber” with the annotation “@Event.context.attribute: 'xsapbphousenumber'”

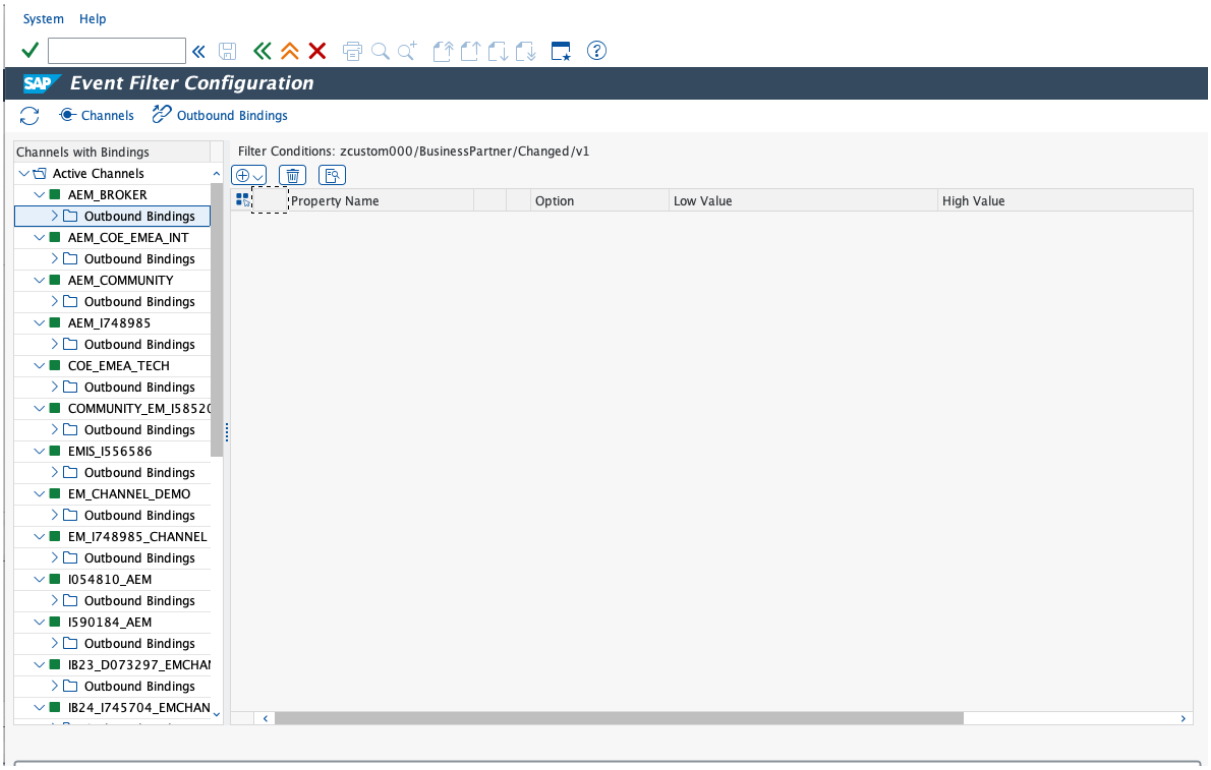
```
@AbapCatalog.viewEnhancementCategory: [#NONE]
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Payload for BP Changed Derived Event'
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{
  serviceQuality: #X,
  sizeCategory: #S,
  dataClass: #MIXED
}
define view entity Z_BP_CHANGED_PAYLOAD_000 as select from
I_BusinessPartner
{
  key BusinessPartner,
    BusinessPartnerCategory,
    BusinessPartnerFullName,
    @Event.context.attribute: 'xsapbphousenumber'
    _DefaultAddress._Address.HouseNumber
}
```

3.

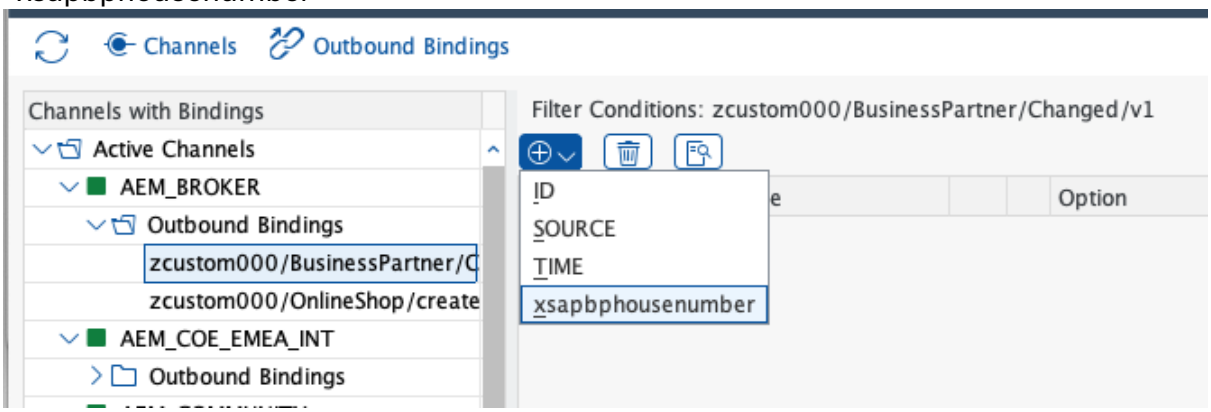
The “@Event.context.attribute” annotation is required to indicate which fields should be accessible through the event context and under which name they should be accessible. The context attributes must always start with “xsap” and allow only specific characters.

4. Save and activate the CDS view

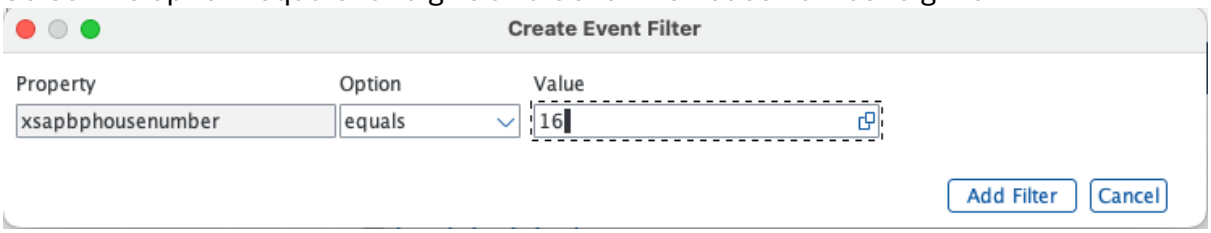
5. Go to transaction “/IWXBE/EVENT_FILTER”



6. Expand the Outbound Bindings of your Channel and select the “zcustom/BusinessPartner/Changed/v1” binding.
7. Click on the “Add Event Filter” button and select your context attribute “xsapbphousenumber”



8. Select the option “equals” and give a value for the house number e.g. 16



This would filter all instances of the derived event and only allow thouse instances where the house number equals 16.

