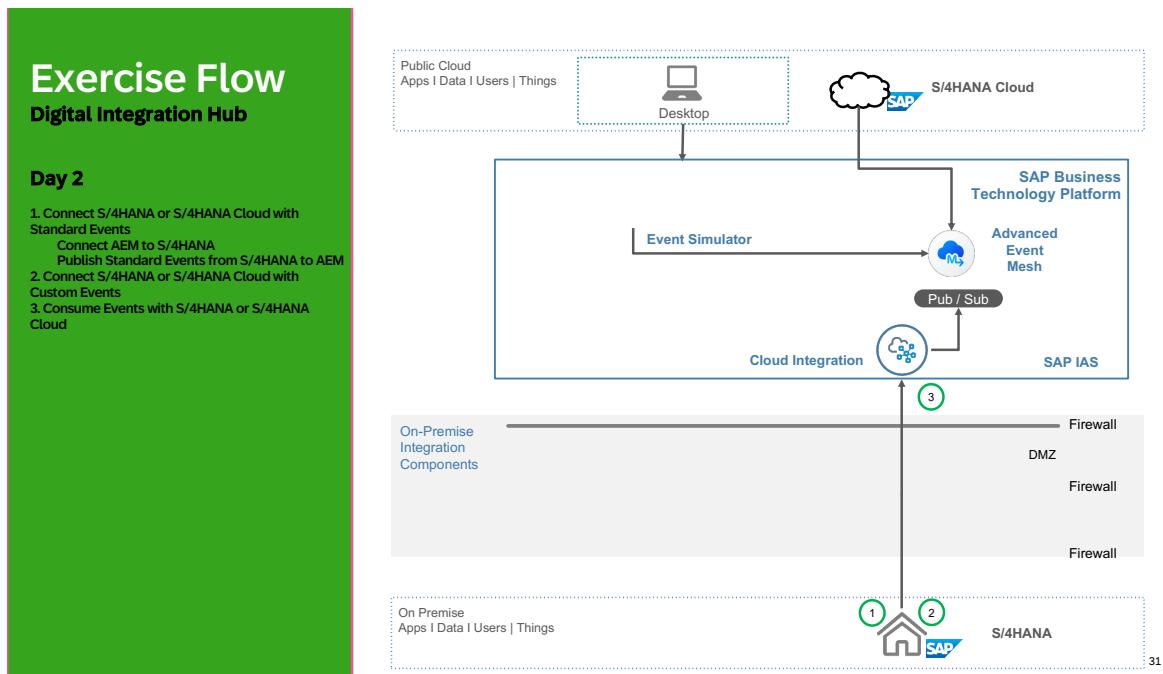


Topic 2

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

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

Welcome to SAP Integration Suite, advanced event mesh

Subaccount ID: aaa3d8bd-9746-4356-9fcd-4e7c731aaef78

Event Portal

- Designer**: Design your event driven architecture components.
- Catalog**: Browse or search for events, schemas, and applications across your enterprise.
- Runtime Event Manager**: Discover, govern, and visualize the flow of events between your applications in each environment.
- KPI Dashboard**: View metrics for your event use in your event driven architecture.

Mission Control

- Cluster Manager**: Control the lifecycle of your Event Broker Services. This option is highlighted with a red box.
- Mesh Manager**: Create and manage your event meshes.

Insights

- Select your broker

Cluster Manager: Services

Filter by service name Only show my services

Create Service BETA

All services

Broker 100	AEM_CommunityCentral	AEM_CommunitySubregion
Florian.lokes@sap.com	eks-eu-central-1a-2	aks-francecentral
Running

3. Select the “Connect” tab

The screenshot shows the SAP Cluster Manager interface. On the left is a sidebar with icons for Event Portal, Designer, Catalog, Runtime Manager, KPI, Mission Control, Cluster Manager (which is selected), Mesh Manager, and Insights. The main area is titled "Service Details: AEM_CommunityCentral". The top navigation bar has tabs: Status, Connect (highlighted with a red box), Manage, Monitoring, Configuration, and Try Me!. Below the tabs are three circular progress bars: Active Connections (16.0 %), Guaranteed Messaging Endpoints (16.0 %), and Queue Usage (0.0 %). The "Active Connections" bar shows counts for AMQP (5), SMF (11), MQTT (0), REST (0), and Web (0). The "Guaranteed Messaging Endpoints" bar shows 16 queues and 0 topic endpoints. The "Queue Usage" bar shows 12566 messages queued and 9 MB spool usage. Under "Availability and Versioning", it says the service is "Running" (Event Broker Service Version 10.8.1.126-1), created by Florian.okos@sap.com on 25/06/2024, 17:04:24, with a Service ID of 69afkf9w2od. "DMR Cluster" details include Hostname (mr-connection-h9tikb3o1b6w.messaging.solace.cloud), Cluster Name (cluster-eks-eu-central-1a-2-lvcu3jkbetw), Primary Router Name (69afkf9w2odprimary), and Cluster Password (redacted). "Management Access" shows Basic Authentication Enabled, LDAP Authentication Disabled, and SSD Authentication Disabled. "Distributed Tracing" indicates it is enabled. There is also a "View Tracing Details" link.

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

The screenshot shows the same SAP Cluster Manager interface as the previous one, but the "View by" dropdown menu is open, highlighting the "Protocol" option (also highlighted with a red box). The main content area lists connection points and supported client libraries:

- > Connect with Java
Solace JCSMP, Solace Java, Solace JMS over SMF, Paho over MQTT, QPID JMS 1.1 over AMQP, QPID JMS 2.0 over AMQP
- > Connect with C and C++
Solace C API over SMF, Paho over MQTT
- > Connect with Python
Solace Python API over SMF, Paho over MQTT
- > Connect with Go
Solace Go API over SMF
- > Connect with JavaScript
Solace Javascript API over SMF, Paho over MQTT
- > Connect with Node.js
Solace Node.js API over SMF, AMQPv10 Open Source over AMQP
- > Connect with .NET
Solace .NET API over SMF, Paho over MQTT
- > Connect with Spring
Spring Cloud Stream, Spring Boot

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

The screenshot shows the SAP Cluster Manager interface. In the top navigation bar, the 'Connect' tab is selected. Below it, under 'Service Details: AEM_CommunityCentral', there's a section titled 'Connect Using a Supported Client Library'. This section lists several connection points:

- Solace Messaging
- Solace Web Messaging
- AMQP** (highlighted with a red box)
- MQTT
- REST

For each connection point, there's a table with columns 'Technology', 'Library', and 'Language'. The 'AMQP' row shows:

Technology	Library	Language
Qpid	QPID JMS 1.1	Java
Qpid	QPID JMS 2.0	Java
AMQP	AMQP10 Open Source API	Node.js

Next to the 'AMQP' row, there are three 'Get Started' buttons, all of which are highlighted with red boxes.

6. Copy the “Host URLs”

This screenshot shows the same SAP Cluster Manager interface as the previous one, but with a different connection point selected: 'QPID JMS 1.1'. A modal window has opened on the right side of the screen, titled 'QPID JMS 1.1'. Inside this modal, under the 'Host URLs' section, a 'Secured AMQP URI' is displayed: `amqps://mconnection-h91kb3o1bw.messaging.solace.cloud:5671`. This URL is also highlighted with a red box.

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

The screenshot shows the SAP Configuration of RFC Connections interface. At the top, there is a toolbar with various icons. Below the toolbar, the title bar reads "SAP Configuration of RFC Connections". Underneath the title bar, there are three status indicators: "Generate RFC Callback Allowlist" (green), "Activate Non-Empty Allowlists" (yellow), and "Allowlist for Dynamic Connections" (blue). A note below the indicators says "RFC callback check not secure". The main area is a table titled "RFC Connections" with columns "Type", "PL Act.", and "Comment". The table lists the following connection types:

RFC Connections	Type	PL Act.	Comment
> ABAP Connections	3		
> HTTP Connections to External Server	G		
> HTTP Connections to ABAP System	H		
> Internal Connections	I		
> Logical Connections	L		
> TCP/IP Connections	T		
> WebSocket RFC	W		
> Connections Using ABAP Driver	X		

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

The screenshot shows the "Create Destination" dialog box. On the left, there are three colored circles (red, grey, green) with corresponding labels: "Destination" (grey), "Connection Type" (green), and an empty label (red). To the right, there are two input fields. The top field is labeled "Destination" and contains the text "AEM_BROKER_CONNECTION". The bottom field is labeled "Connection Type" and contains the text "G HTTP connection to external server" with a dropdown arrow icon. At the bottom right of the dialog are two buttons: a green checkmark button and a red X button.

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”

SAP RFC Destination AEM_BROKER_CONNECTION

Connection Test

RFC Destination	AEM_BROKER_CONNECTION	
Connection Type	G HTTP Connection to External Server	
Description	Description	
Description 1	Broker setting for AEM Broker h91kb3o1b6w	
Description 2		
Description 3		

Administration **Technical Settings** Logon & Security Special Options

Target System Settings

Host	tcp://mr-connection-h91kb3o1b6w.messaging.s...	Port	5671
Path Prefix			

HTTP Proxy Options

Global Configuration	
Proxy Host	
Proxy Service	
Proxy User	
Proxy PW Status	is initial
Proxy Password	

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

SAP RFC Destination AEM_BROKER_CONNECTION

Connection Test

RFC Destination	AEM_BROKER_CONNECTION	
Connection Type	G HTTP Connection to External Server	
Description	Description 1: Broker setting for AEM Broker h91kb3o1b6w Description 2: Description 3:	

Administration Technical Settings **Logon & Security** Special Options

Logon Procedure

Logon with User

Do Not Use a User OAuth Settings
 Basic Authentication

User:
PW Status:

Logon with Ticket

Do Not Send Logon Ticket
 Send Logon Ticket Without Ref. to a Target System
 Send Assertion Ticket for Dedicated Target System

System ID: Client

Logon with MQTT/AMQP

Users:
PW Status:
Password:

Security Options

Status of Secure Protocol

SSL Inactive Active

SSL Client PSE ID: SSL Client (Standard) ▼
 Do Not Use Certificate for Logon

TLS Attributes ▼

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”

SAP RFC Destination AEM_BROKER_CONNECTION

Connection Test

RFC Destination	AEM_BROKER_CONNECTION	
Connection Type	G HTTP Connection to External Server	
Description	Description 1: Broker setting for AEM Broker h91kb3o1b6w Description 2: Description 3:	

Administration Technical Settings **Logon & Security** Special Options

Logon Procedure

Logon with User

Do Not Use a User OAuth Settings

Basic Authentication

User:

PW Status:

Logon with Ticket

Do Not Send Logon Ticket

Send Logon Ticket Without Ref. to a Target System

Send Assertion Ticket for Dedicated Target System

System ID: Client

Logon with MQTT/AMQP

Users:

PW Status:

Password:

Security Options

Status of Secure Protocol

SSL: Inactive Active

SSL Client PSE ID:

Do Not Use Certificate for Logon

TLS Attributes

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 your system own certificate and export it as base64

SAP Trust Manager: Display

The screenshot shows the SAP Trust Manager interface. On the left, a tree view lists various SSL client configurations. In the center, the 'SSL client SSL Client (Standard)' section is selected. Under 'Own Certificate', the subject is listed as 'CN=T33_SSLC_DEFAULT, OU=ISAP-INTERN, OU=SAP Web AS, O=SAP Trust Community, C=DE' (Self-Signed). Below this, a 'Certificate List' table shows several certificates from DigiCert and Thawte. At the bottom, there are buttons for 'Verification PSE' and 'Password'. A red box highlights the 'Export' button (represented by a floppy disk icon) in the toolbar at the bottom.

17. Remember the common name (CN)

SAP Trust Manager: Display

This screenshot is identical to the previous one, showing the SAP Trust Manager interface. The 'Own Certificate' section is selected, displaying the same certificate details. However, the common name 'CN=T33_SSLC_DEFAULT...' is now highlighted with a red box. The rest of the interface, including the certificate list and export options, remains the same.

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

Event Broker Service Settings

Authentication
Enabled

Certificate Authorities
2 Client Certificate Authorities
1 Domain Certificate Authority

Client Profiles
1 Client Profile

Broker Manager Quick Settings

Message VPN Clients Queues Access Control Bridges

Other Management Tools

- SEMP - REST API
- Broker Manager - Web Application
- SolAdmin - Desktop Application

19. Enable “Client Certificate Authentication”

Service Authentication

Client Authentication

Type

Internal Database

Client Certificate Authentication

Validate Certificate Dates

OAuth Provider Authentication

Allow API Provided Username

Username Source

Common Name

LDAP Profile

Save

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

The screenshot shows the SAP Event Broker Service Settings page. The 'Manage' tab is selected. In the 'Event Broker Service Settings' section, there is a box labeled 'Certificate Authorities' containing '2 Client Certificate Authorities' and '1 Domain Certificate Authority'. This box is highlighted with a red border. Below this are sections for 'Broker Manager Quick Settings' (Message VPN, Clients, Queues, Access Control, Bridges) and 'Other Management Tools' (SEMP - REST API, Broker Manager - Web Application, SolAdmin - Desktop Application).

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

The screenshot shows the 'Certificate Authorities' page. The 'Client Certificate Authorities' tab is selected. A red box highlights the 'Add Client Certificate Authority' button. Below it is a table with two entries:

Certificate Authority Name	Manage
T33	[Edit] [Delete]
T33PFEILA	[Edit] [Delete]

22. Save the Certificate Authority

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

Service Details: AEM_CommunityCentral

Event Broker Service Settings

Broker Manager Quick Settings

Access Control (highlighted with a red box)

Other Management Tools

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

Client Username	Client Profile	ACL Profile	Enable	Subscription Manager	Dynamic
#client-username	#client-profile	#acl-profile	Yes	No	No
#rdp/BPA RDP	default	#acl-profile	Yes	No	No
T33_SSLC_DEFAULT	default	default	Yes	No	No
default	default	default	No	No	No
email-profile	default	mail	Yes	No	No
sc-dt-trace-collector	#telemetry-Tracing	#telemetry-Tracing	Yes	No	No
solace-cloud-client	default	default	Yes	No	No
t33_Purchaseorder	default	default	Yes	No	No

25. Enter the in step 17 mentioned common name (CN) (here T33_SSLC_DEFAULT) as Client Username and hit “Create”. Don’t forget to enable the username.

Action	Subscription Manager	Dynamic
Enable	No	No
Yes	No	No
Yes	No	No
No	No	No
Yes	No	No
Yes	No	No

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

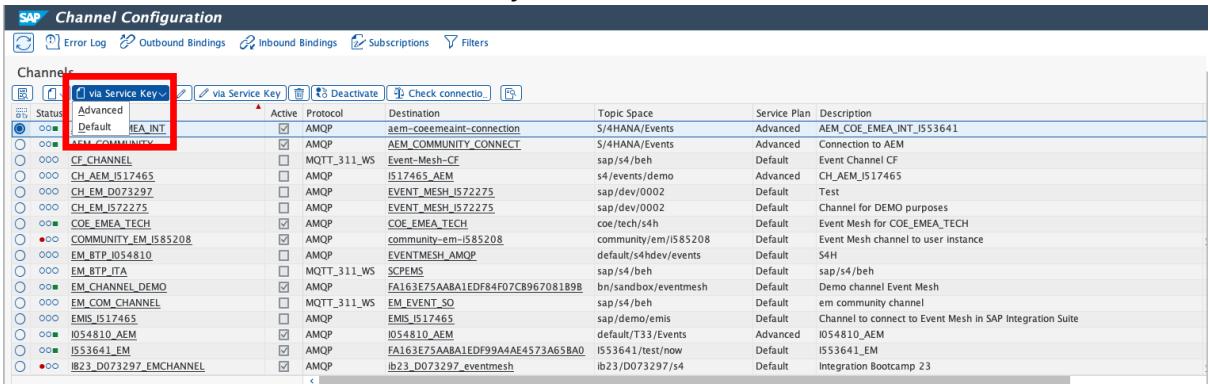
26. Open “Instances and Subscriptions” and select the instance of your AEM

27. Press “View Credentials” and copy the service key

Name	Created On	Modified On	Status
SK_aem-validation-service	25 Jun 2024	25 Jun 2024	Created

28. Go back to SAPGUI and open transaction /IWXB/E/CONFIG.

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



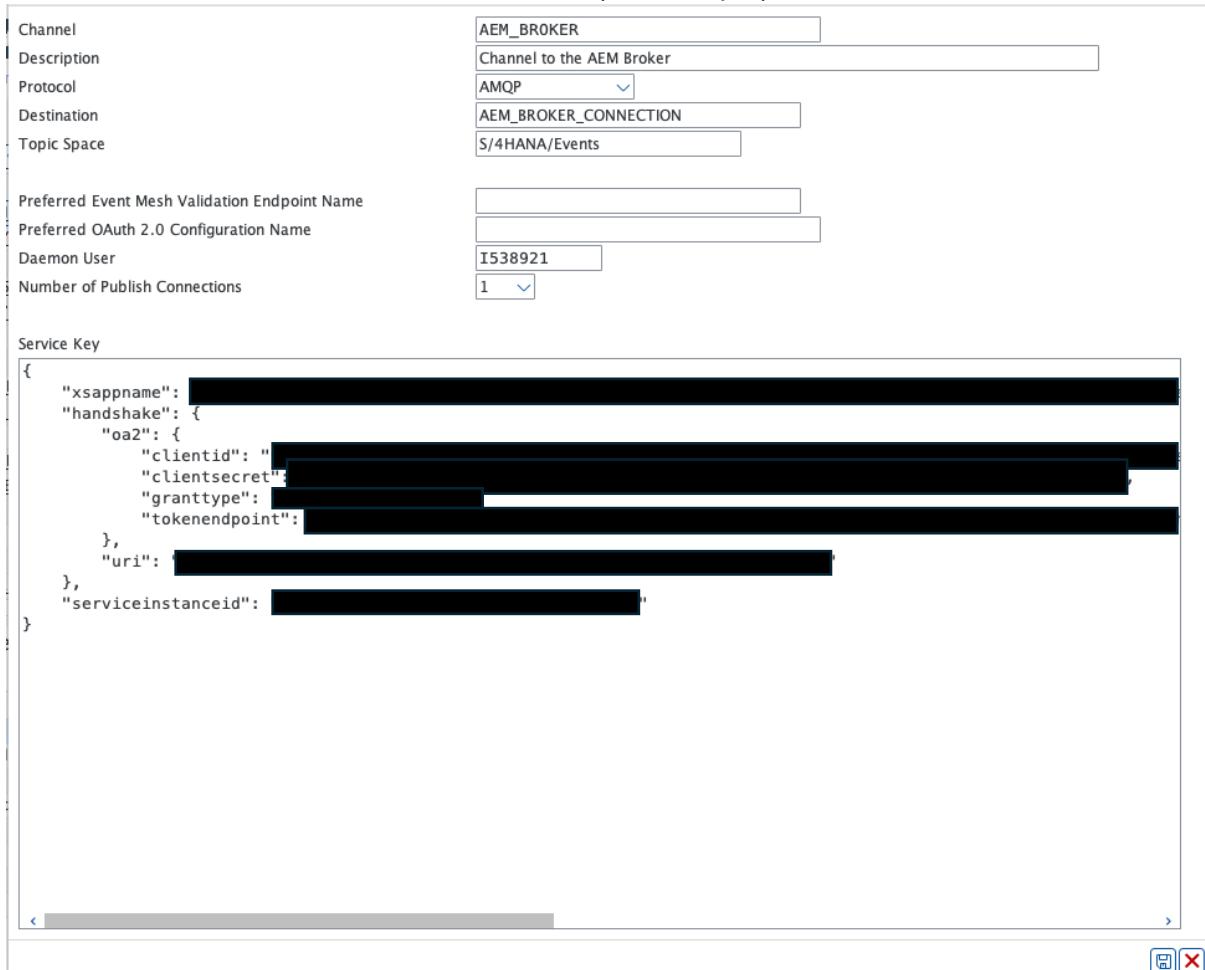
Channel Configuration							
		Error Log		Outbound Bindings		Inbound Bindings	
		Subscriptions		Filters			
Channels							
Status	Protocol	Active	Destination	Topic Space	Service Plan	Description	
Default [AEM_COMMUNITY]	AMQP	<input checked="" type="checkbox"/>	aem-coemeainit-connection	S/4HANA/Events	Advanced	AEM_COE_EMEA_INT_I53641	
CF_CHANNEL	MQTT_311_WS	<input type="checkbox"/>	Event-Mesh-CF	S/4HANA/Events	Advanced	Connection to AEM	
CH_AEM_I517465	AMQP	<input type="checkbox"/>	I517465_AEM	s4/events/demo	Default	Event Channel CF	
CH_EM_D073297	AMQP	<input type="checkbox"/>	EVENT_MESH_I572275	sap/dev/0002	Advanced	CH_AEM_I517465	
CH_EM_I572275	AMQP	<input type="checkbox"/>	EVENT_MESH_I572275	sap/dev/0002	Default	Test	
COE-EMEA_TECH	AMQP	<input type="checkbox"/>	COE_EMEA_TECH	coe/tech/s4h	Default	Channel for DEMO purposes	
COMMUNITY_EM_I585208	AMQP	<input checked="" type="checkbox"/>	community-em-i585208	community/em/i585208	Default	Event Mesh for COE_EMEA_TECH	
EM_BTP_I054810	MQTT_311_WS	<input type="checkbox"/>	EVENTMESH_AMQP	default/s4dev/events	Default	Event Mesh channel to user instance	
EM_BTP_ITA	AMQP	<input type="checkbox"/>	SCPEMS	sap/s4/beh	S4H		
EM CHANNEL DEMO	AMQP	<input checked="" type="checkbox"/>	FA163E75AABA1EDF84F07CB96708189B	br/sandbox/eventmesh	Default	Demo channel Event Mesh	
EM_COMM CHANNEL	MQTT_311_WS	<input type="checkbox"/>	EM_EVENT_SO	sap/s4/beh	Default	em community channel	
EMIS_I517465	AMQP	<input type="checkbox"/>	EMIS_I517465	sap/demo/emis	Default	Channel to connect to Event Mesh in SAP Integration Suite	
I054810_AEM	AMQP	<input checked="" type="checkbox"/>	I054810_AEM	default/t33/Events	Advanced	I054810_AEM	
I53641_EM	AMQP	<input checked="" type="checkbox"/>	FA163E75AABA1EDF994AE4573A65BA0	I53641/test/now	Default	I53641_EM	
IB23_D073297_EMCHANNEL	AMQP	<input checked="" type="checkbox"/>	ib23.D073297.eventmesh	ib23/D073297/s4	Default	Integration Bootcamp 23	

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)



Channel	AEM_BROKER
Description	Channel to the AEM Broker
Protocol	AMQP
Destination	AEM_BROKER_CONNECTION
Topic Space	S/4HANA/Events
Preferred Event Mesh Validation Endpoint Name	[redacted]
Preferred OAuth 2.0 Configuration Name	[redacted]
Daemon User	I538921
Number of Publish Connections	1
Service Key	<pre>{ "xsappname": "[redacted]", "handshake": { "oa2": { "clientid": "[redacted]", "clientsecret": "[redacted]", "granttype": "[redacted]", "tokenendpoint": "[redacted]" }, "uri": "[redacted]" }, "serviceinstanceid": "[redacted]" }</pre>

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

The screenshot shows the SAP Solace Clients interface. At the top, there are tabs for Clients Summary, Solace Clients, MQTT, REST, and AMQP. Below the tabs, it says "11 Client". There is a search bar with the placeholder "Search by name". A table lists two clients:

Name	Client Username	Subscriptions	Incoming Message Discards	Outgoing Message Discards	No Subscription Match	Client Address	Slow Subscriber
#amqp/client/341215995a9ea802/0AB4011...	T33_SSCLC_DEFAULT	0	0	0	0	130.214.228.42:29200	No
#amqp/client/3702b5b6d9611711/0AB40116...	T33_SSCLC_DEFAULT	0	0	0	0	130.214.228.42:48566	No

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 **/IWXB/E/OUTBOUND_CFG**
2. Select your AEM channel and click “Create new topic binding”

The screenshot shows the SAP Outbound Binding Configuration interface. The title bar says "SAP Outbound Binding Configuration". On the left, there is a sidebar titled "Channels" with sections for "Active Channels" and "Inactive Channels". Under "Active Channels", the "AEM_BROKER" channel is selected and highlighted with a red box. On the right, the main area is titled "Outbound Bindings of Channel AEM_BROKER" and shows a toolbar with icons for creating, deleting, and filtering bindings. A "Topic" input field is present at the bottom of the toolbar.

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

The screenshot shows the "Create Outbound Binding" dialog box. It has three tabs at the top: a red dot, a grey dot, and a green dot. The green dot is selected. The title bar says "Create Outbound Binding". The "Topic" field contains the value "sap/s4/beh/businesspartner/v1/BusinessPartner/Changed/*". At the bottom right of the dialog are "Save" and "Cancel" buttons.

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”

Cluster Manager > Service Details
Service Details: AEM_CommunityCentral Status Connect Manage Monitoring Configuration Try Me! Open Broker Manager ...

Event Broker Service Settings

Authentication Enabled

Certificate Authorities 2 Client Certificate Authorities 1 Domain Certificate Authority

Client Profiles 1 Client Profile

Broker Manager Quick Settings

Message VPN Clients Queues Access Control Bridges

Other Management Tools

SEMP - REST API The Solace Element Management Protocol (SEMP) is a REST API that you can use to manage the Event Broker Service. API

Broker Manager - Web Application The Broker Manager is a browser-based administration console that you can use to manage the Event Broker Service. Settings

SolAdmin - Desktop Application SolAdmin is a legacy desktop application that you can use to manage the Event Broker Service. Desktop

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

Create Queue

Queue Name

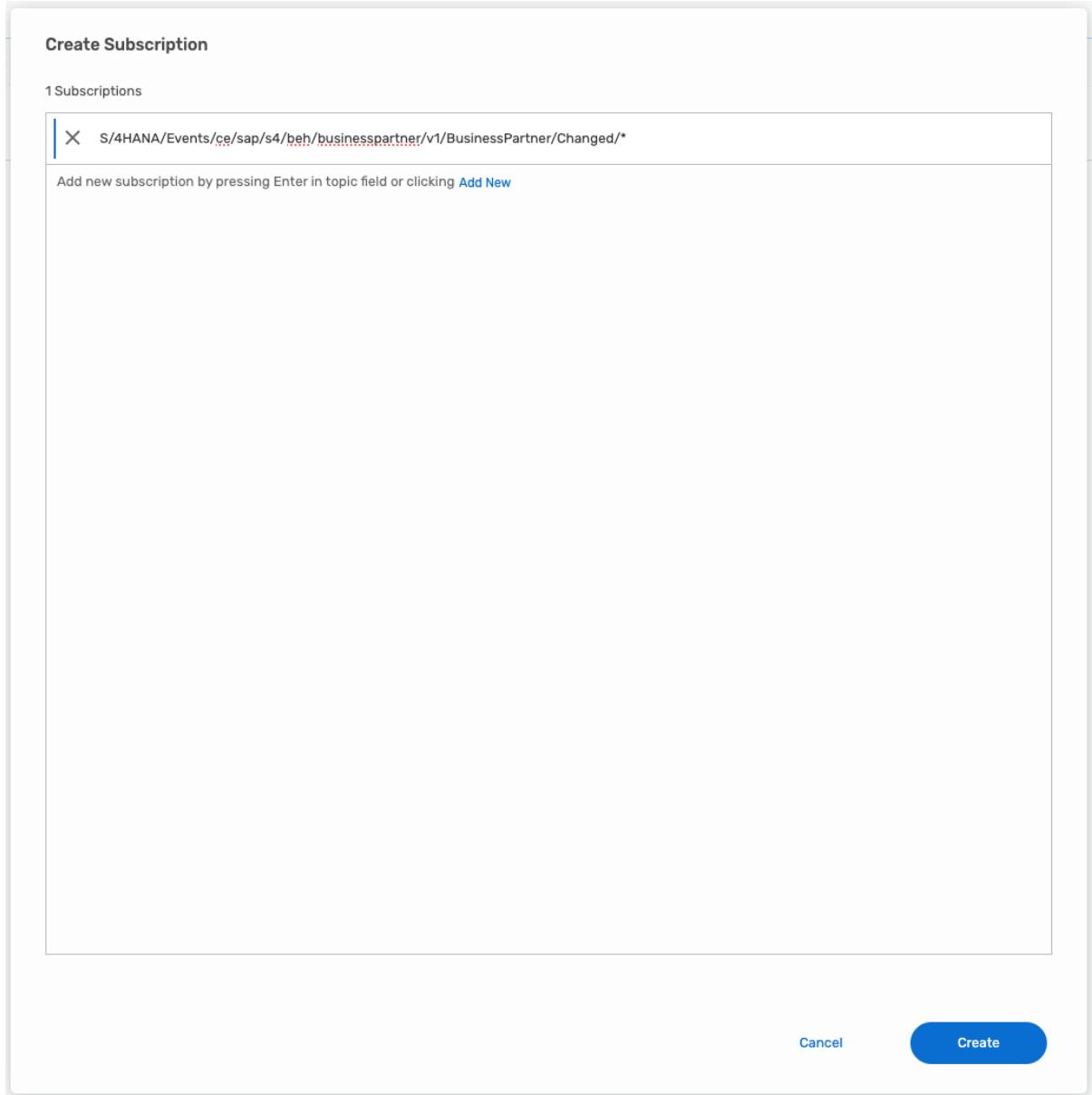
Business_Partner

Cancel Create

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”

✓ T33_800_i538921_en [T33, 800, I538921, EN]

> Local Objects (\$TMP)

> Favorite

> Favorites

> System

> Application

> Release

New

Duplicate Tree...

Show In

Copy

⌘ C

Delete

⌦

Add Package...

Refresh

F5

Coverage As

>

Coverage As

>

Run As

>

Debug As

>

Profile As

>

Expand Tree by...