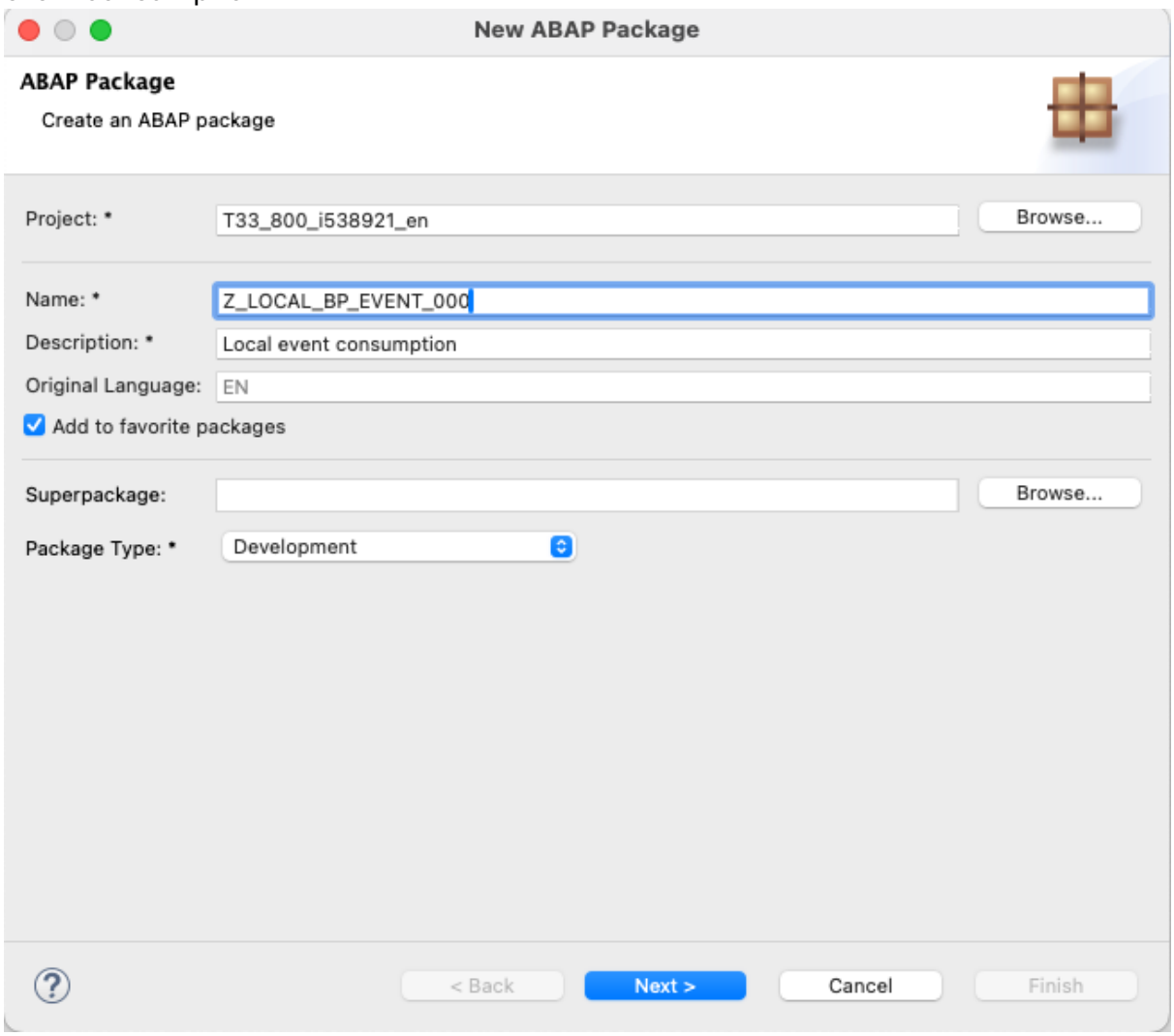
9. Go into transaction “BP”, select a business partner, change the house number to 16 and save.

10. Go into transaction “/IWXBE/EVENT_MONITOR”, navigate to the derived event and check that the event has been fired
11. Go back to transaction “BP”, change the house number to 17 and save.
12. Again check in the event monitor if the event has been fired. There should be no event listed for the latest change.

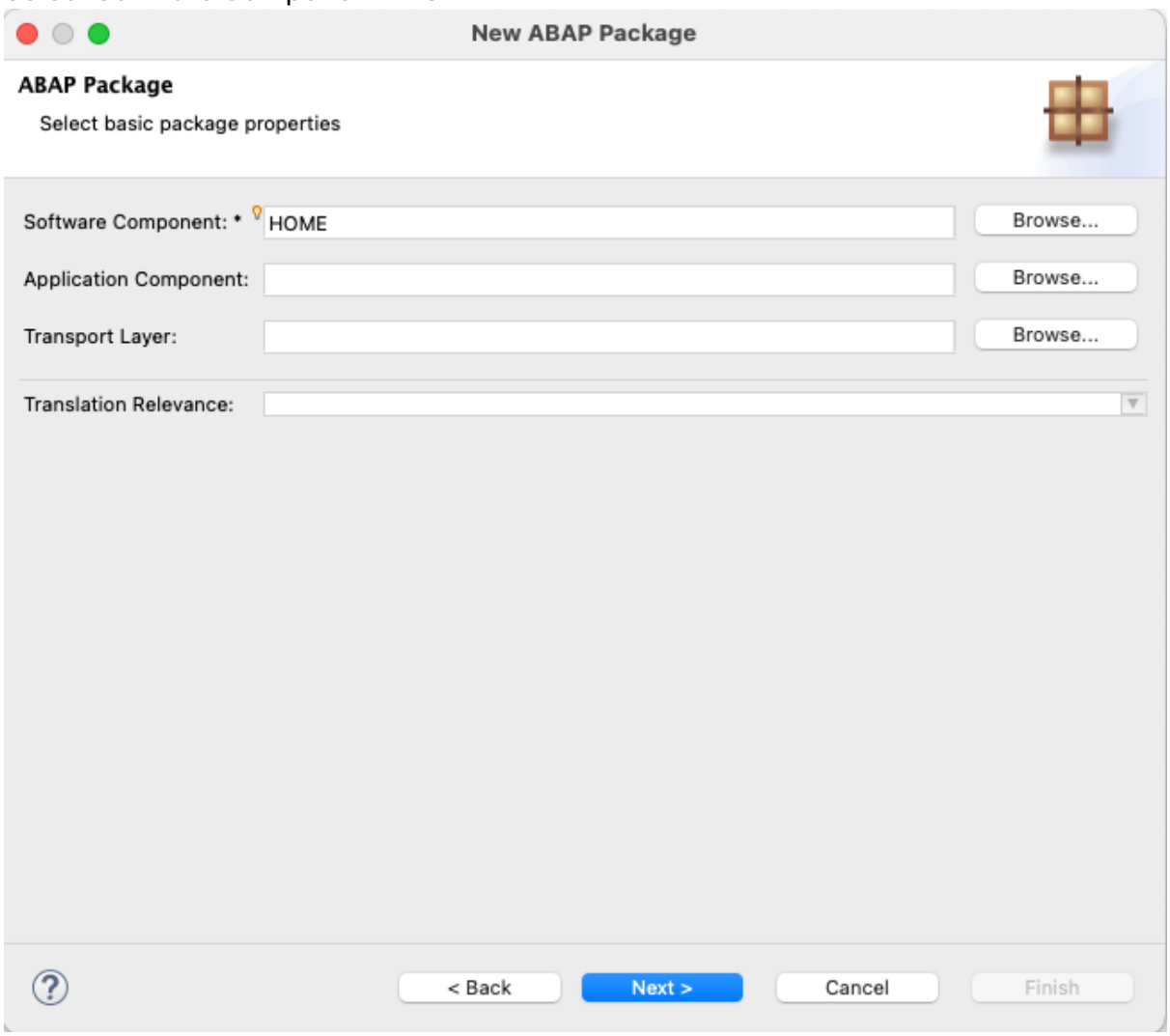
Creating local event handlers

Besides creating and triggering events RAP also provides the option to consume events. These events can be consumed directly from the system itself or from the Advanced Event Mesh. Let’s create an event handler to consume local events of Business Partner Changed.