Configure Tree... ⌘ I

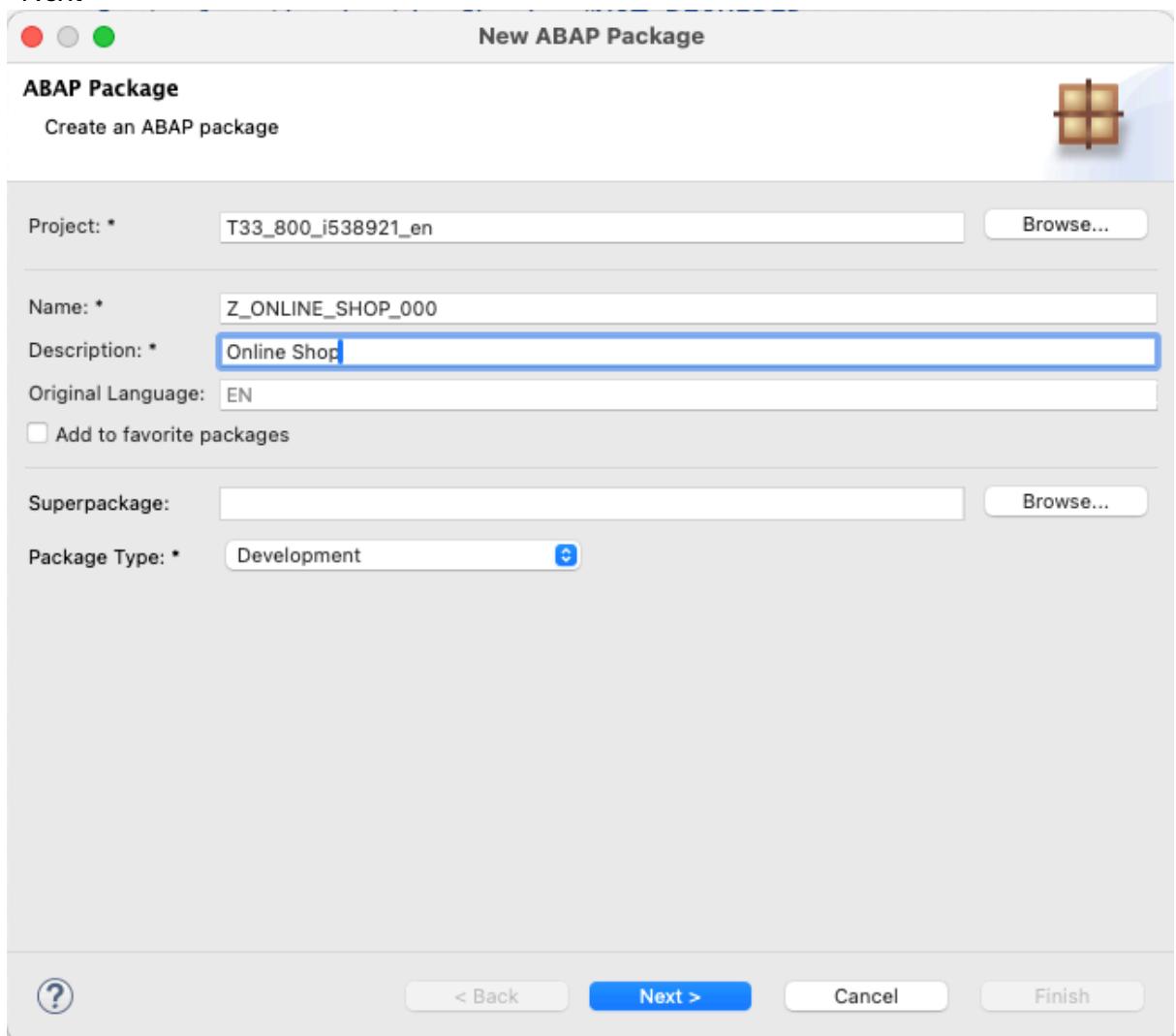
ABAP Class

ABAP Interface

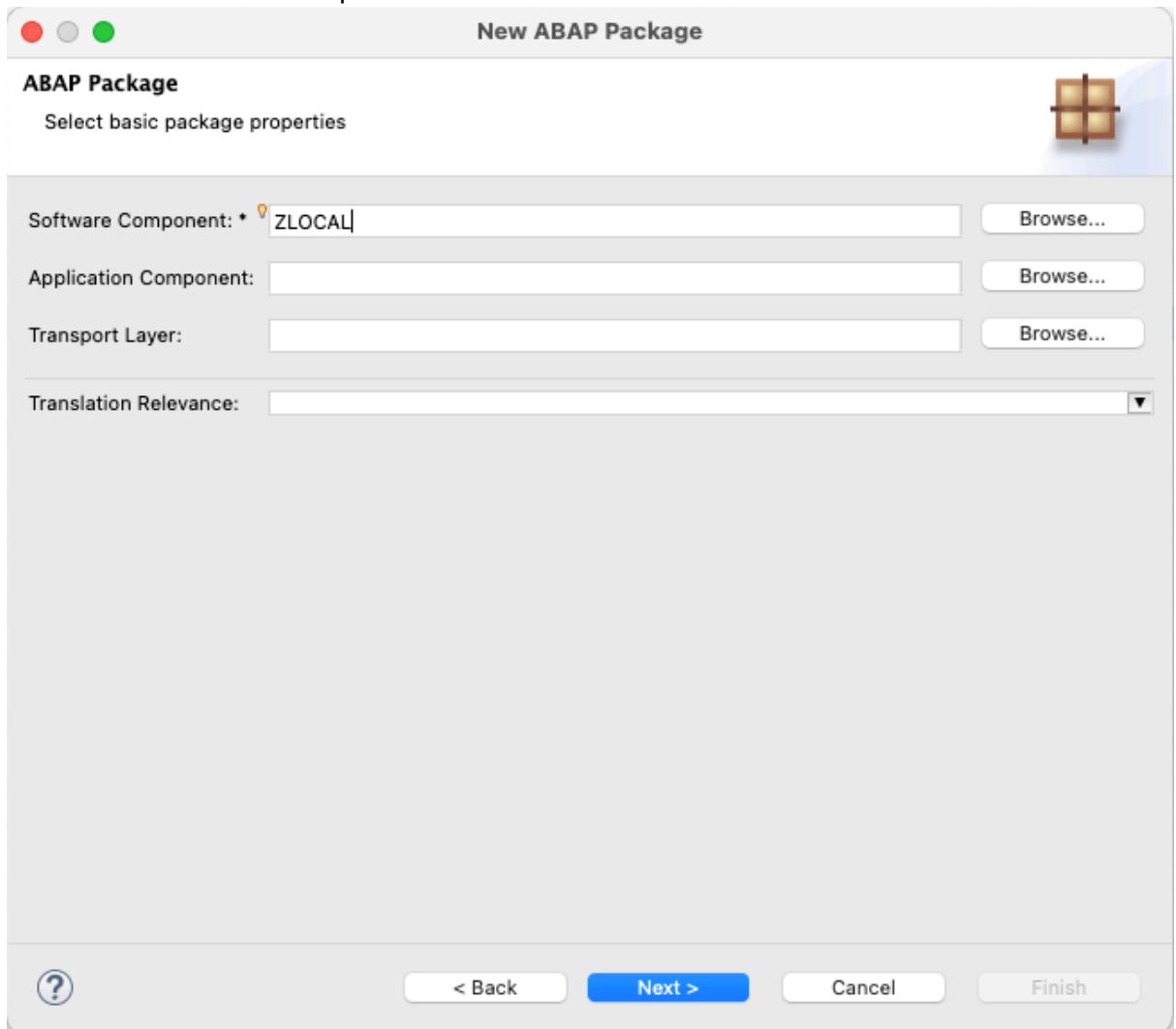
ABAP Package

Other ABAP Repository Object

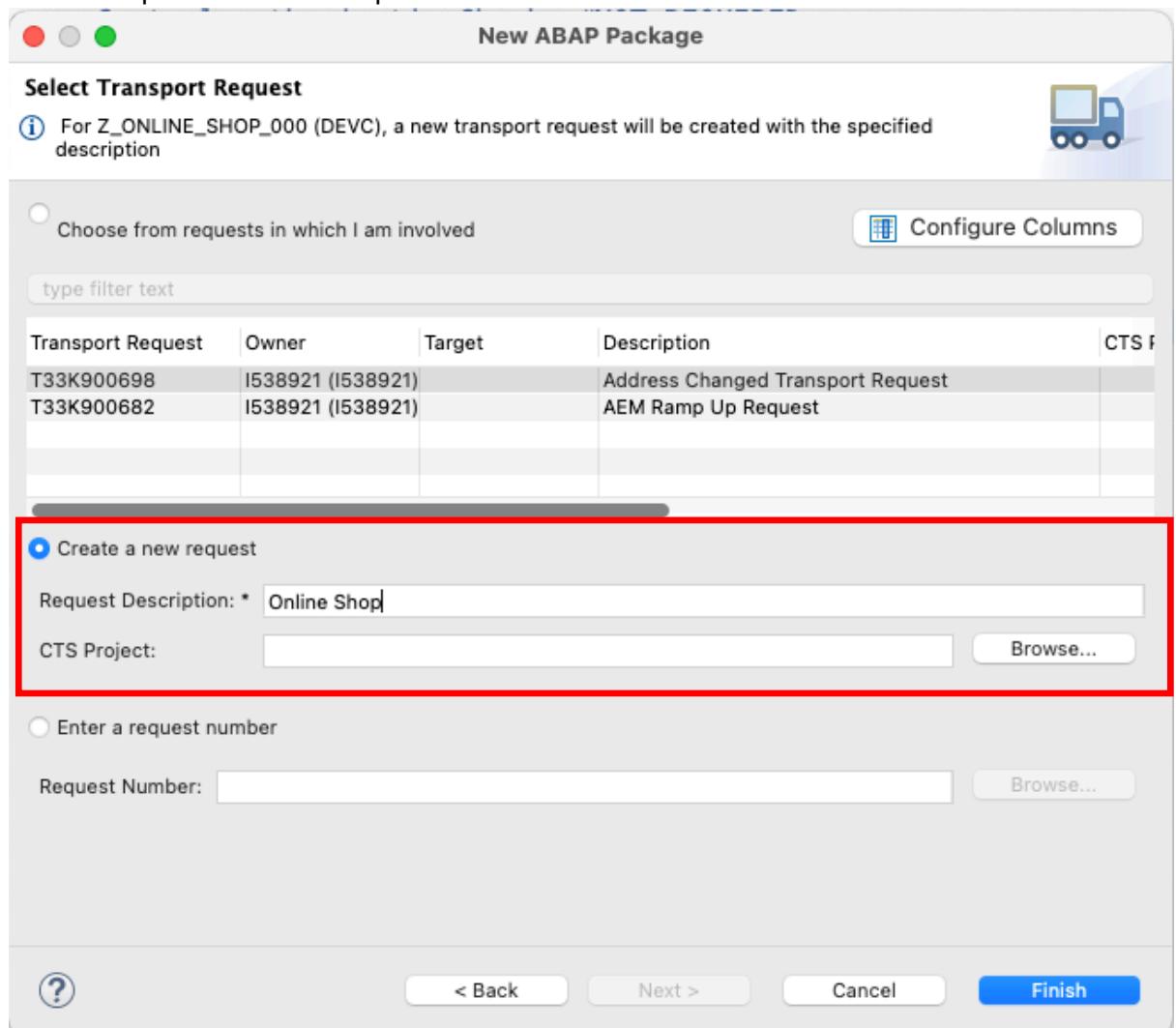
2. Give the name “Z_ONLINE_SHOP_000”, the description “Online Shop” and hit “Next”



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



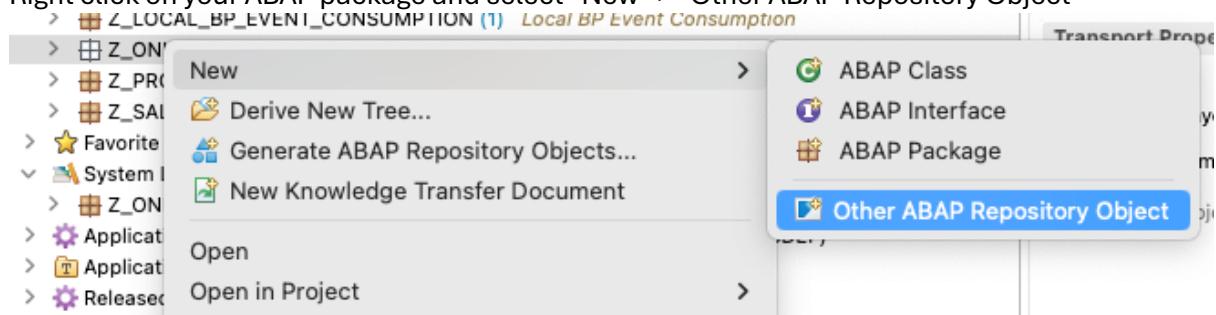
- Create a new transport request by selecting “Create a new request” and giving the description “Online Shop”



- Press “Finish”.

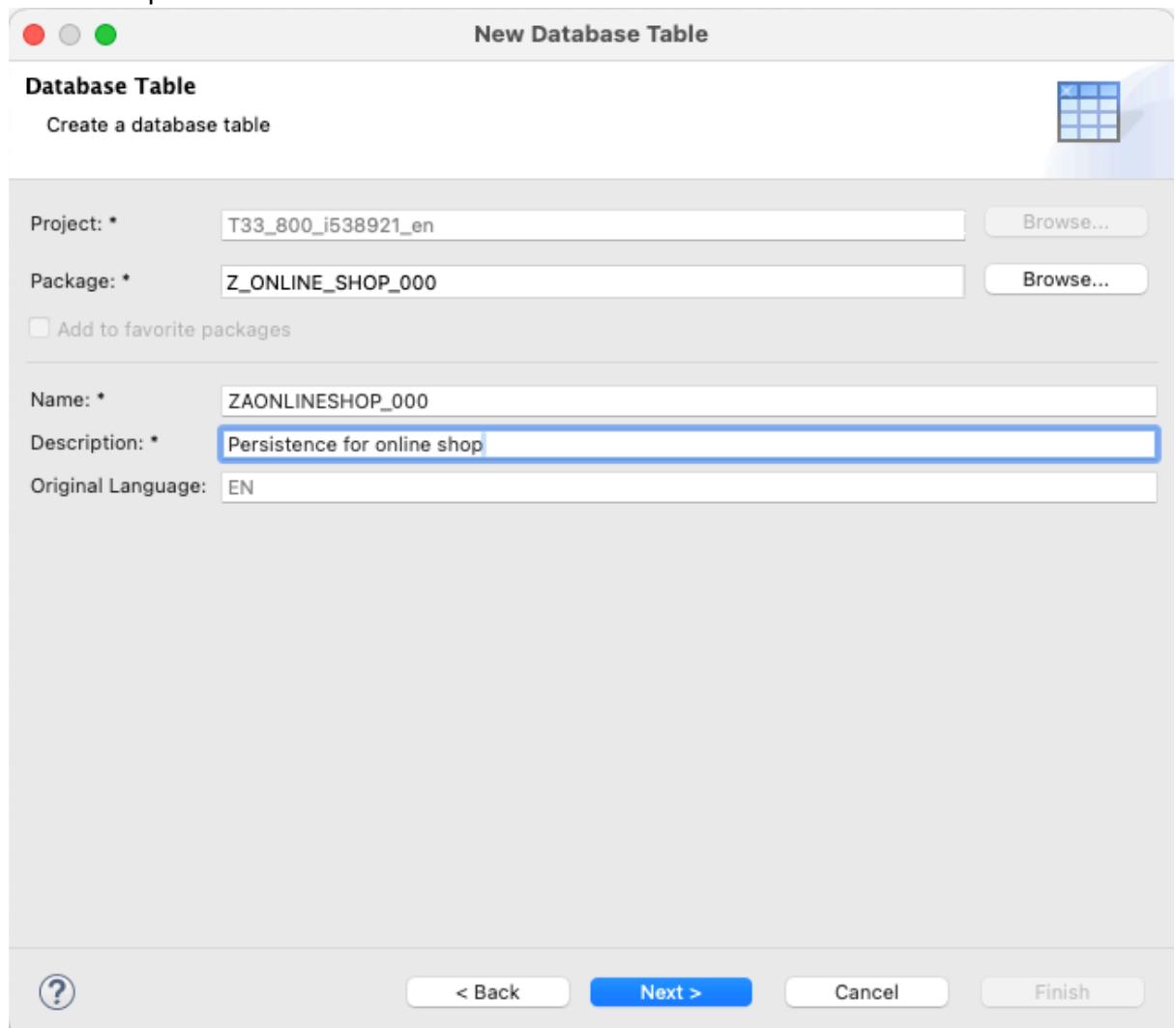
Lets create a new database table

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



- 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

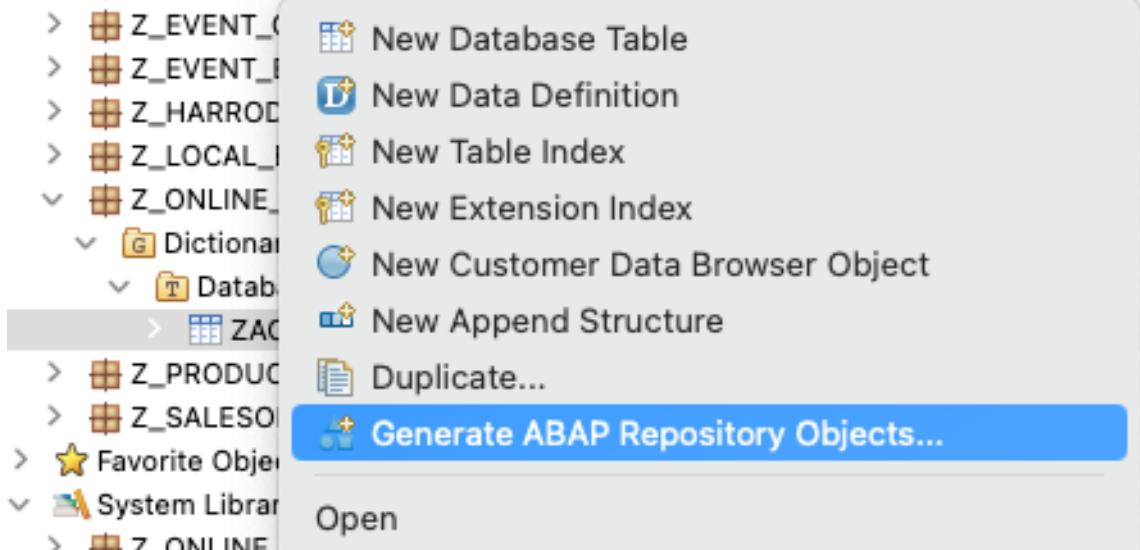
```
@EndUserText.label : 'Persistence for online shop'  
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE  
@AbapCatalog.tableCategory : #TRANSPARENT  
@AbapCatalog.deliveryClass : #A  
@AbapCatalog.dataMaintenance : #RESTRICTED  
define table zaonlineshop_000 {  
  
key client : abap.clnt not null;  
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”

The screenshot shows the SAP Fiori Launchpad interface for selecting a generator. At the top, there's a header with the SAP logo and a search bar labeled "Generate multiple repository objects". Below the header, there are two input fields: "Project: * T33_800_i538921_en" and "Referenced Object: * ZAONLINESHOP_000". A "Browse..." button is also present. On the left, a tree view shows categories like "ABAP RESTful Application Programming Model" and "Business Configuration Management". Under "ABAP RESTful Application Programming Model", "OData UI Service" is selected and highlighted in blue. To the right of the tree, there's a detailed description of the "OData UI Service". It explains that an OData UI service makes it possible to consume a RAP service with a Fiori Elements UI or other UI clients. It creates all UI-specific information that is annotated in the RAP artifacts in the OData service metadata. A UI service can be previewed with the Fiori Elements App Preview in the service binding artifact. The description continues to say that this generator creates all the development objects that are relevant for a business object and for a RAP service on the basis of one database table. The end result is a full-blown RAP UI service. It lists the objects created: CDS view entity, CDS behavior definition, ABAP implementation class, CDS projection view entity, CDS projection behavior definition, Draft database table, Metadata extension, CDS service definition, and Service binding. At the bottom of the screen, there are navigation buttons: a question mark icon, "< Back", "Next >" (which is highlighted in blue), "Cancel", and "Finish".

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	Artifact Names
Busienss 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”

The screenshot shows the SAP Fiori Launchpad with the title "Service Binding: ZUI_ONLINESHOP_000_O4". The "General Information" section indicates a "Binding Type: OData V4 - UI". The "Services" section shows a table with one row: "Ser Version" (1.0.0) and "AF Service Definition" (Nc_ZUI_ONLINESHOP_000). The "Service Version Details" section notes that the local service endpoint is not published and provides a link to "Publish local service endpoint". A red box highlights the "Publish" button in the top right corner of the main content area.

19. Double-click on the entity “OnlineShop” in the “Entity Set and Association” section to open the Fiori elements App Preview.

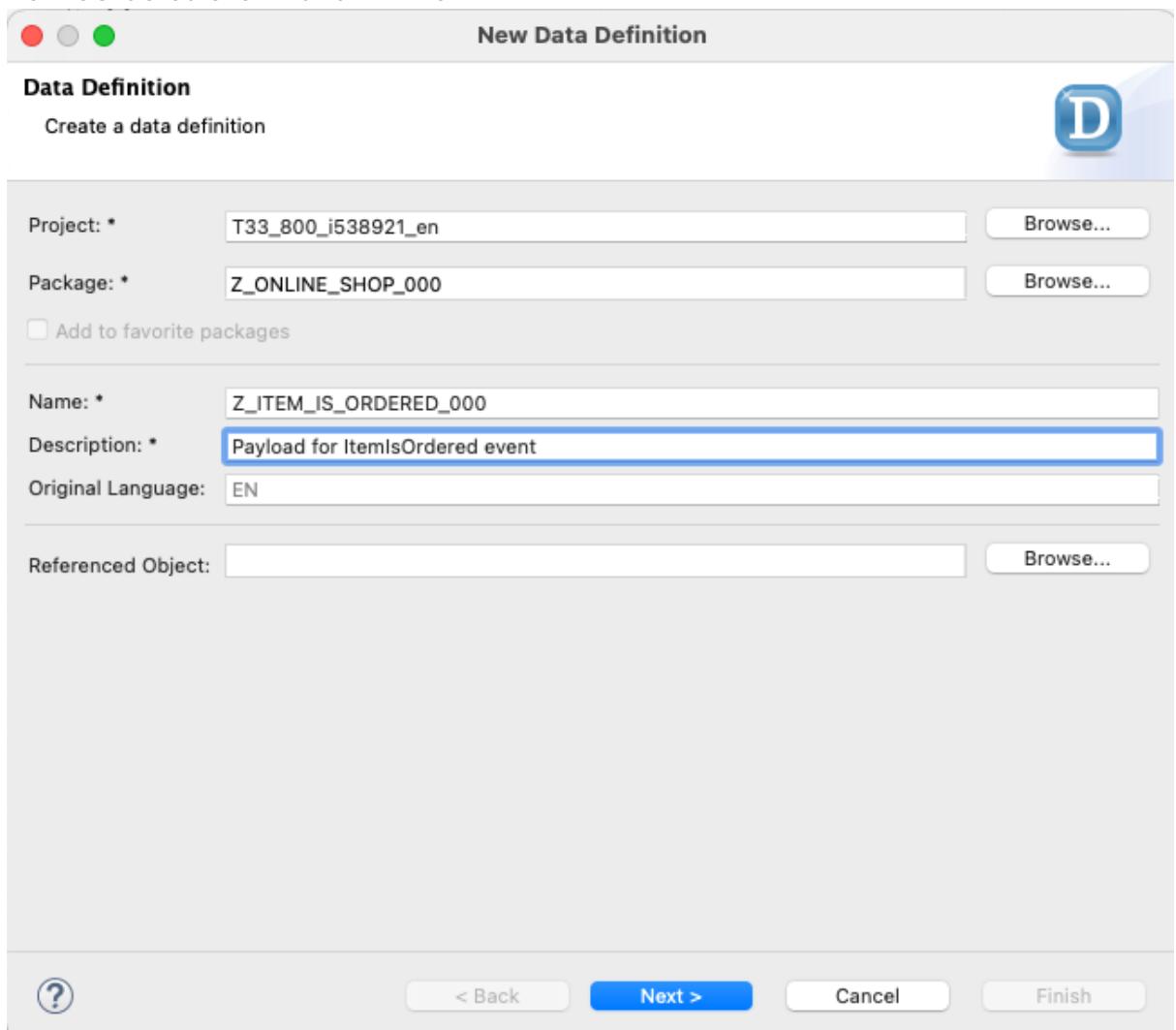
20. Use your S/4 login credentials to authenticate

The screenshot shows the Fiori App Preview for the "OnlineShops" entity set. The table header includes columns: OrderID, Ordereditem, Deliverydate, and Creationdate. A message at the bottom of the table area states: "No data found. Try adjusting the search or filter parameters." The top navigation bar includes "Standard", "Go", "Adapt Filters (1)", and other standard Fiori icons.

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.

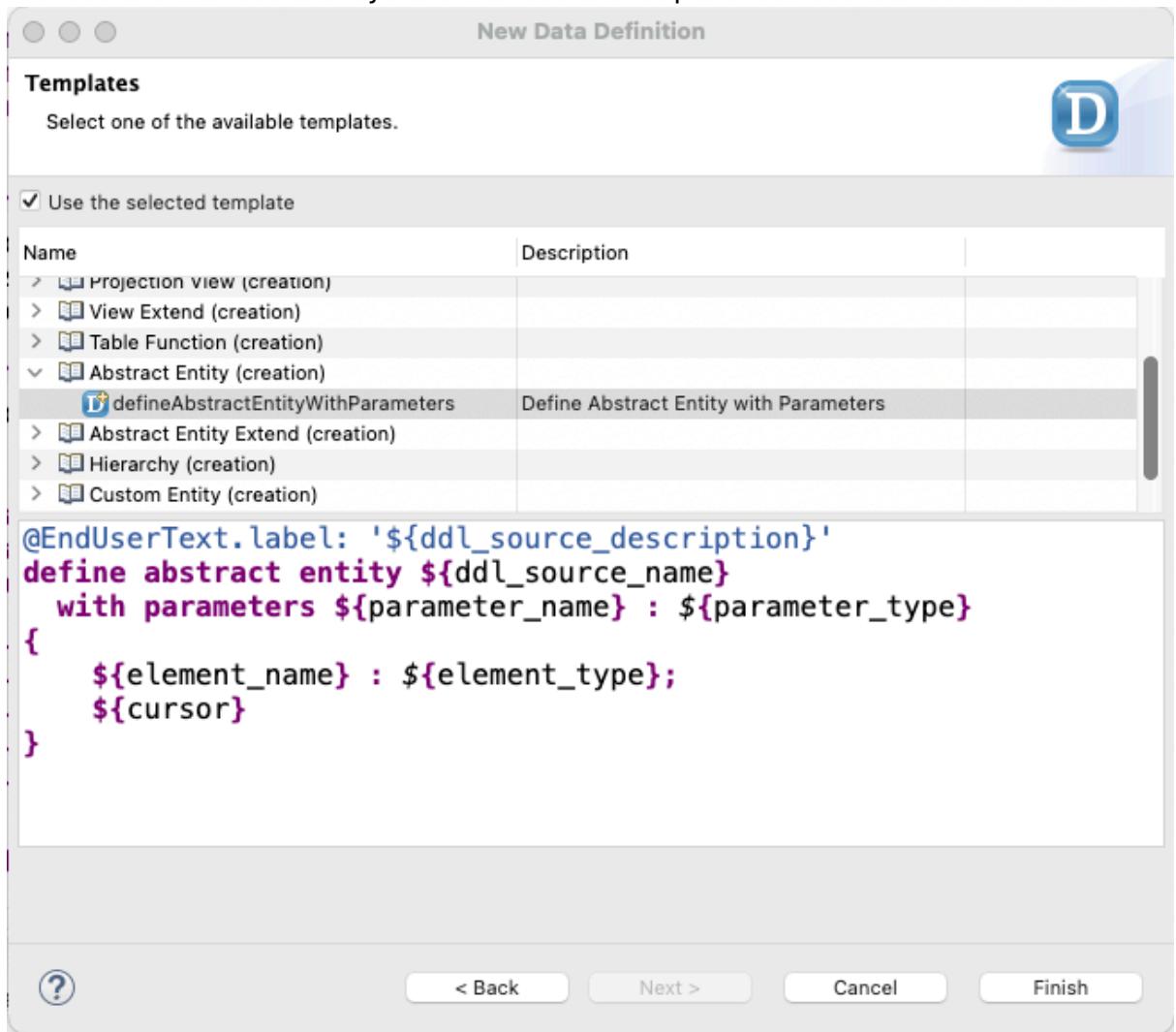
21. Right click your package Z_ONLINE_SHOP_000 and select “New” > “Other ABAP Repository Object”

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”

```
1 managed 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;
27
28     draft action Edit;
29     draft action Activate optimized;
30     draft action Discard;
31     draft action Resume;
32     draft determine action Prepare;
33
```

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 X
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 cursor 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 X [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 zaonlineshop_000
7 draft table ZDONLINESHOP_000
8 etag master LocalLastChanged
9 lock master total etag LastChanged
10 authorization master
11
12 {
13   field ( readonly )
14   OrderUUID,
15   LastChanged,
16   LocalLastChanged;
17
18   field ( numbering : managed )
```

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.

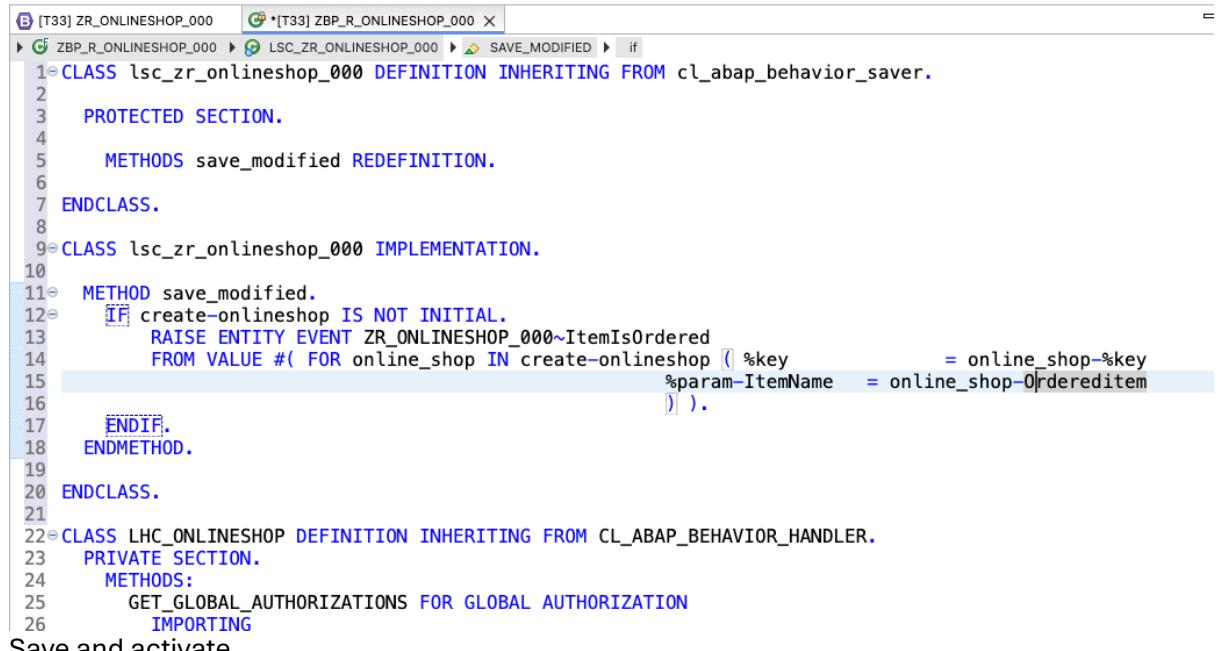
IF create-onlineshop IS NOT INITIAL.

```

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



```

1 CLASS lsc_zr_onlineshop_000 DEFINITION INHERITING FROM cl_abap_behavior_saver.
2   PROTECTED SECTION.
3   METHODS save_modified REDEFINITION.
4   ENDCLASS.
5
6 CLASS lsc_zr_onlineshop_000 IMPLEMENTATION.
7
8   METHOD save_modified.
9     IF create-onlineshop IS NOT INITIAL.
10       RAISE ENTITY EVENT ZR_ONLINESHOP_000~ItemIsOrdered
11         FROM VALUE #( FOR online_shop IN create-onlineshop ( %key      = online_shop-%key
12           %param-ItemName = online_shop-Ordereditem
13         )).
14       ENDIF.
15     ENDMETHOD.
16
17   ENDCLASS.
18
19 CLASS LHC_ONLINESHOP DEFINITION INHERITING FROM CL_ABAP_BEHAVIOR_HANDLER.
20   PRIVATE SECTION.
21   METHODS:
22     GET_GLOBAL_AUTHORIZATIONS FOR GLOBAL AUTHORIZATION
23       IMPORTING
24
25
26

```

35. Save and activate

With that the event is triggered whenever a new order is placed. However it is not yet sent 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”

New Event Binding

Event Binding

Create Event Binding

Project: * T33_800_i538921_en

Package: * Z_ONLINE_SHOP_000

Add to favorite packages

Name: * Z_EB_ONLINE_SHOP_000

Description: * Event Binding for Online Shop ItemIsOrdered

Original Language: EN

< Back Cancel

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..”

Event Binding: Z_EB_ONLINE_SHOP_000 No Event defined

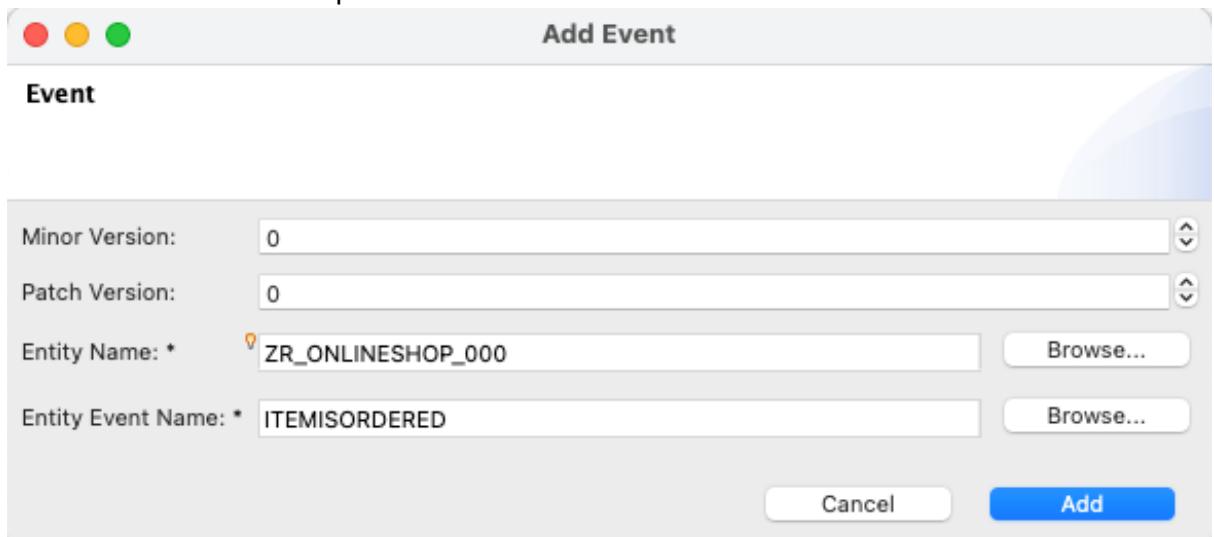
General Information

Type Namespace: *	zcustom000
SAP Object Type: *	OnlineShop
Operation: *	created
Type:	

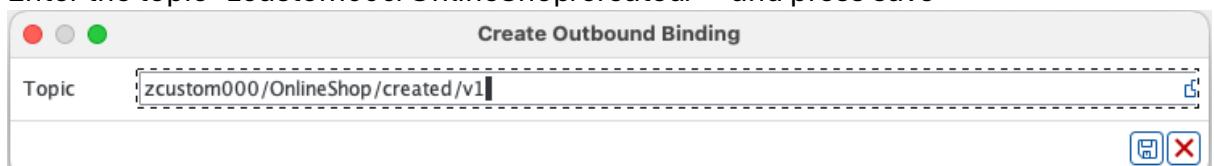
Event Versions

Major Version	Minor Version	Patch Version	Entity Name	Entity Event Name	
					<input type="button" value="Add..."/>
					<input type="button" value="Edit..."/>
					<input type="button" value="Remove"/>

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

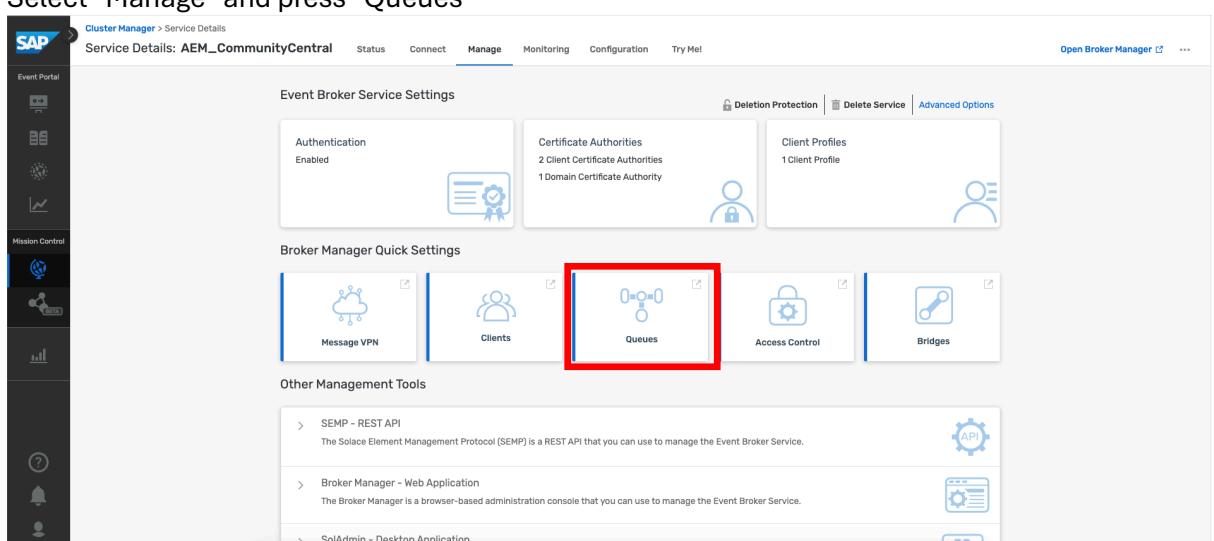


43. Activate the event binding.
 44. Next go into transaction “/IWXB/E/OUTBOUND_CFG” and select the channel to your AEM that has been previously set up
 45. Press “Create new topic binding”
 46. Enter the topic “zcustom000/OnlineShop/created/*” and press save



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”

Create Subscription

1 Subscriptions

X S/4HANA/Events/ce/zcustom000/OnlineShop/created/v1

Add new subscription by pressing Enter in topic field or clicking Add New

Cancel Create

With that the event `ItemIsOrdered` will be triggered whenever a new order is created and sent 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

Standard

Editing Status:

All

OnlineShops

OrderID	Ordereditem	Deliverydate	Creationdate
---------	-------------	--------------	--------------

To start, set the relevant filters and choose "Go".

Create Delete | Go Adapt Filters (1)

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

The screenshot shows the SAP Fiori OnlineShop application interface. At the top, there is a header bar with the title "OnlineShop". Below the header, there is a form with four input fields: "OrderID" (containing "1"), "Ordereditem" (containing "Item 1"), "Deliverydate" (containing "e.g. Dec 31, 2024"), and "Creationdate" (containing "e.g. Dec 31, 2024"). At the bottom right of the form, there are buttons for "Draft updated", "Create", and "Discard Draft".

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

The screenshot shows the SAP Fiori AEM_CommunityCentral application interface. On the left, there is a sidebar with navigation options: "AEM_CommunityCentral", "Change VPN", "Messaging", "Clients", "Queues" (which is selected and highlighted in blue), and "Connector Wizards", "Access Control". The main area is titled "Queues | Online Shop" and shows a table with one message listed under "Messages Queued". The table columns are: Message ID (55245), Spooled Time (2024-09-22 23:04:35), Content Size (B) (0), Attachment Size (B) (528), Priority (Yes), Undelivered (0), Redeliveries (Yes), DMO Eligible (Yes), and Replication State (Not replicated). There is also a "Action" button at the top right of the table.

Message ID	Spooled Time	Content Size (B)	Attachment Size (B)	Priority	Undelivered	Redeliveries	DMO Eligible	Replication State
55245	2024-09-22 23:04:35	0	528	Yes	0	Yes	Yes	Not replicated

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”

New Data Definition

Data Definition

Create a data definition

Project: * T33_800_i538921_en

Package: * Z_ONLINE_SHOP_000

Add to favorite packages

Name: * Z_ITEM_IS_ORDERED_DEEP_000

Description: * Deep structure for ItemIsOrdered event

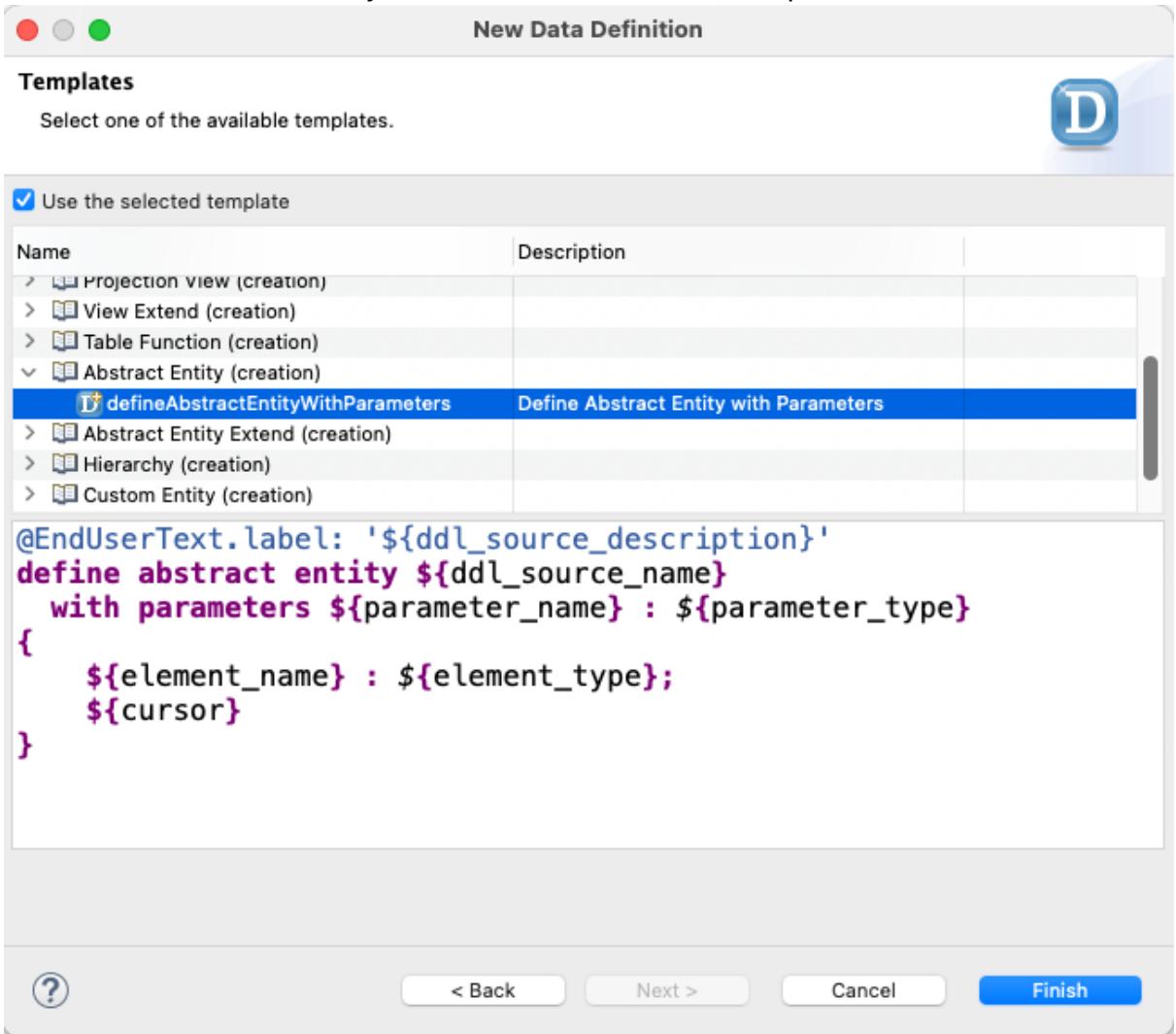
Original Language: EN

Referenced Object:

< Back **Next >** Cancel Finish

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_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”

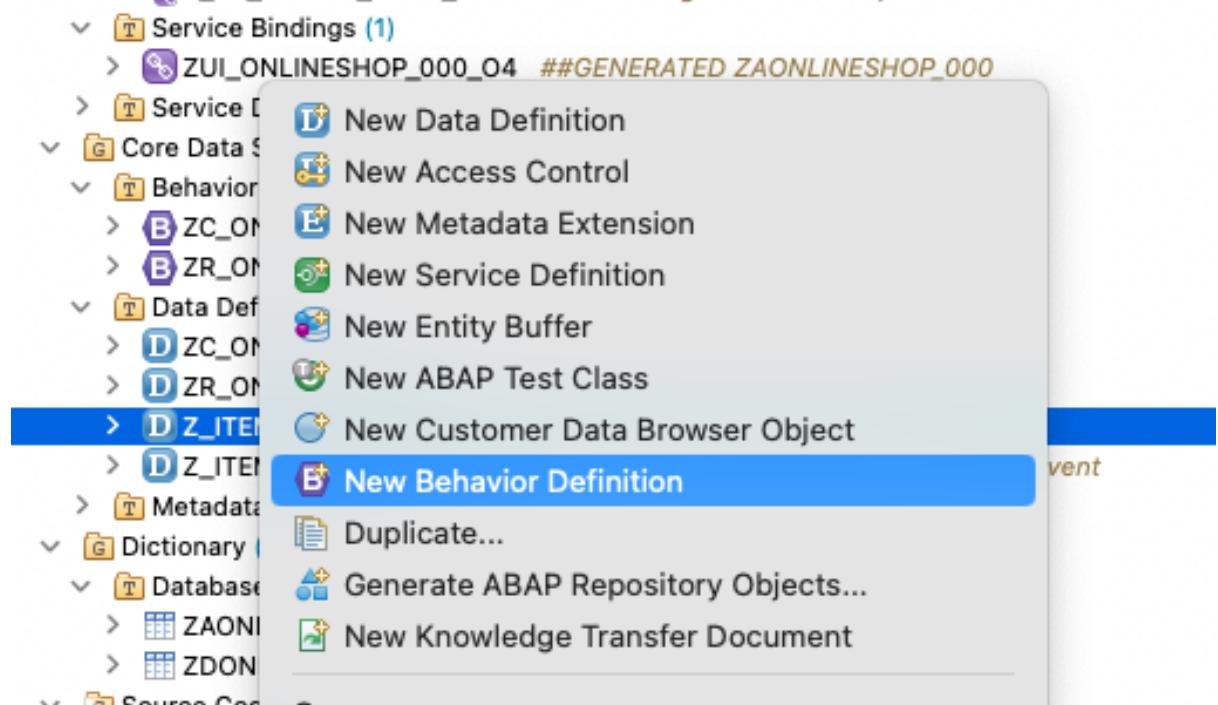
```