1. Create a new ABAP Package “Z_LOCAL_BP_EVENT_000” with description “Local event consumption”

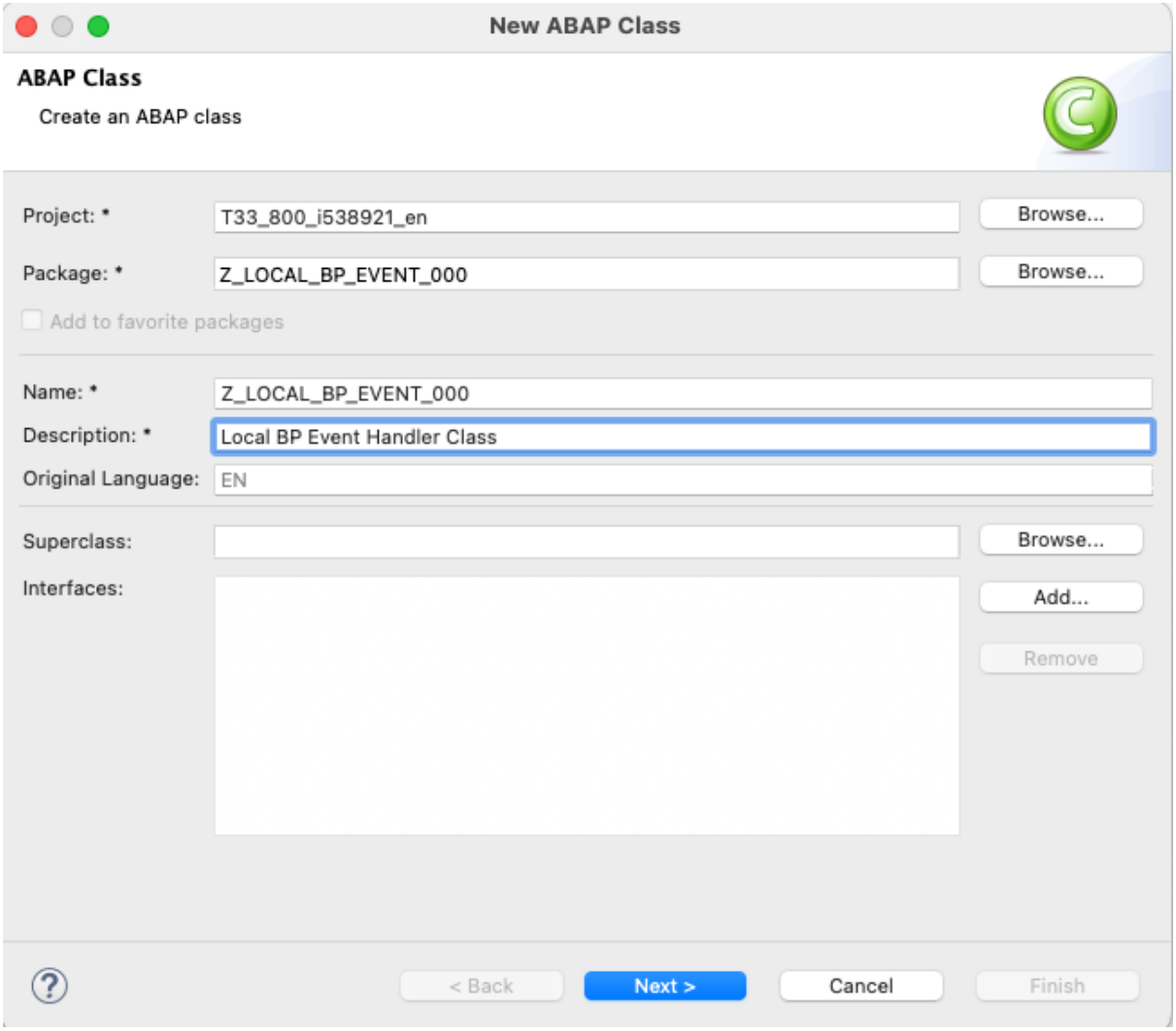


2. Select Software Component “HOME”



3. Select a transport request and click “Finish”
4. Right click your package and select “New”>”ABAP Class”

5. Give the name “Z_LOCAL_BP_EVENT_000” and the description “Local BP Event Handler Class”



6. Click “Next”, select a transport request and hit “Finish”
7. Adjust the global class according to the following code:

```
CLASS z_local_bp_event_000 DEFINITION PUBLIC ABSTRACT FINAL FOR
EVENTS OF I_BusinessPartnerTP_2.

 PUBLIC SECTION.
 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.

CLASS z_local_bp_event_000 IMPLEMENTATION.
ENDCLASS.
```

This declares the ABAP class as an event handler for all the events defined on the “I_BusinessPartnerTP_2” object. Since this object is not released the ABAP Language Version “ABAP for Cloud Development” can’t be used. Hence, we selected the Software Component “HOME” when creating the package.

8. Next switch over to the local types.
9. Enter the following code:

```
CLASS lcl_abap_behv_event_handler DEFINITION INHERITING FROM
cl_abap_behavior_event_handler.

 PRIVATE SECTION.

   METHODS consume_changed FOR ENTITY EVENT changed_instances FOR
BusinessPartner~Changed.

ENDCLASS.

CLASS lcl_abap_behv_event_handler IMPLEMENTATION.

 METHOD consume_changed.

  LOOP AT changed_instances INTO DATA(changed_instance).

    cl_demo_output=>write( 'Event consumed' ).

  ENDLOOP.

 ENDMETHOD.

ENDCLASS.
```

This defines a new event handler method “consume_changed” for the Business Partner Changed event. The variable “changed_instance” is used to store the instance of the event and can be used to access the payload of the event within the method implementation. For each event defined on the “I_BusinessPartnerTP_2” behavior definition a dedicated method can be created. The “cl_demo_output=>write(‘Event consumed’)” statement is used as a placeholder, as it serves no function. It can be replaced for dedicated business logic.

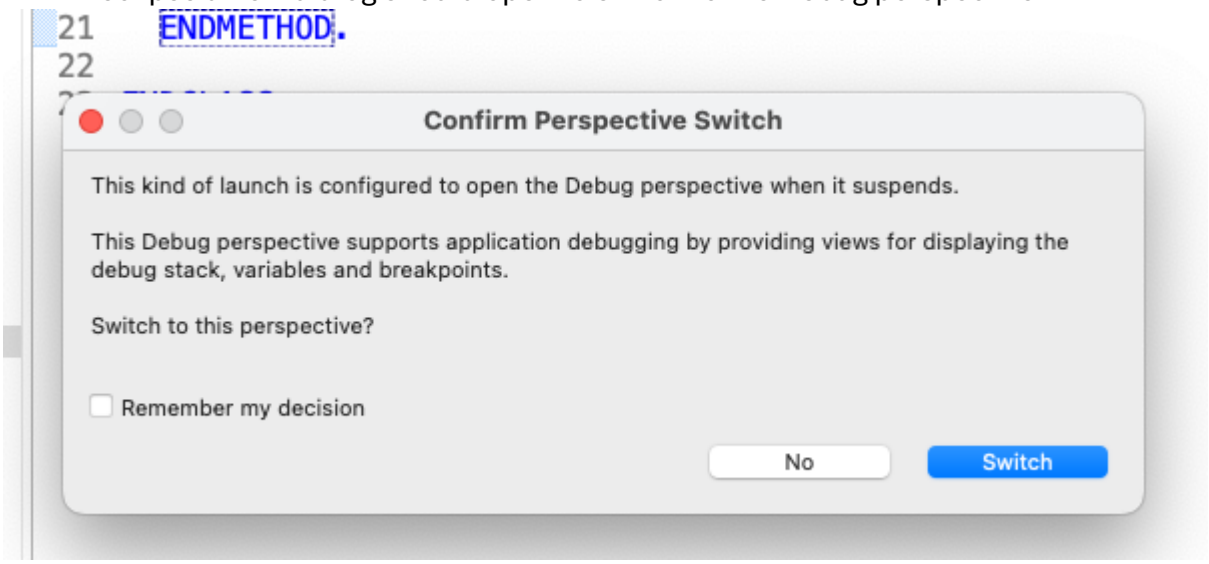
Let’s check if the local event handler is executed.