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 “/IWXB/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

Event ID	Arrival Time	Publish Time	Acknowledged Time	Dynamic Topic Segments
FA163E75AABA1EDFAA95A7237AED7BB5	22.11.2024 08:3...	22.11.2024 08:3...	22.11.2024 08:3...	
FA163E75AABA1EDFAA96EFFD4D40B8B5	22.11.2024 09:5...	22.11.2024 09:5...	22.11.2024 09:5...	

```
{
  "type": "zcustom000.OnlineShop.created.v1",
  "specversion": "1.0",
  "source": "/default/sap.s4.custom/T33CLNT800",
  "id": "fa163e75-aaba-1edf-aa96-effd4d40bbb5",
  "time": "2024-11-22T09:52:36Z",
  "datacontenttype": "application/json",
  "data": {
    "OrderUUID": "fa163e75-aaba-1edf-aa96-ef295ffa1bb5",
    "ItemName": "Item 4",
    "dates": {
      "Deliverydate": "2024-11-28",
      "Creationdate": "2024-11-22"
    }
  }
}
```

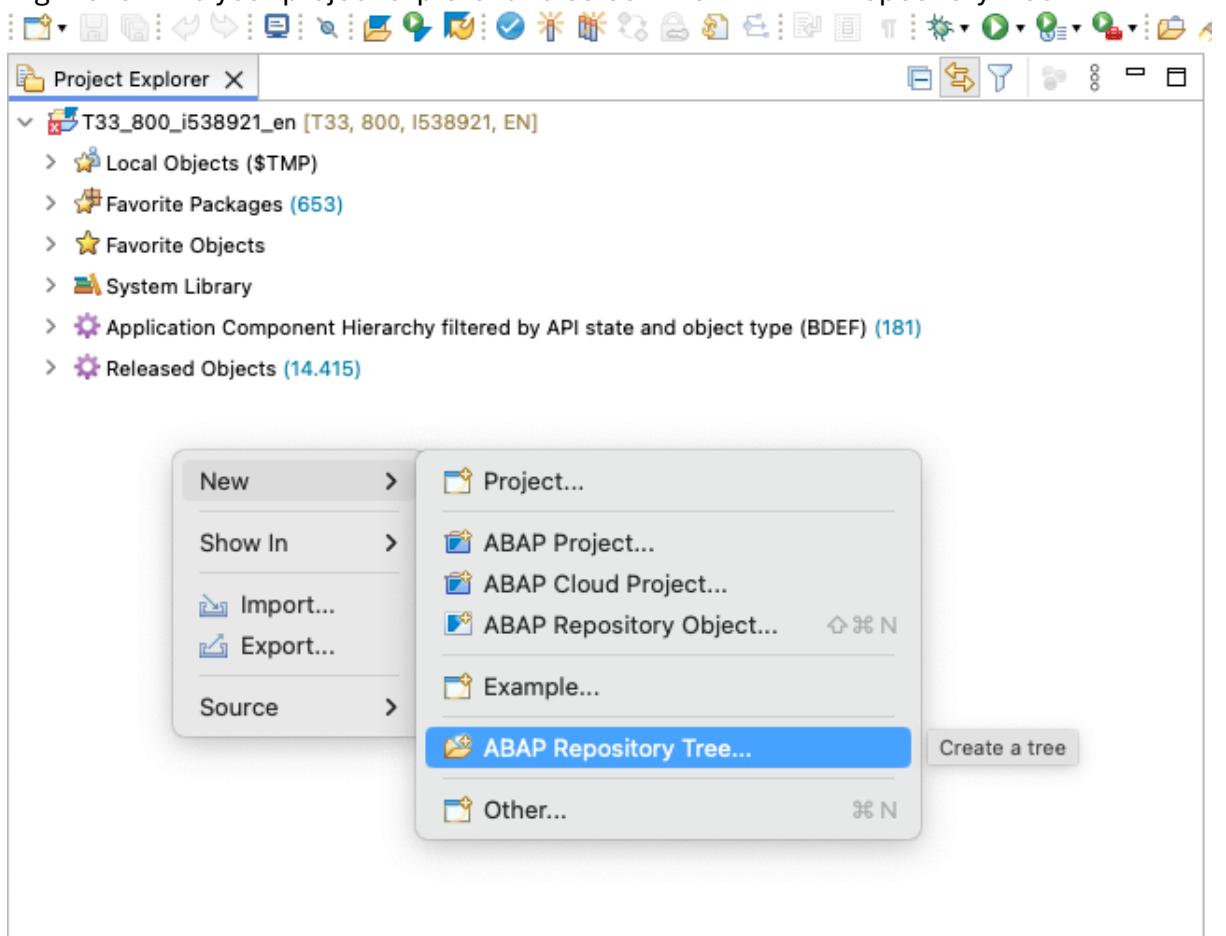
Customizing standard events using derived events

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.

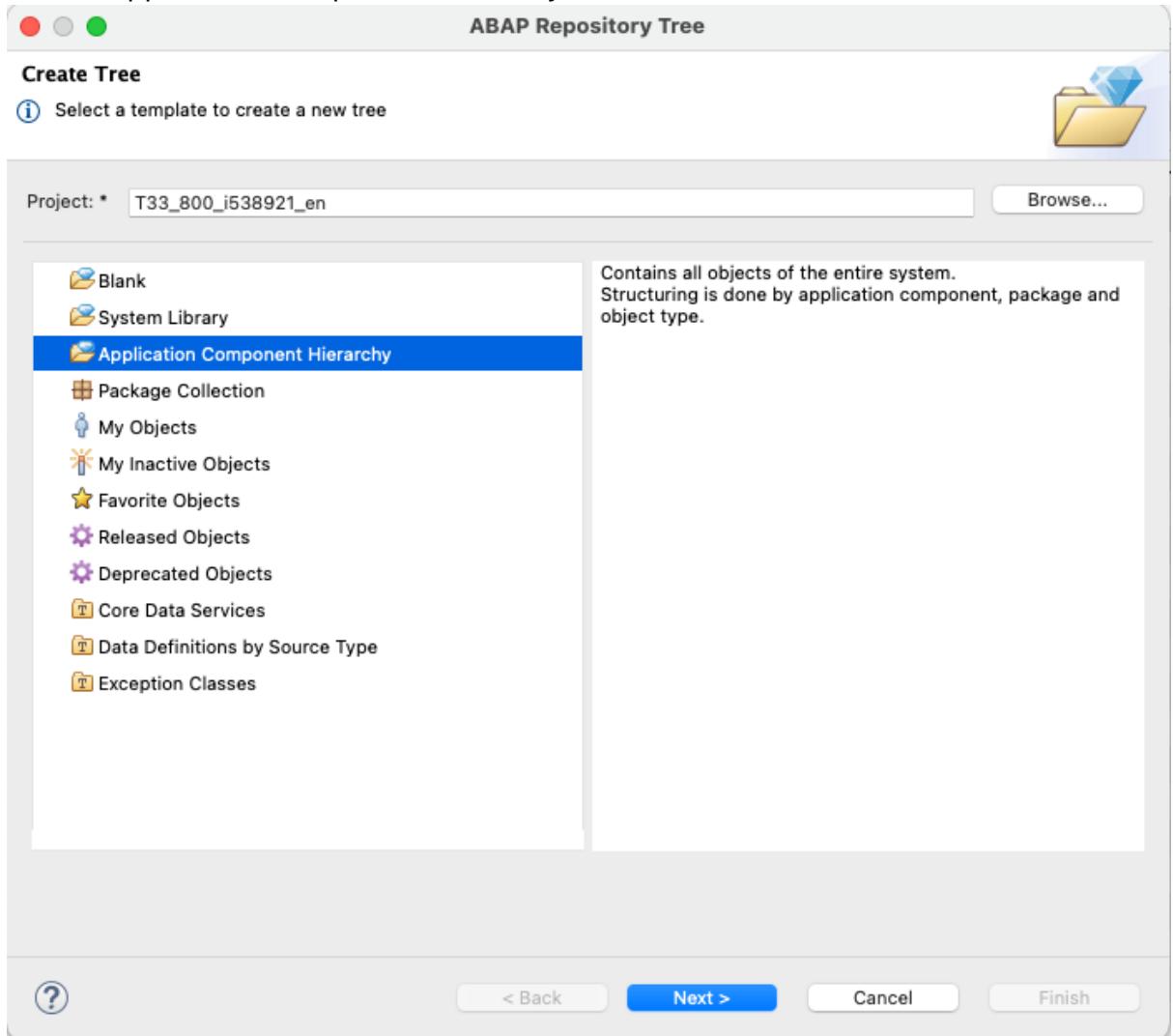
Let's look at how such a derived event can be created for the Business Partner Changed event.

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”

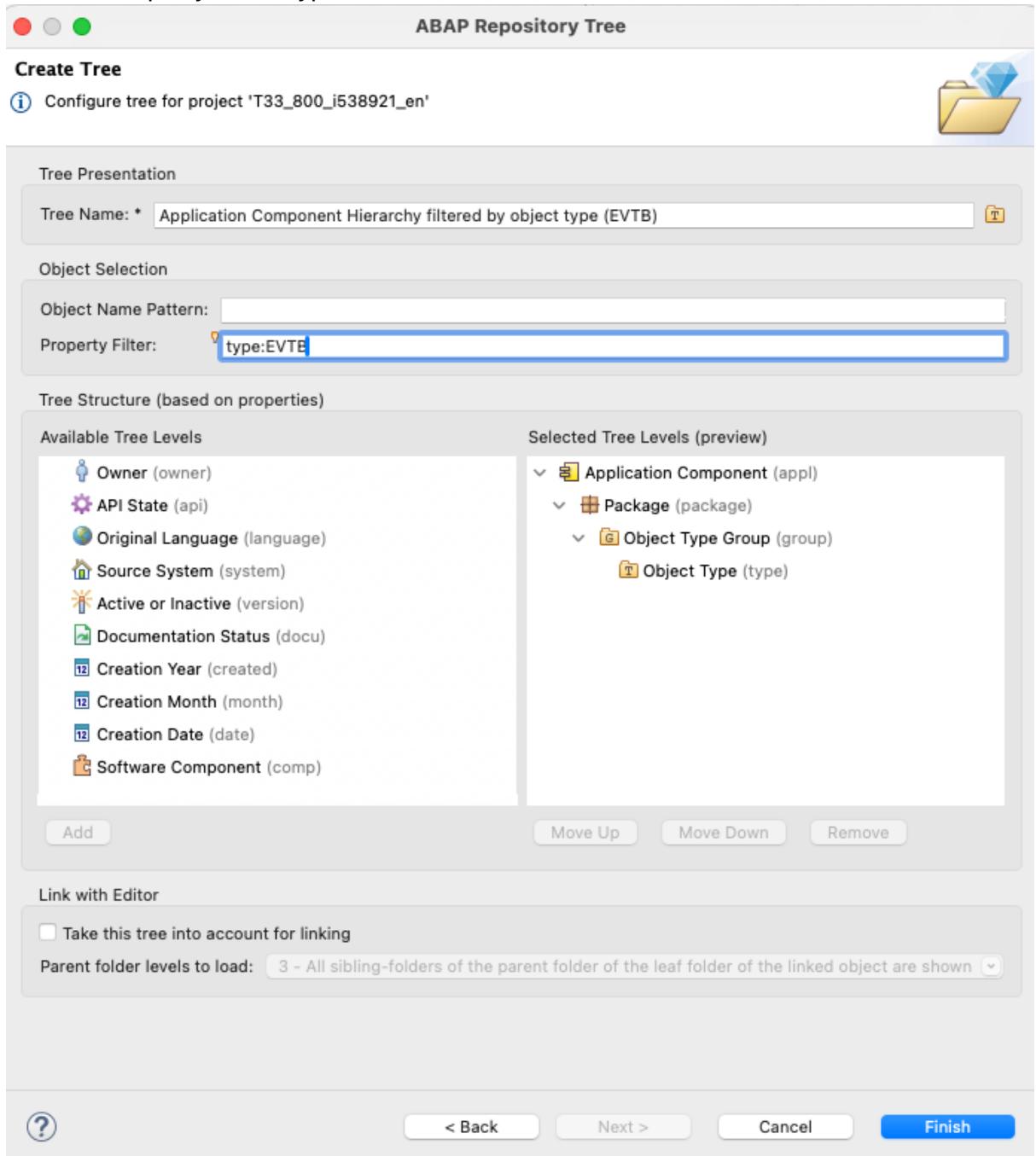


2. Select “Application Component Hierarchy” and hit “Next”



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

- ✓ Application Component Hierarchy filtered by object type (EVTB) (633)
 - > - (5) No application component assigned
 - > AC (1) Accounting - General
 - ✓ AP (2) Application Platform
 - > AP-MD (2) Master Data
 - > AP-MD-BP (2) SAP Business Partner
 - > AP-MD-BP-RAP (2) Business Partner RAP
 - ✓ MDC_BUPA (2) Business Partner
 - ✓ MDC_BUPA_MAIN (2) Business Partner Main
 - ✓ MDC_BUPA_BO (2) Business Partner: Business Objects
 - > Business Services (2)
 - > Event Bindings (2)
 - > S_BUSINESSPARTNER_CHANGE Event Binding for Business Partner change
 - > S_BUSINESSPARTNER_CREATE Event Binding for Business Partner create
 - > BC (3) Basis Components
 - > CA (63) Cross-Application Components
 - > CM (15) Enterprise Contract Management
 - > CRM (59) Customer Relationship Management
 - > FI (31) Financial Accounting
 - > FIN (2) Financials
 - > IS (2)
 - > LE (19) Logistics Execution
 - > LO (67) Logistics - General
 - > MM (97) Materials Management
 - > PLM (26) Product Lifecycle Management
 - > PM (42) Plant Maintenance
 - > PP (19) Production Planning and Control
 - > PPM (6) Portfolio and Project Management
 - > PS (8) Project System
 - > QM (28) Quality Management
 - > SD (108) Sales and Distribution
 - > TM (30) Transportation Management
 - > Released Objects (14.415)

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

Event Binding: S_BUSINESSPARTNER_CHANGE

General Information					
Type Namespace:	sap.s4.beh				
SAP Object Type:	BusinessPartner				
Operation:	Changed				
Type:	sap.s4.beh.businesspartner.v1.BusinessPartner.Changed.v*				
Event Versions					
Major Version	Minor Version	Patch Version	Entity Name	Entity Event Name	
0001	0	0	I_BUSINESSPARTNER_2	CHANGED	Add... Edit... Remove