10. Set a breakpoint by double clicking the line number of the “cl_demo_output=>write(‘Event consumed’) statement

```
Z_LOCAL_BP_EVENT_000 ▸ LCL_ABAP_BEHV_EVENT_HANDLER ▸ CONSUME_CHANGED ▸ loop
1  CLASS lcl_abap_behv_event_handler DEFINITION INHERITING FROM cl_abap_behavior_event_handler.
2
3    PRIVATE SECTION.
4
5      METHODS consume_changed FOR ENTITY EVENT changed_instances FOR BusinessPartner~Changed.
6
7  ENDCLASS.
8
9  CLASS lcl_abap_behv_event_handler IMPLEMENTATION.
10
11   METHOD consume_changed.
12
13     LOOP AT changed_instances INTO DATA(changed_instance).
14
15       cl_demo_output=>write( 'Event consumed' ).
16
17     ENDLOOP.
18
19   ENDMETHOD.
20
21 ENDCLASS.
```

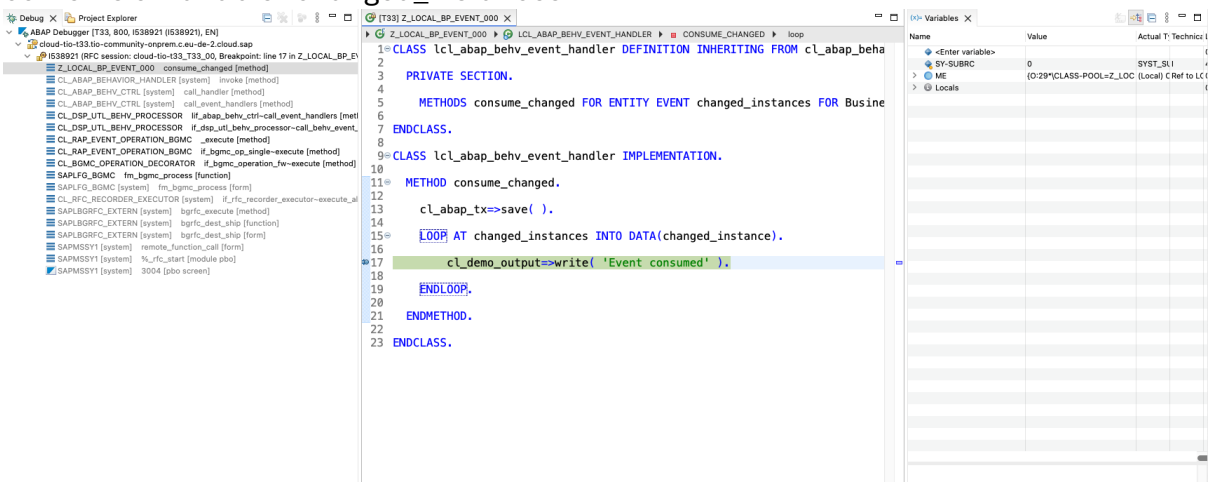
The breakpoint should be indicated by a blue dot at the beginning of the line

11. Go into SAPGUI and open transaction “BP”
12. Select a business partner, change it and save the changes.
13. Within eclipse a new dialog should open to switch to the Debug perspective



14. Press “Switch”

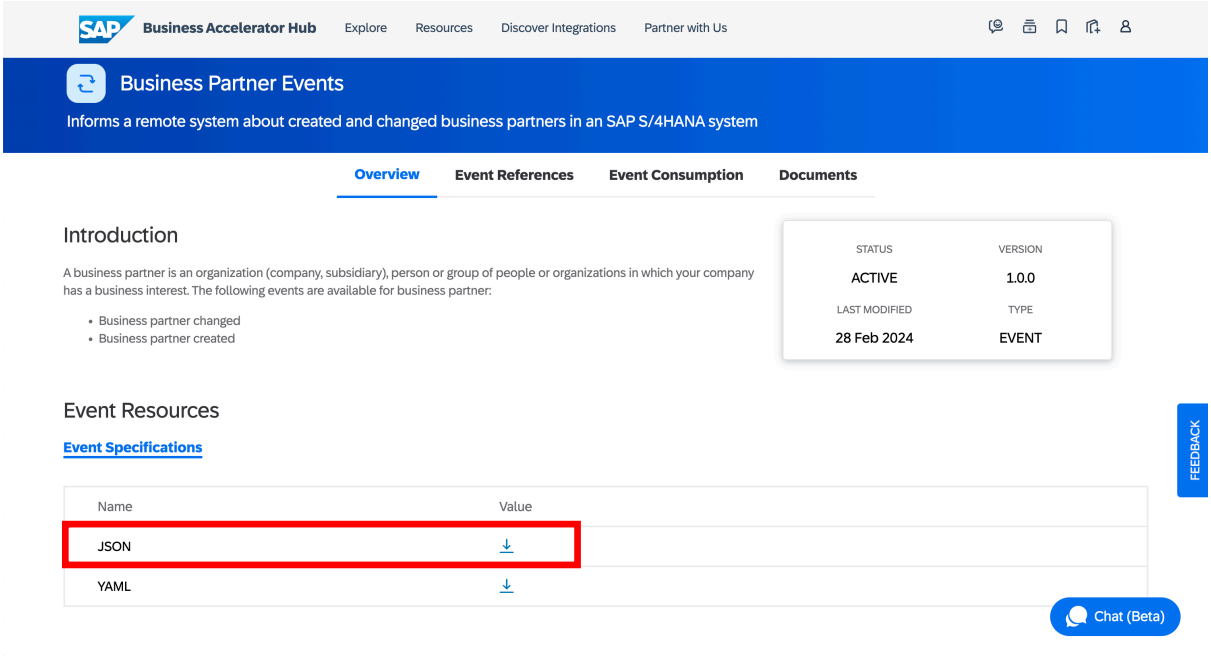
15. The program should have stopped at the breakpoint allowing you to inspect the contents of variable “changed_instances”