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

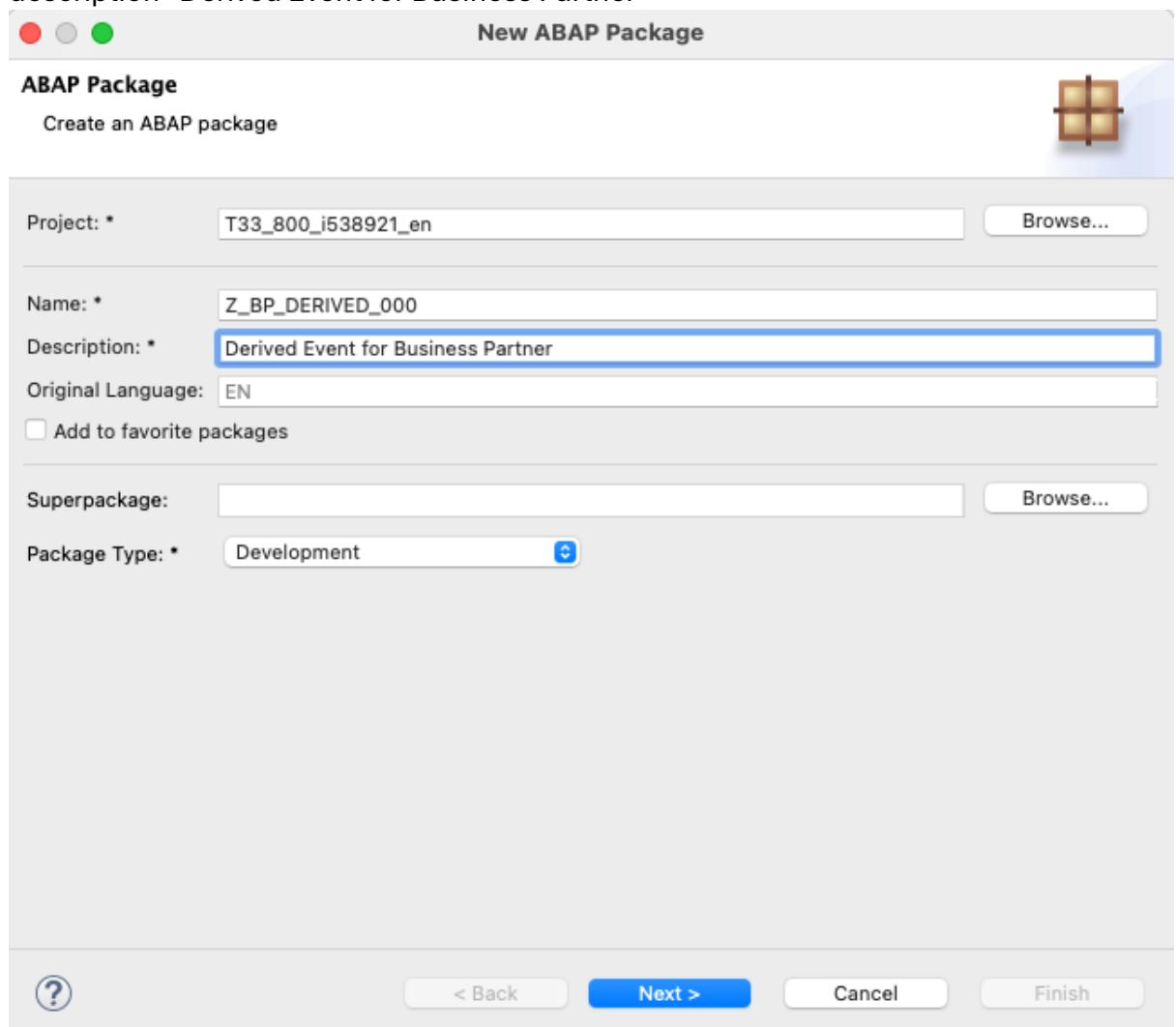
```
B [T33] I_BUSINESSPARTNERTP_2 X
39   'B_BUP_DCPA';
40 }
41 //define authorization context NoCheckWhenReadingorModifying for disable(read,modify,save : late) {
42 @define authorization context NoCheckWhenReadingorModifying for disable(read,modify,save : late) {
43   'B_BUPA_ADR';
44   'B_BUPA_BNK';
45   'F_BNKA_MAN';
46   'F_BNKA_MAO';
47   'B_BUPA_GRP';
48   'B_BUPA_RLT';
49   'B_BUP_PCPT';
50   'B_BUP_DCPD';
51   // 'B_BUPA_RAT';
52   // 'B_BUPA_CRS';
53   'B_BUP_DCPA';
54 }
55
56 //with dvm;
57 foreign entity I_BusPartRelationshipTP alias BusPartRelationship;
58 foreign entity I_PaymentCardMasterTP_2 alias PaymentCardMaster;
59 @define behavior for I_BusinessPartnerTP_2 alias BusinessPartner with unmanaged save
60 lock master unmanaged
61 total etag ETag
62 authorization master ( instance )
63 draft table bupa_root_d_2
64 late numbering
65 extensible
66 {
67   create;
68   update;
69   delete;
70   draft action Edit with additional implementation;
71   draft action Activate;
72   draft action Discard;
73   draft action Resume;
74   draft determine action Prepare extensible;
75   event created;
76   event Changed;
77   // determine action draft_validate {
78   //   validation ( always )validateAll;
79   // }
80
81   association _BusinessPartnerRole { create(precheck); with draft; }
82   association _BusinessPartnerTaxNumber { create; with draft; }
83   association _BusinessPartnerIdentification { create; with draft; }
84   association _BusinessPartnerIndustry { create; with draft; }
85   association _BusinessPartnerBank { create(precheck); with draft; }
86   // association _BusinessPartnerAddress { create; with draft; }
87   association _BusinessPartnerAddress { create(precheck); with draft; }
88   association _BusPartRelationship { create; with draft; }
89   association _BusPartContactPerson { create; with draft; }
90   association _BusinessPartnerPaymentCards { create; with draft; }
91   // association _BPAddressIndependentPhone { create; with draft; }
92   // association _BPAddressIndependentMobile { create; with draft; }
93   // association _BPAddressIndependentEmail { create; with draft; }
94   association _BPAddrIndependentPhone { create; with draft; }
95   association _BPAddrIndependentMobile { create; with draft; }
96   association _BPAddrIndependentEmail { create; with draft; }
97   association _BPAddIndependentWebsite { create; with draft; }
98   association _BPAddIndependentFax { create; with draft; }
99   association _BPDataController { create; with draft; }
100  // field ( readonly, numbering : managed )
101  field ( readonly:update )
102  BusinessPartner.
```

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.

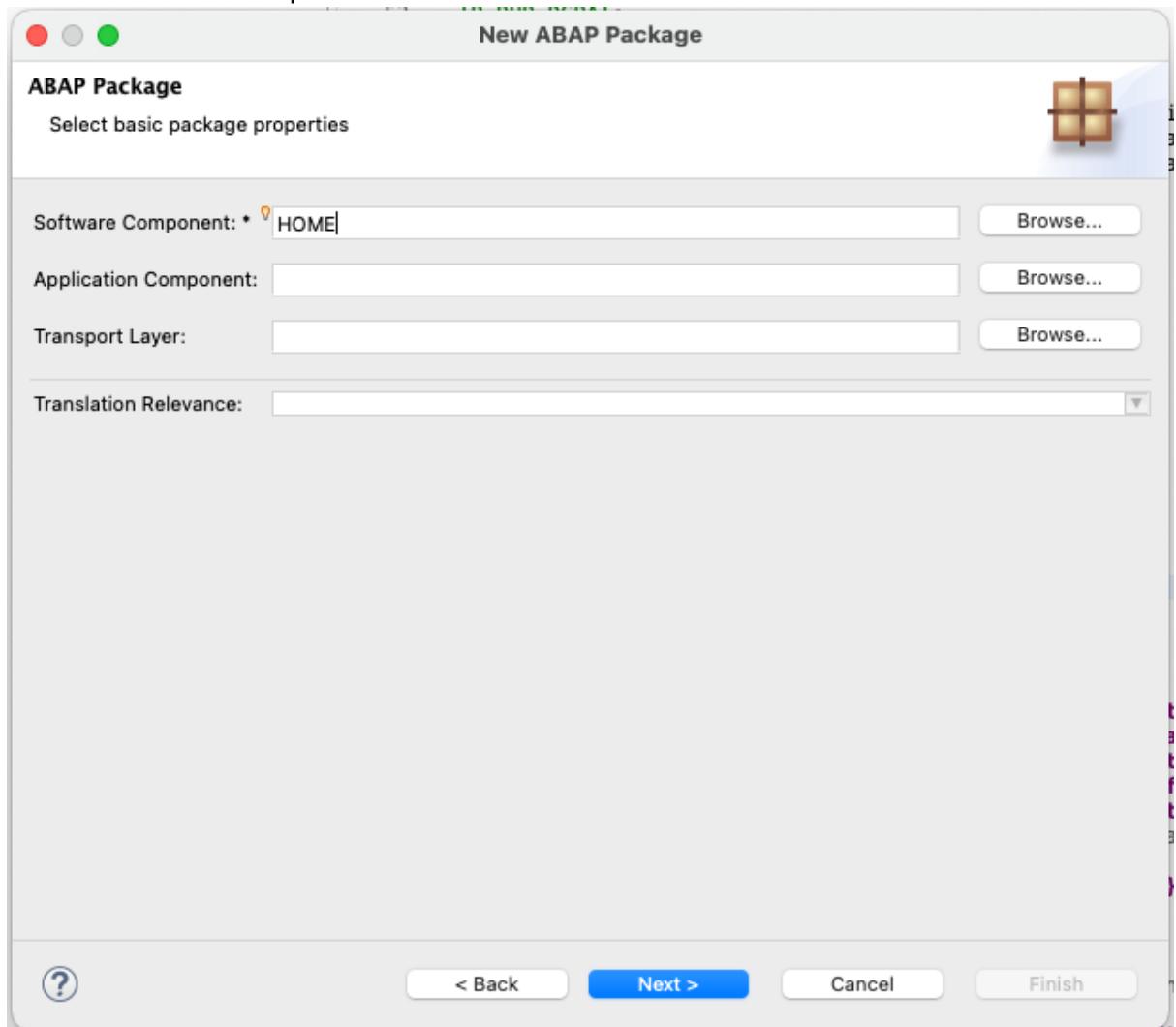
Since derived events use CDS views to define and populate the data of the payload, let's start of by creating such a CDS view.

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_PAYLOADADD_000” and the description “Payload for BP Changed Derived Event”

New Data Definition

Data Definition

Create a data definition

D

Project: * T33_800_i538921_en Browse...

Package: * Z_BP_DERIVED_000 Browse...

Add to favorite packages

Name: * Z_BP_CHANGED_PAYLOAD_000

Description: * Payload for BP Changed Derived Event

Original Language: EN

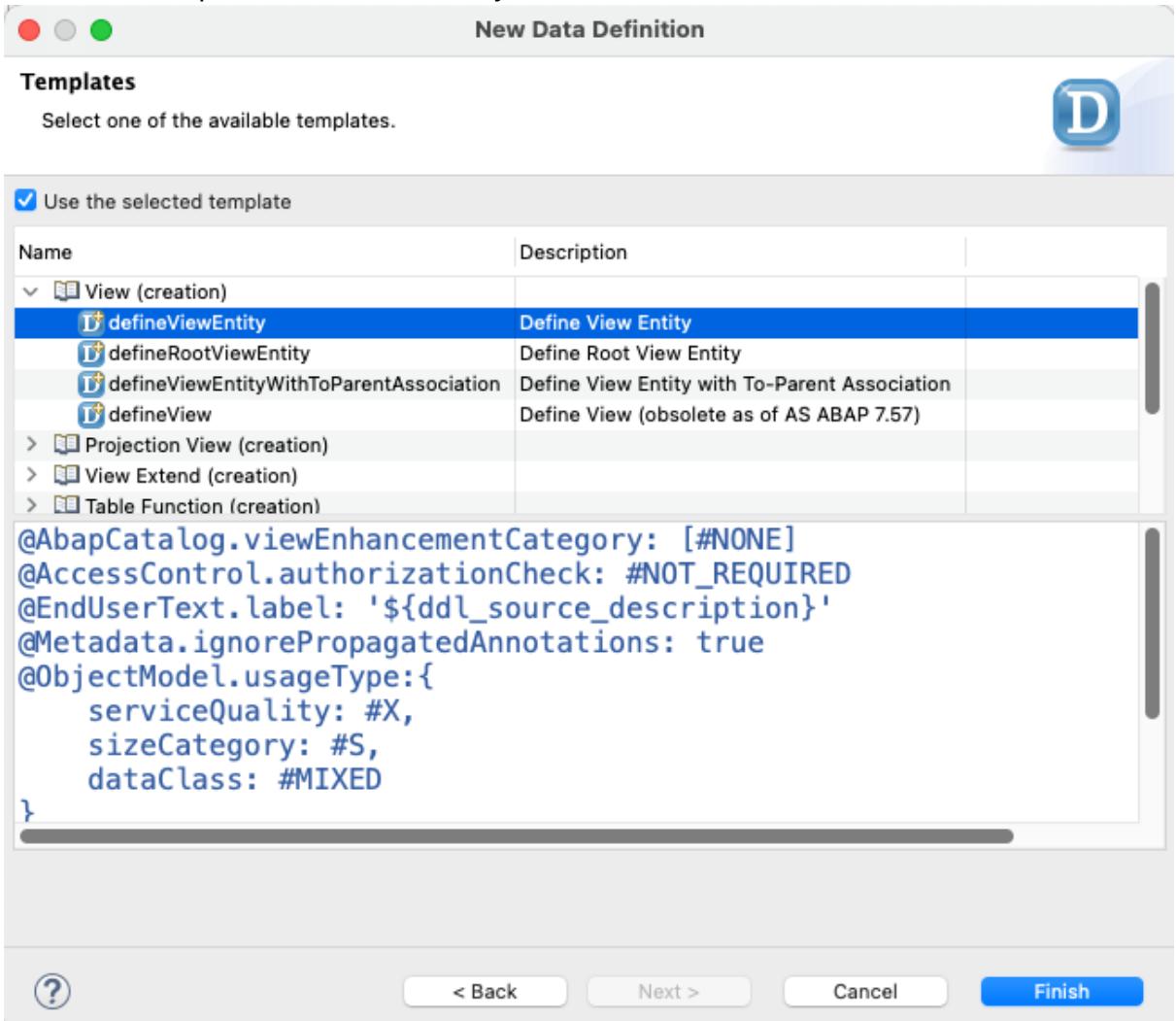
Referenced Object: Browse...

?

< Back Next > Cancel Finish

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:

```

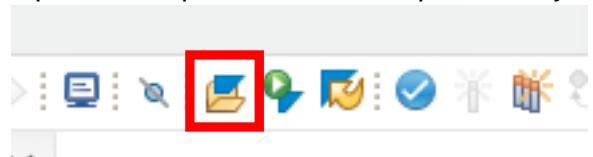
@AbapCatalog.viewEnhancementCategory: [#NONE]
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Payload for BP Changed Derived Event'
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{ 
    serviceQuality: #X,
    sizeCategory: #S,
    dataClass: #MIXED
}
define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

```

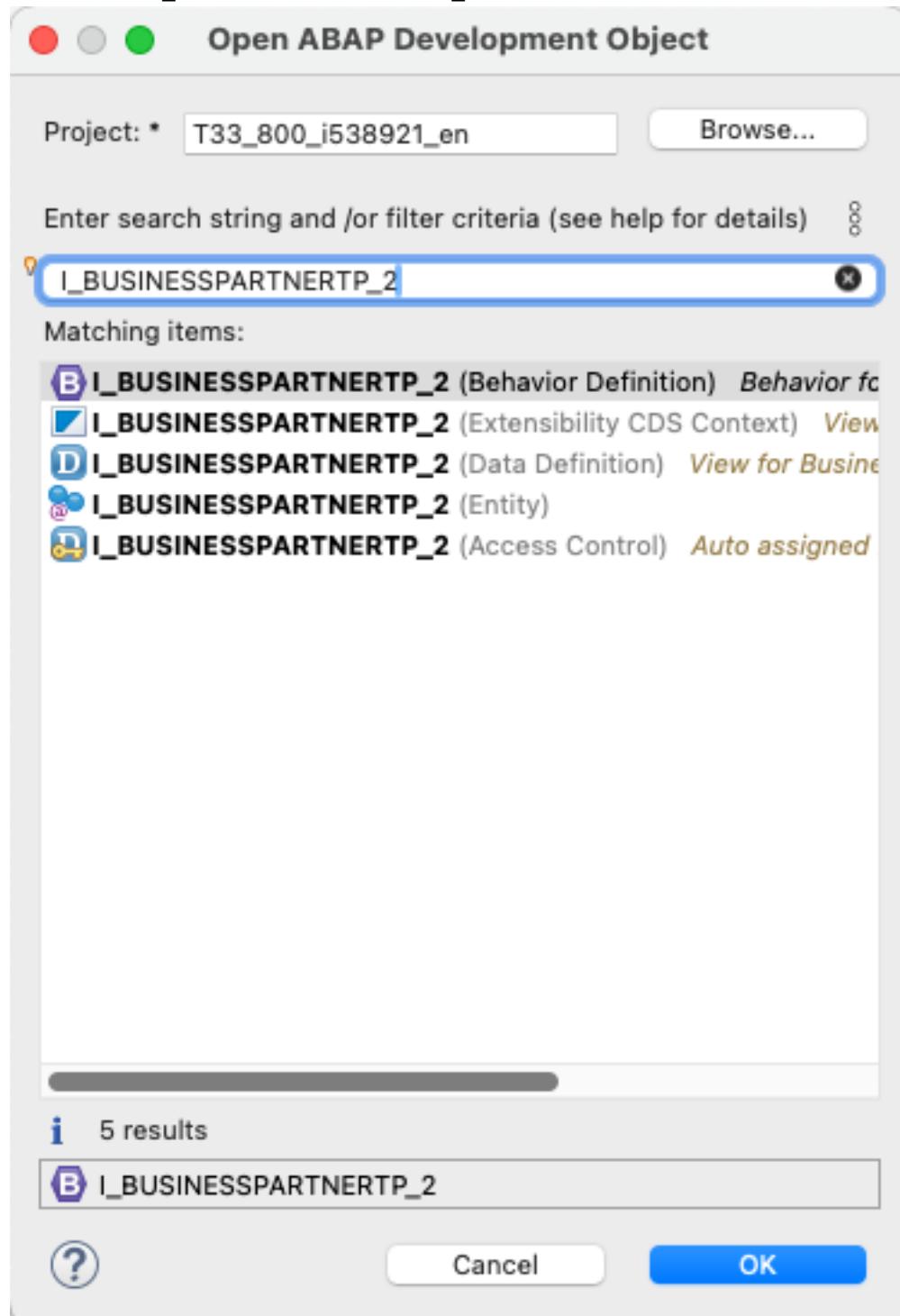
19. Save and activate the CDS view.