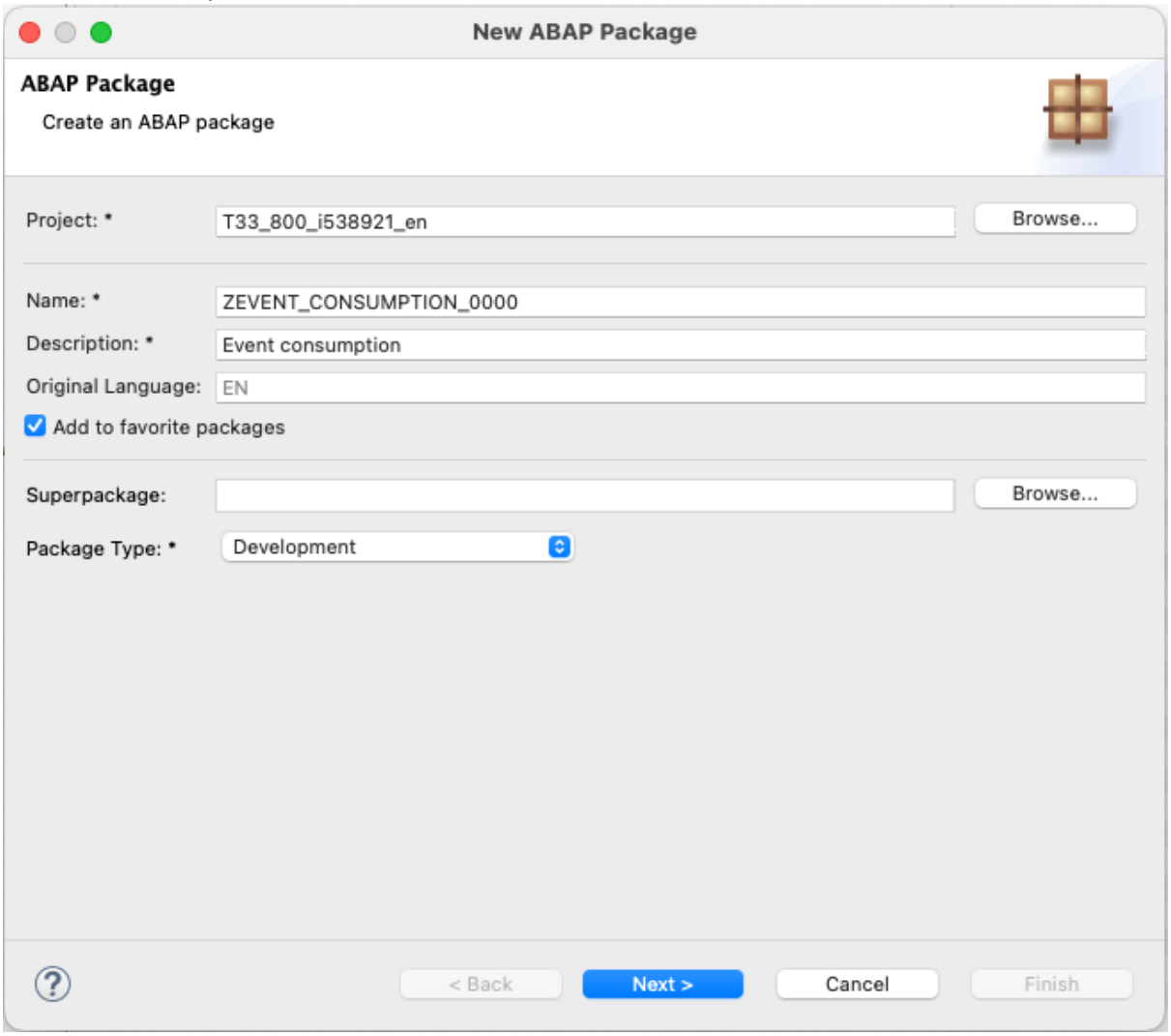
Receive events with S/4HANA

Besides sending events to the Advanced Event Mesh it is also possible to receive events from it. To demonstrate that we will configure the system to receive the Business Partner Changed event that is send to the AEM as per the first step. For that we’ll be creating a consumption model in ABAP Development Tools, configure an inbound binding and configure a subscription.

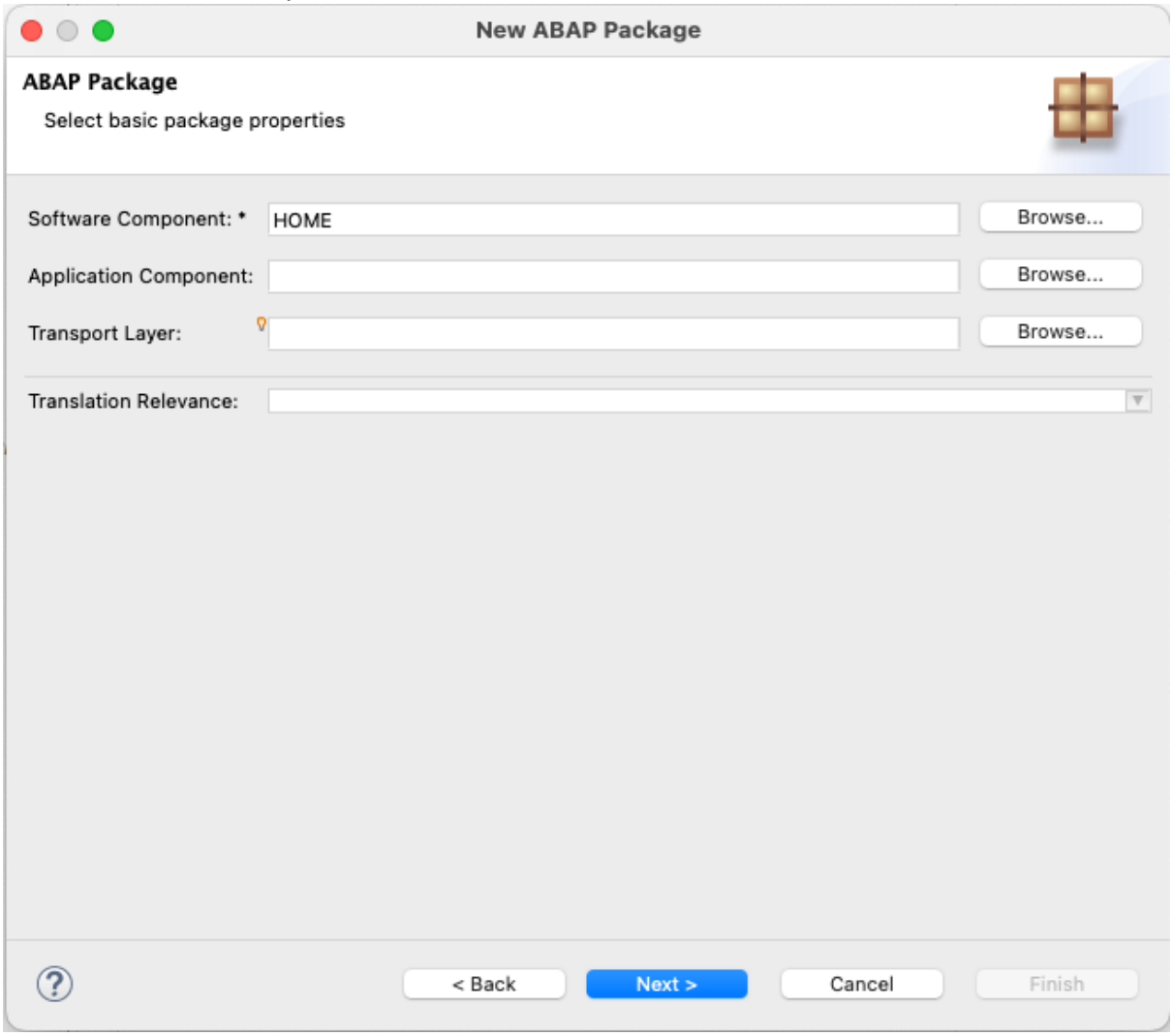
1. Go to the event specification of Busienss Partner Events within the Business Accelerator Hub and download the JSON file.