Next let's define the derived event itself.

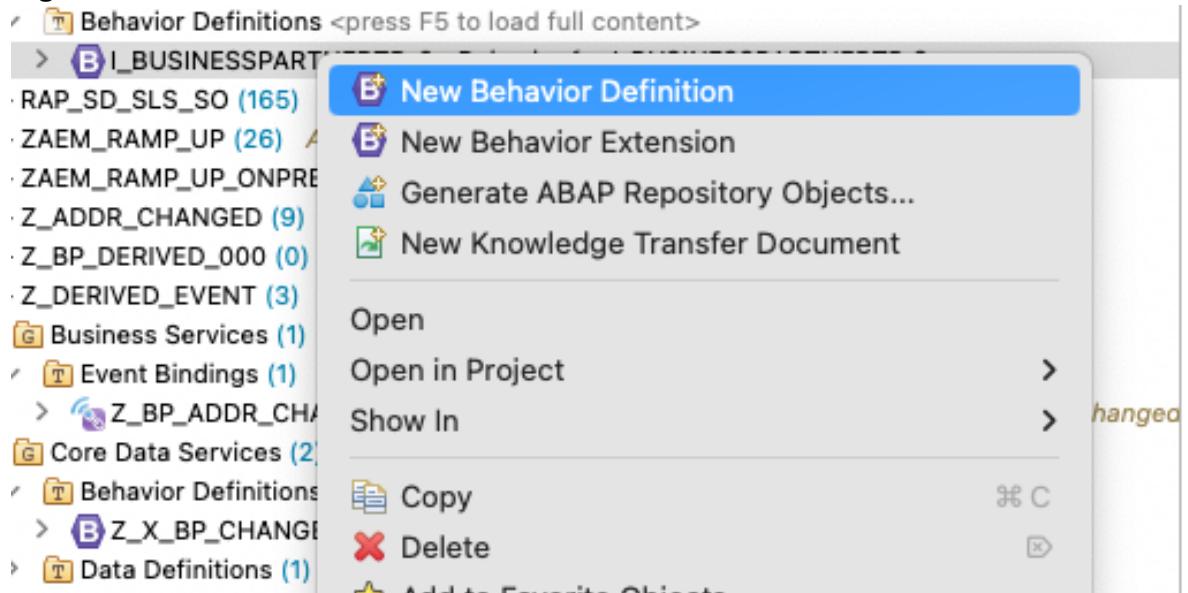
20. Open the “Open ABAP Development Object” dialog



21. Search for “I_BUSINESSPARTNER_TP_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 except the one for “BusinessPartner”

```
*[T33] Z_X_BP_DERIVED_000 X
1 extension implementation in class zbp_x_bp_derived_000 unique;
2
3 extend behavior for BusinessPartner
4 {
5 }
```

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”

New Event Binding

Event Binding

Create Event Binding

Project: *	T33_800_i538921_en	Browse...
Package: *	Z_BP_DERIVED_000	Browse...
<input type="checkbox"/> Add to favorite packages		
Name: *	Z_EB_BP_DERIVED_000	
Description: *	Event Binding for BP Derived Event	
Original Language:	EN	

< Back **Next >** Cancel Finish

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

Event Binding: Z_EB_BP_DERIVED_000

General Information

Type Namespace: * zcustom000

SAP Object Type: * BusinessPartner

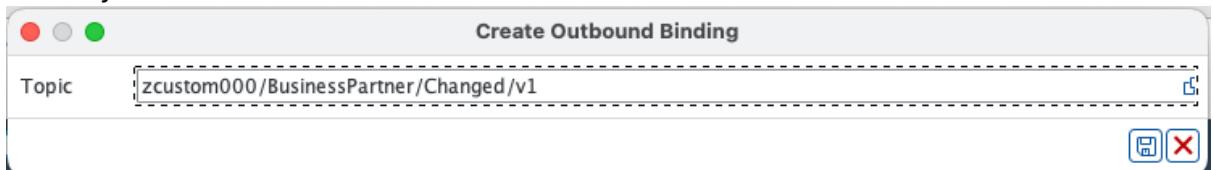
Operation: * Changed

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 “/IWXB/E/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 whenever the standard event is triggered and should be sent 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 “/IWXB/EVENT_MONITOR”
45. Select your channel, select “Outbound Events” and select your event (depending on your S/4HANA version the steps could be different)

The screenshot shows the SAP Enterprise Event Enablement - Event Monitor interface. At the top, there are buttons for 'Delete Events', 'Refine Search Criteria', and 'Refresh'. Below this, a section titled '1 Outbound Events for Channel AEM_BROKER' displays a single event. The event details are as follows:

Topic	S/4HANA/Events/ce/zcustom000/BusinessPartner/Changed/v1
Status	
Event ID	FA163E75AABA1EDFAAFFA1FACF8FBBS
Arrival Time	26.11.2024 14:0...
Publish Time	26.11.2024 14:0...
Acknowledged Time	26.11.2024 14:0...
Dynamic Topic Segments	

Below the table, under the 'Payload' section, is a JSON representation of the event data:

```

"time": "2024-11-26T14:07:48Z",
"datacontenttype": "application/json",
"data":
{
    "BusinessPartner": "101",
    "BusinessPartnerCategory": "2",
    "BusinessPartnerFullName": "Test3 Test2"
}

```

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.

The screenshot shows the SAP Fiori interface for the Business Partner (Gen.) screen. At the top, there are two input fields: 'Business Partner' with value '101' and 'Change in BP role' with value 'Business Partner (Gen.)'. Below this is a navigation bar with tabs: 'Address' (which is selected), 'Address Overview', 'Identification', 'Control', 'Payment Transactions', 'Status', and 'A'. Under the 'Address' tab, there is a 'Name' section. It contains a 'Title' field with a dropdown arrow and a 'Name' field with a dropdown arrow. The dropdown for 'Name' shows two entries: 'Test3' and 'Test'. The 'Test' entry is currently selected, indicated by a dashed border around its row.

6. Go into transaction “/IWXB/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 “/IWXB/EVENT_FILTER”

The screenshot shows the SAP Event Filter Configuration interface. On the left, there is a tree view of 'Channels with Bindings' under 'Active Channels'. One node is expanded to show 'Outbound Bindings'. On the right, there is a table titled 'Filter Conditions: zcustom000/BusinessPartner/Changed/v1'. The table has columns for 'Property Name', 'Option', 'Low Value', and 'High Value'. There are several rows in the table.

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”

The screenshot shows the SAP Event Filter Configuration interface. The 'Outbound Bindings' section is expanded, and the 'zcustom000/BusinessPartner/Changed/v1' binding is selected. On the right, the 'Filter Conditions' table shows the context attribute 'xsapbphousenumber' highlighted in blue.

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

The screenshot shows the 'Create Event Filter' dialog box. It has three tabs at the top: a red tab, a grey tab, and a green tab (selected). The main area has three columns: 'Property', 'Option', and 'Value'. The 'Property' column contains 'xsapbphousenumber', the 'Option' column contains 'equals', and the 'Value' column contains '16'. At the bottom right are 'Add Filter' and 'Cancel' buttons.

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

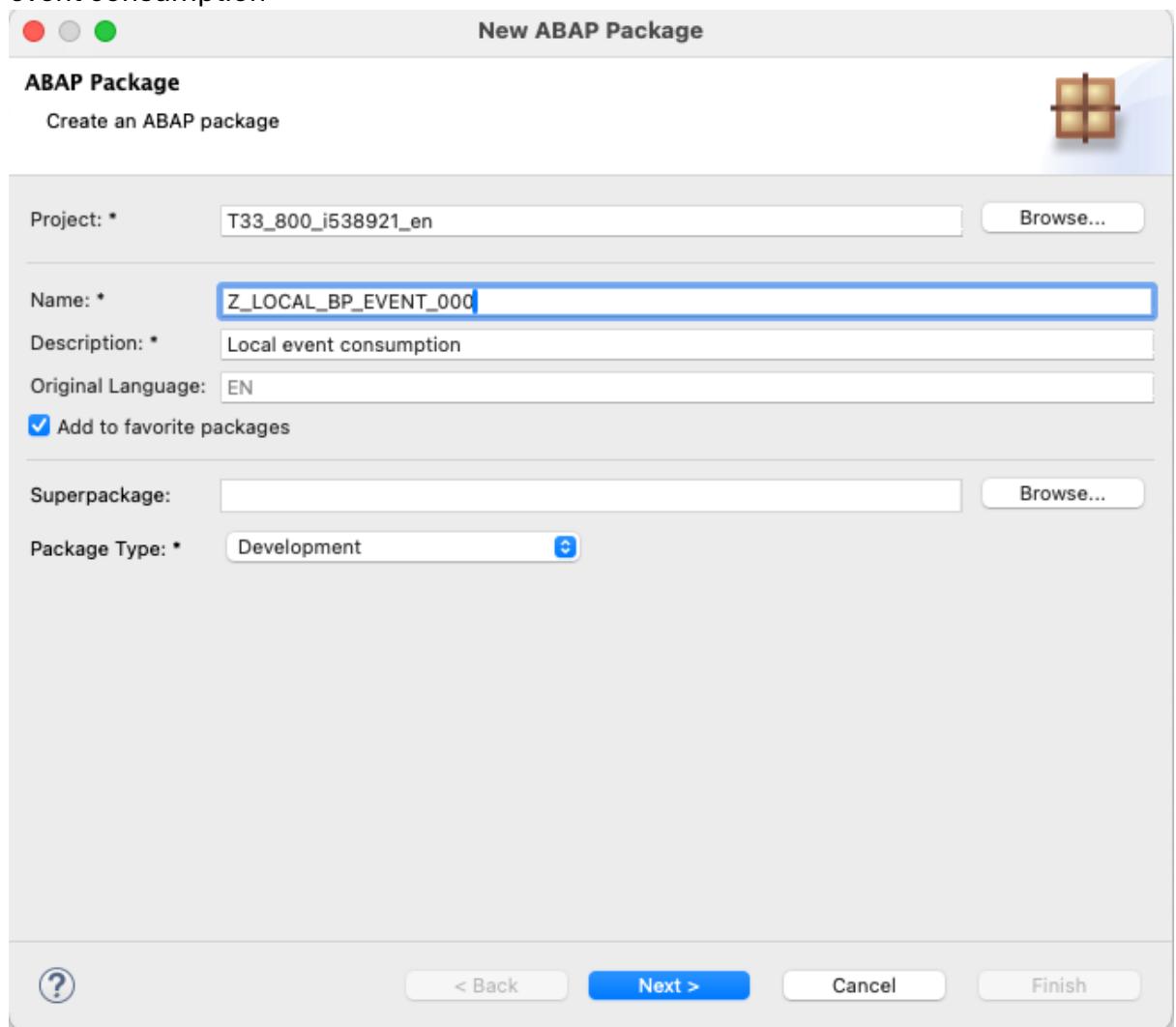
9. Go into transaction “BP”, select a business partner, change the house number to 16 and save.

10. Go into transaction “/IWXB/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.

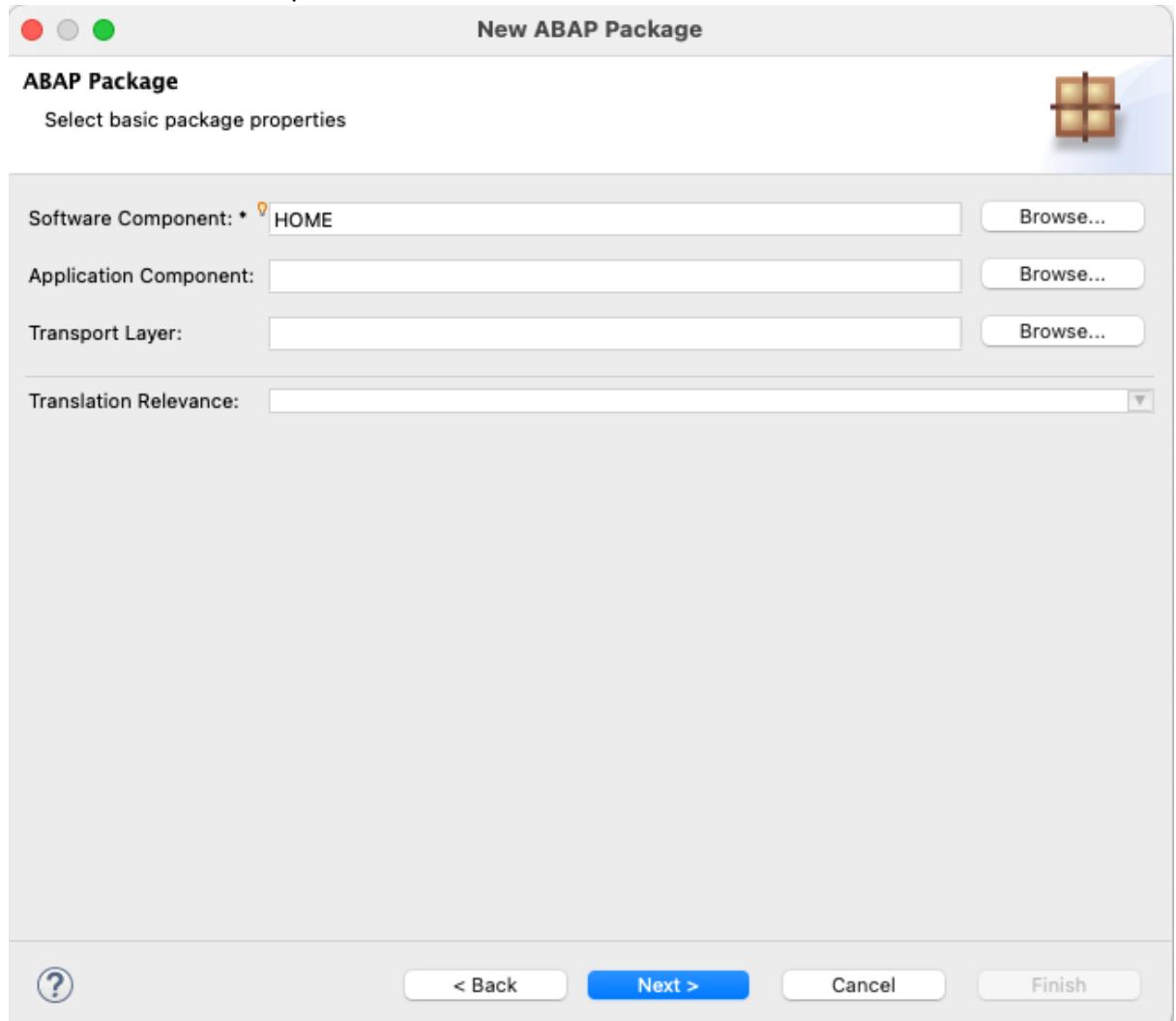
Creating local event handlers

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”



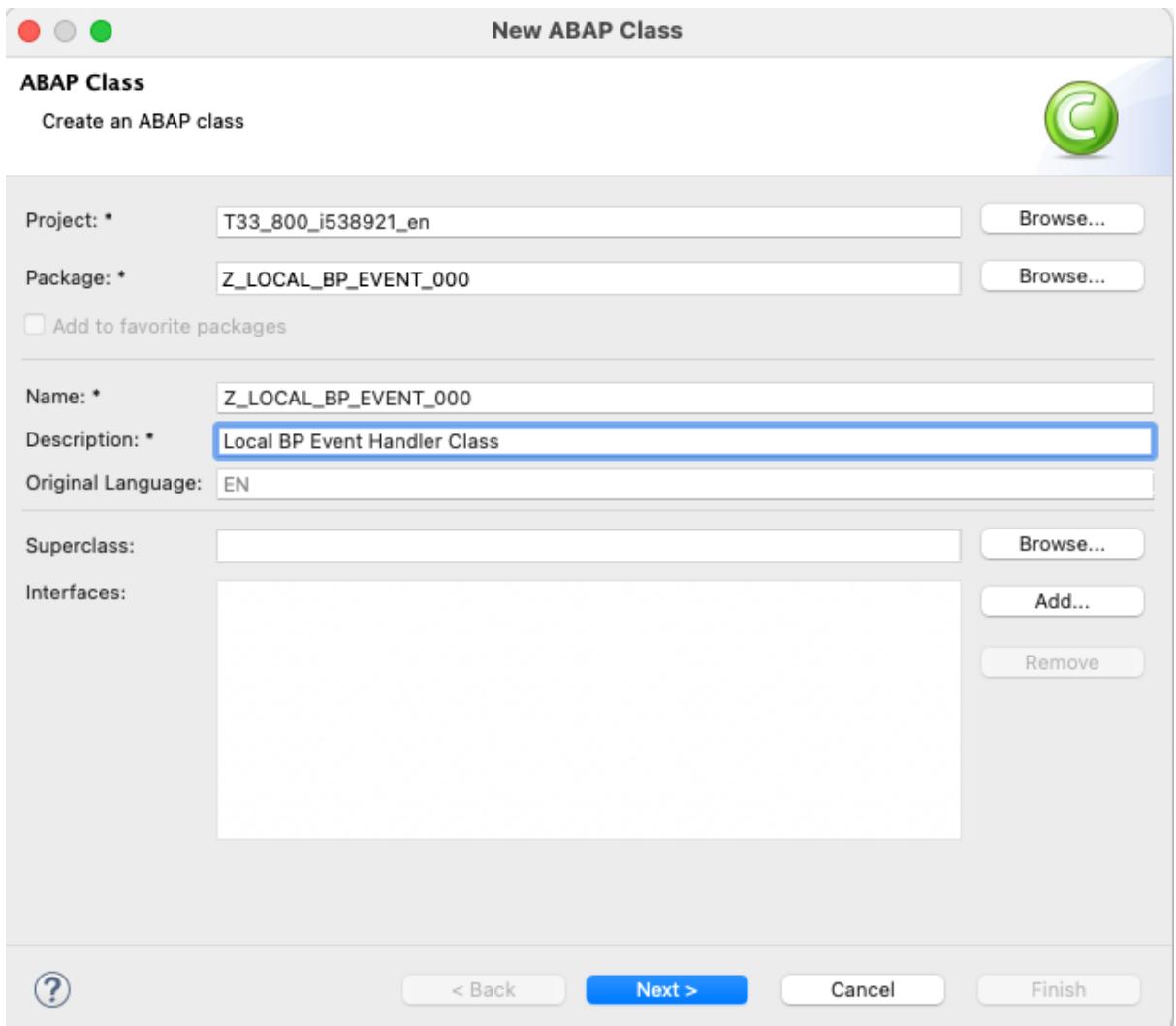
2. Select Software Component “HOME”



3. Select a transport request and click “Finish”

4. Right click your package and select “New”>”ABAP Class”

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



- Click “Next”, select a transport request and hit “Finish”
- 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.