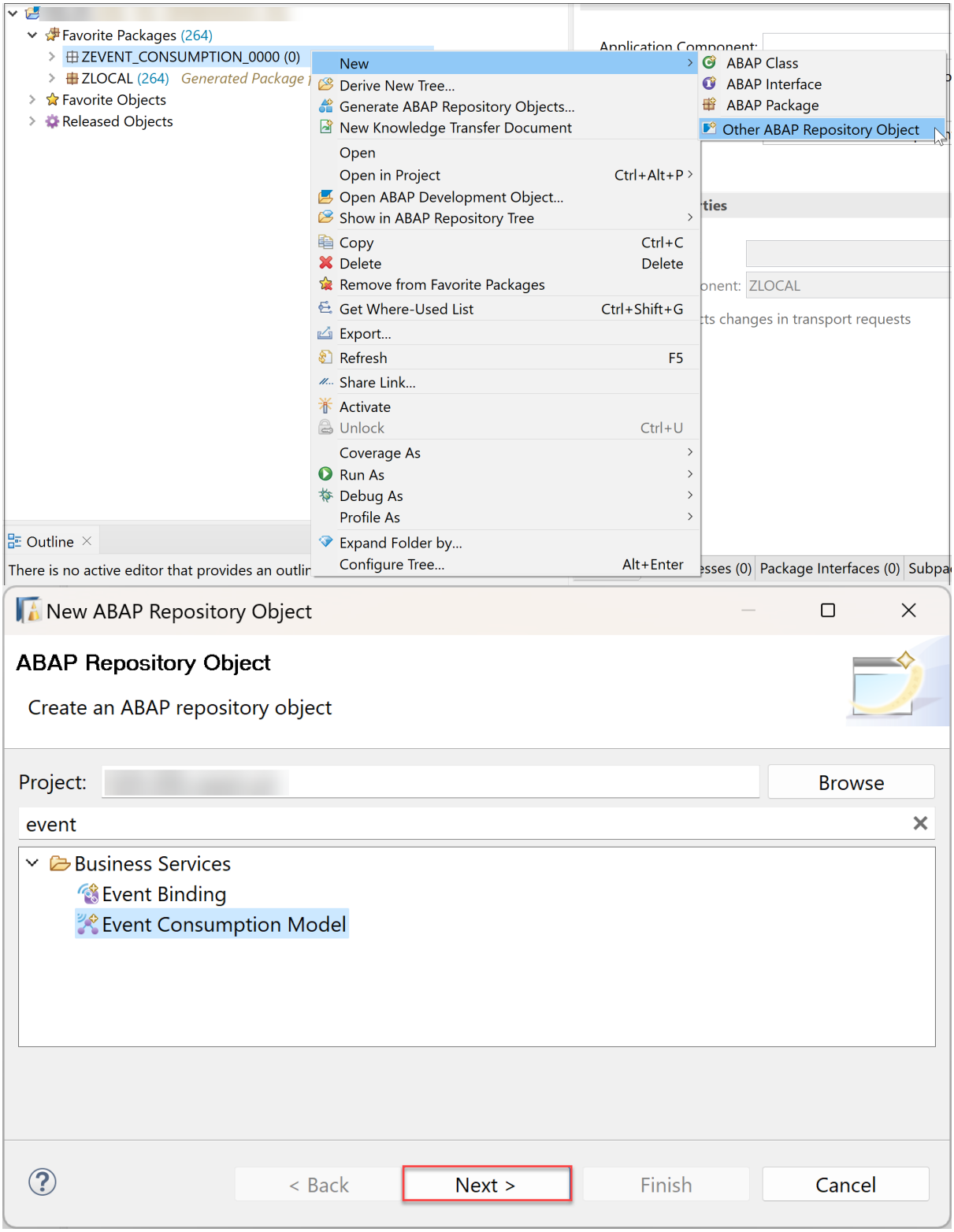
2. Create a new ABAP Package “ZEVENT_CONSUMPTION_0000” with description “Event consumption”



3. Select Software Component “HOME”

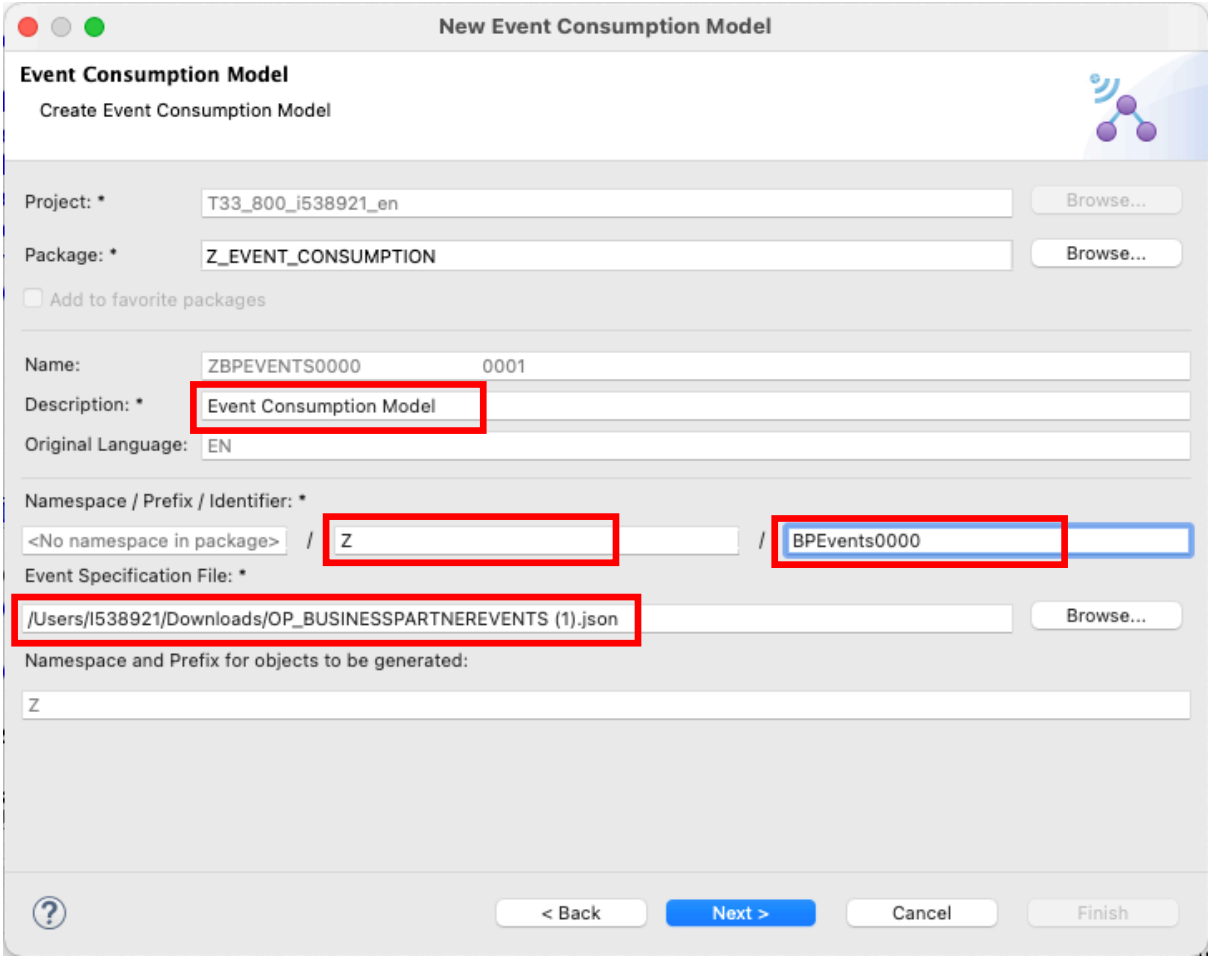


4. Right-click your package and choose **New** > **Other ABAP Repository Object** > **Business Services** > **Event Consumption Model** and click **Next** to launch the creation wizard.

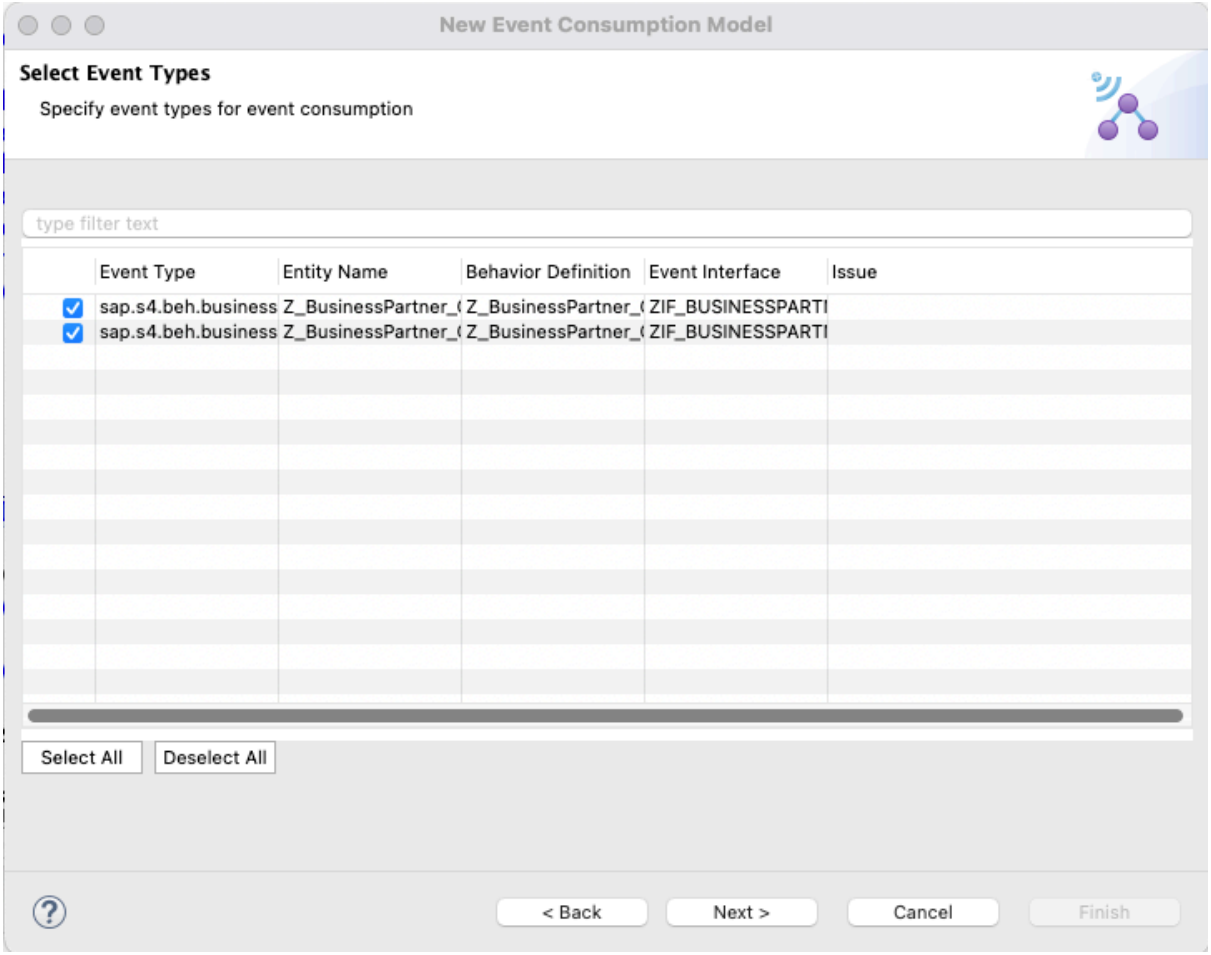


5. Fill the fields and upload the JSON Event Specification file you saved before.
   Description: Event consumption model

Namespace/Prefix/Identifier: “Z” and “BP_Events_0000”



6. Select both events and press “Next”

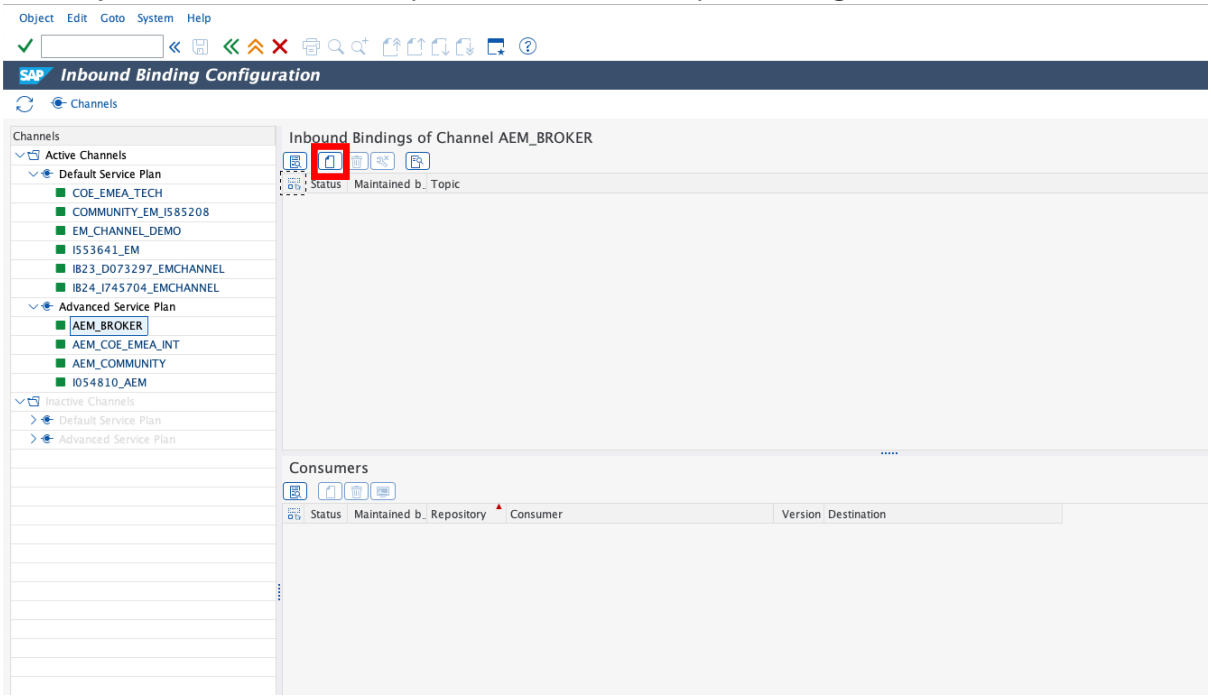


7. Press “Next” to the consumer Artifacts and “Next” to the ABAP Artifact Generation List
8. Select a Transport Request and click “Finish”
9. Save and activate your event consumption model

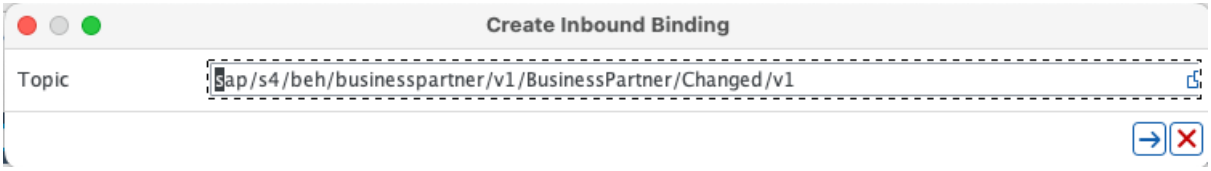
With the event consumption model created we can go into SAP GUI and configure the inbound binding

10. Open transaction /IWXBE/INBOUND_CFG

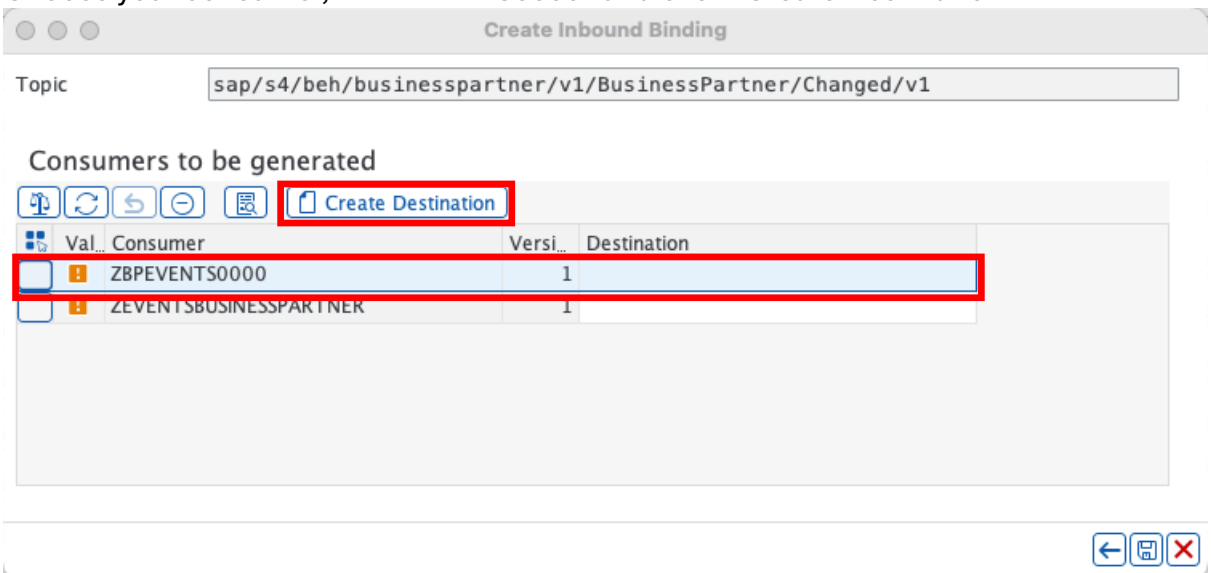
11. Select your AEM channel and press “Create new topic binding”



12. Enter “sap/s4/beh/businesspartner/v1/BusinessPartner/Changed/v1” or use the search help



13. Switch to the next page
14. Choose your consumer, “ZBPEVENTS0000” and click “Create Destination”

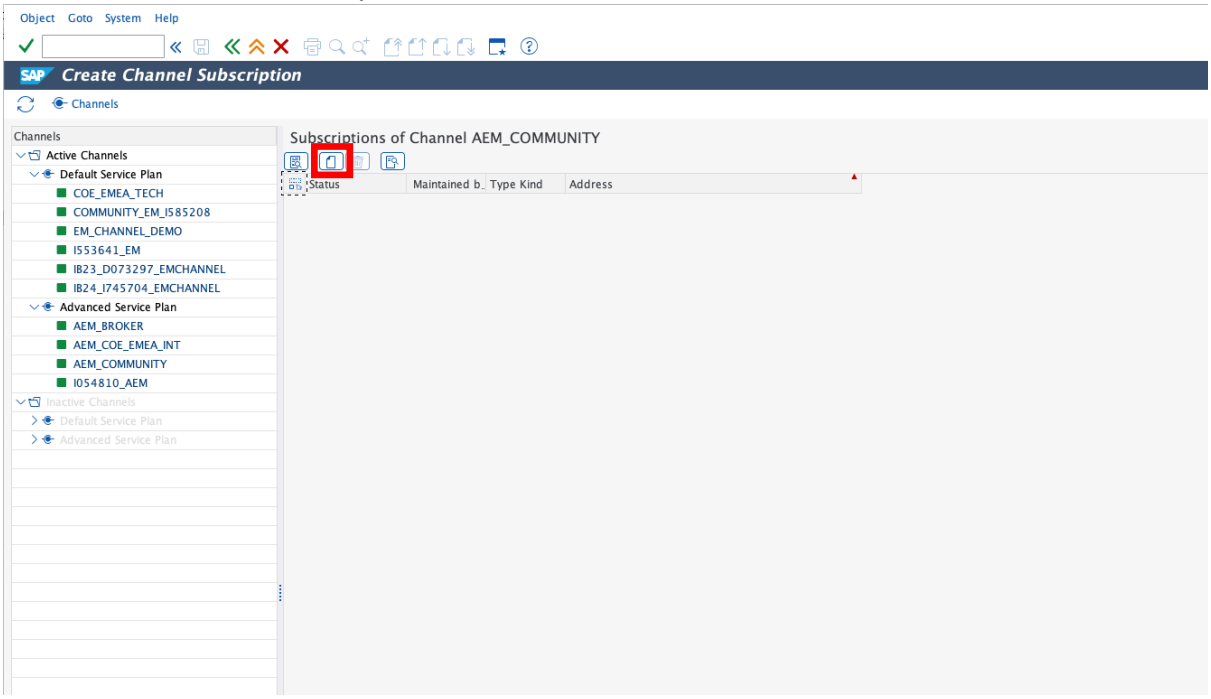


15. Enter your user name in the Create Destination dialog and press save
16. Press save on the Create Inbound Binding dialog

With the Inbound Binding configured we can configure the subscription next

17. Open transaction “/IWXBE/SUBSCRIPTION”

18. Click “Create new subscription”



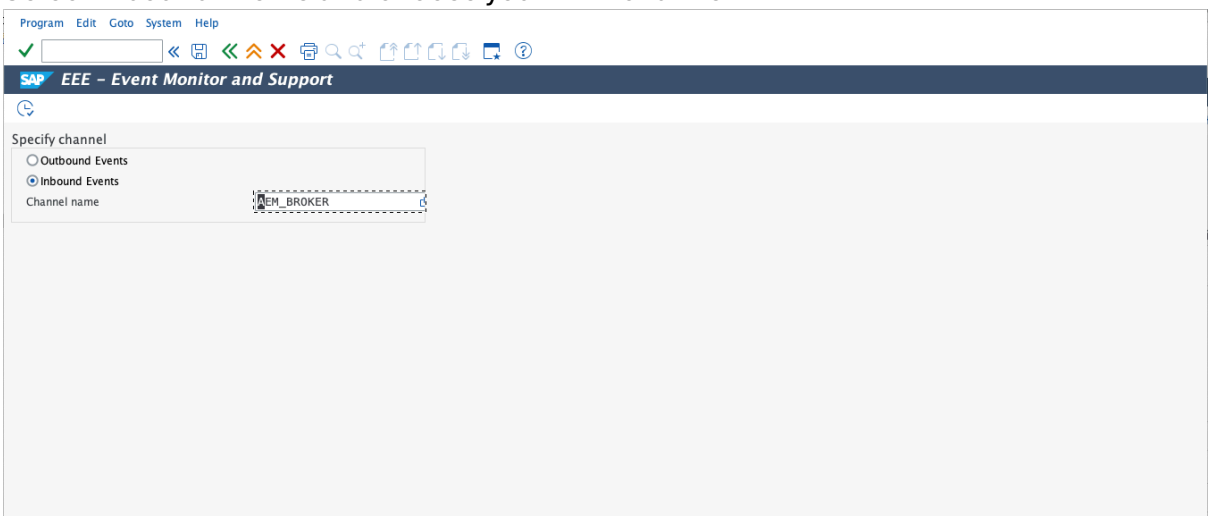
19. Enter the name of the queue you created. If you followed the steps of this tutorial it should be “Business_Partner”.

20. Save the subscription

With the subscription being configured events that are send to the AEM queue should be consumed by the S/4 system. Let’s test that

21. Go into transaction BP, select a business partner, edit it and save the changes. If the outbound binding and the queue has been properly configured as shown in this tutorial, this should trigger a Business Partner Changed event that should be send to the AEM. As a subscriber to the queue the event should then be received again by the S/4 system.
22. To check if the event has been properly received open transaction /IWXBE/EVENT_MONITOR.

23. Select Inbound Events and choose your AEM channel



24. Choose execute and view the received events