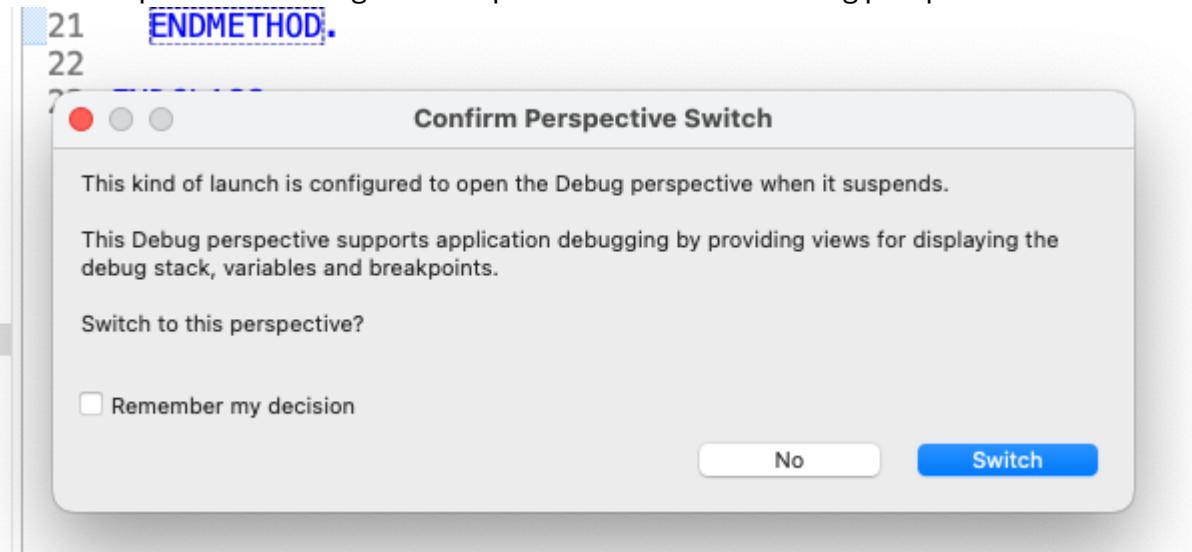
10. Set a breakpoint by double clicking the line number of the “cl_demo_output=>write(‘Event consumed’)” statement

```

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 ENDOBJ.
18
19 ENDMETHOD.
20
21 ENDCCLASS.

```

- 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”

```

1 CLASS lcl_abap_behv_event_handler DEFINITION INHERITING FROM cl_abap_behv_event_handler.
2   CONSUME_CHANGED loop
3   PRIVATE SECTION.
4   METHODS consume_changed FOR ENTITY EVENT changed_instances FOR Business
5     ENDCLASS.
6
7   METHODS consume_changed.
8
9   CLASS lcl_abap_behv_event_handler IMPLEMENTATION.
10
11   METHOD consume_changed.
12     cl_abap_tx->save( ).  

13
14     LOOP AT changed_instances INTO DATA(changed_instance).  

15       cl_demo_output->write( 'Event consumed' ).  

16
17
18   ENDMETHOD.  

19
20   ENDLOOP.  

21
22 ENDCLASS.
23

```

Name	Value	Actual T: Technicki
<Enter variable>		
MY-SUBRC	0	SYST_BUI
ME	(029*CLASS-POOL=Z_LOC (Local) C Ref to LCL	
Locals		

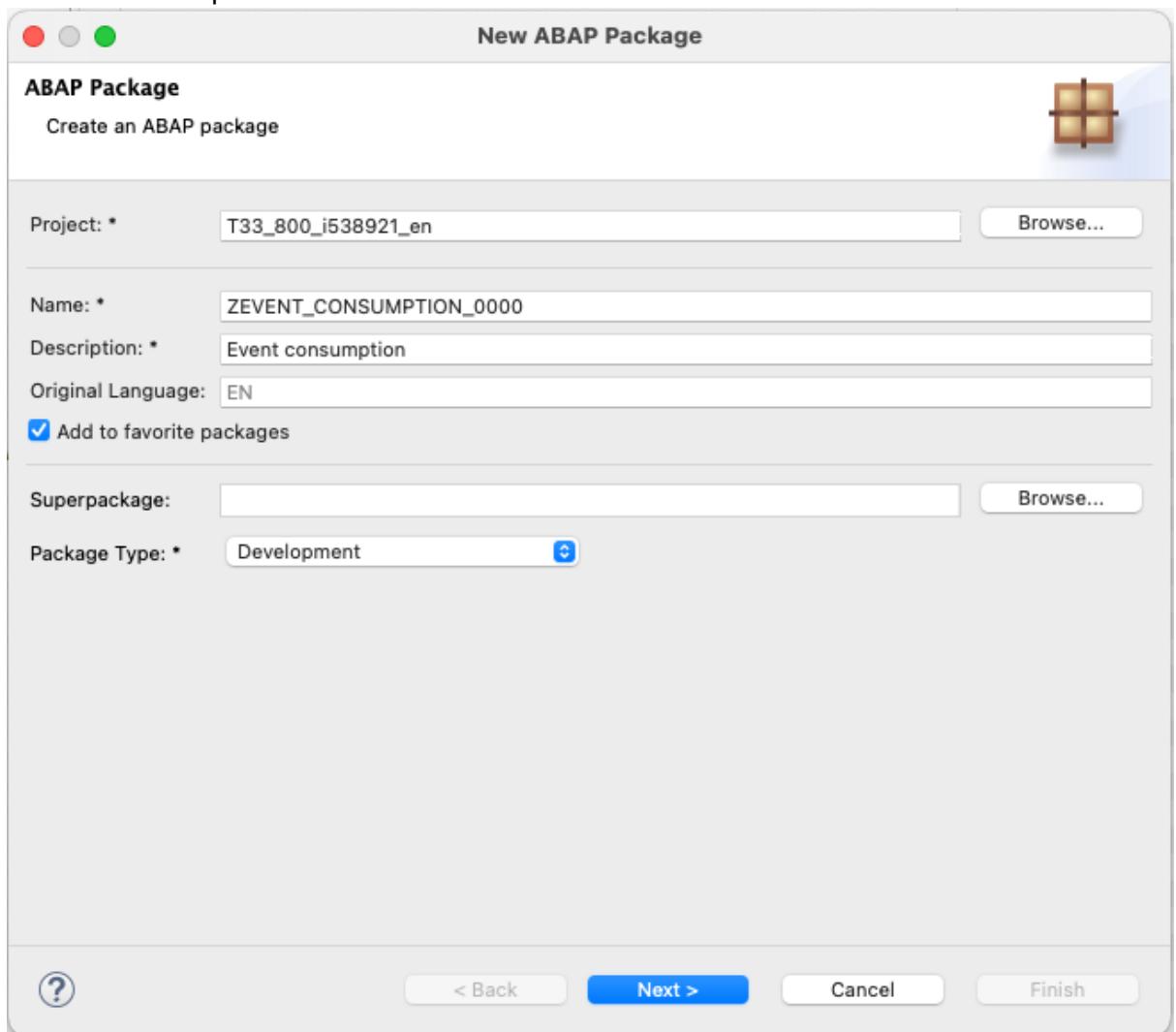
Receive events with S/4HANA

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.

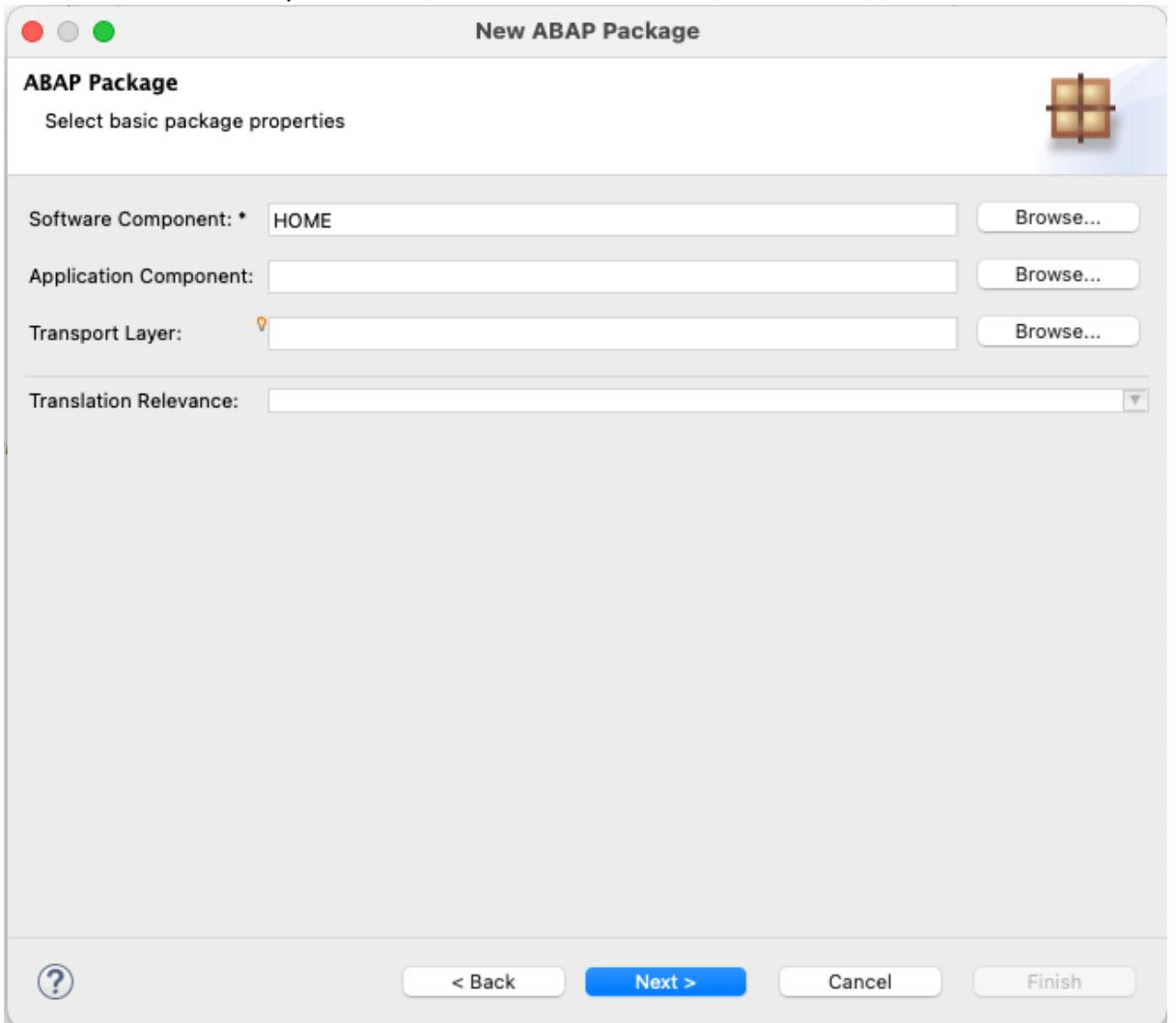
1. Go to the event specification of [Business Partner Events](#) within the Business Accelerator Hub and download the JSON file.

The screenshot shows the SAP Business Accelerator Hub interface. At the top, there's a navigation bar with links for Explore, Resources, Discover Integrations, and Partner with Us. Below the navigation is a blue header bar with the title "Business Partner Events" and a subtitle "Informs a remote system about created and changed business partners in an SAP S/4HANA system". The main content area has tabs for Overview, Event References, Event Consumption, and Documents. The Overview tab is selected. On the left, there's an "Introduction" section with a brief description of what a business partner is and a list of available events: "Business partner changed" and "Business partner created". To the right, there's a summary card with details like Status (ACTIVE), Version (1.0.0), Last Modified (28 Feb 2024), and Type (EVENT). Below the introduction, there's a "Event Resources" section with a "Event Specifications" heading. Under this heading, there's a table with two rows: "JSON" and "YAML". The "JSON" row is highlighted with a red box. At the bottom right of the resources section is a "Chat (Beta)" button. A "FEEDBACK" button is located on the far right edge of the page.

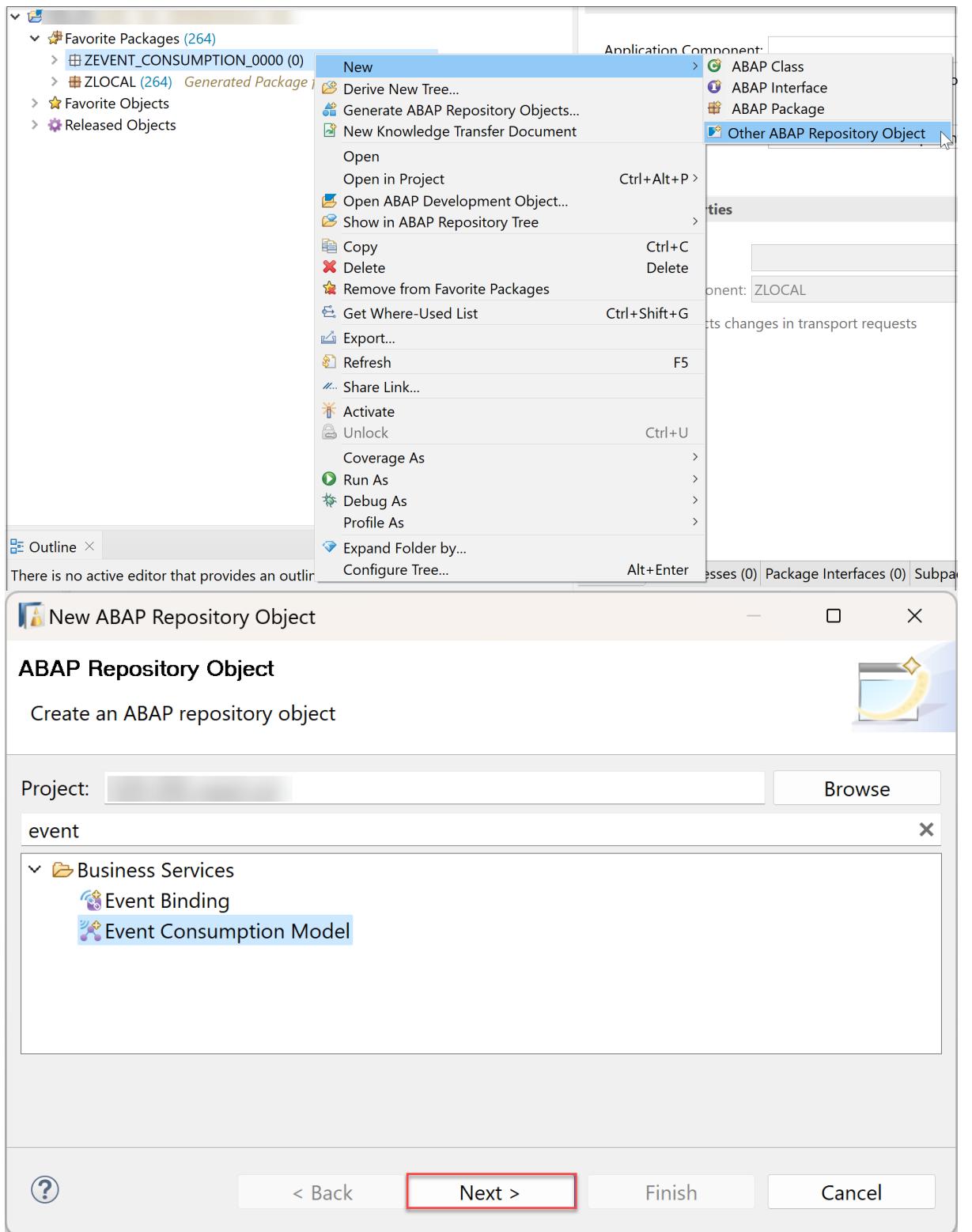
2. Create a new ABAP Package “ZEVENT_CONSUMPTION_0000” with description “Event consumption”



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.



- 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"

New Event Consumption Model

Event Consumption Model

Create Event Consumption Model

Project: * T33_800_i538921_en

Package: * Z_EVENT_CONSUMPTION

Add to favorite packages

Name: ZBPEVENTS0000 0001

Description: * Event Consumption Model

Original Language: EN

Namespace / Prefix / Identifier: *

<No namespace in package> / /

Event Specification File: *

/Users/i538921/Downloads/OP_BUSINESSPARTNEREVENTS (1).json

Namespace and Prefix for objects to be generated:

Z

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”

The screenshot shows the SAP Inbound Binding Configuration interface. On the left, there is a tree view under 'Channels' with sections for 'Active Channels' and 'Advanced Service Plan'. Under 'Advanced Service Plan', the 'AEM_BROKER' node is selected and highlighted with a red box. On the right, there is a table titled 'Inbound Bindings of Channel AEM_BROKER' with columns for 'Status', 'Maintained by', and 'Topic'. The 'Topic' column contains entries like 'COE_EMEA_TECH', 'COMMUNITY_EM_1585208', etc. Below the table, there is a section titled 'Consumers' with a table showing 'Status', 'Maintained by', 'Repository', 'Consumer', 'Version', and 'Destination'.

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

The screenshot shows the 'Create Inbound Binding' dialog. At the top, there are three status indicators (red, grey, green). Below them is a title bar with 'Create Inbound Binding'. The main area has a 'Topic' field containing the value 'sap/s4/beh/businesspartner/v1/BusinessPartner/Changed/v1'. To the right of the field are two buttons: a blue arrow pointing right and a red X.

13. Switch to the next page

14. Choose your consumer, “ZBPEVENTS0000” and click “Create Destination”

The screenshot shows the 'Create Inbound Binding' dialog. At the top, there are three status indicators (grey, grey, grey). Below them is a title bar with 'Create Inbound Binding'. The main area has a 'Topic' field containing the value 'sap/s4/beh/businesspartner/v1/BusinessPartner/Changed/v1'. Below the topic field is a section titled 'Consumers to be generated' with a table. The table has columns for 'Val...', 'Consumer', 'Versi...', and 'Destination'. It contains two rows: one for 'ZBPEVENTS0000' and another for 'ZEVENTSBUSINESSPARTNER'. A red box highlights the 'Create Destination' button at the top of the table. To the right of the table are three buttons: a blue arrow pointing left, a blue arrow pointing right, and a red X.

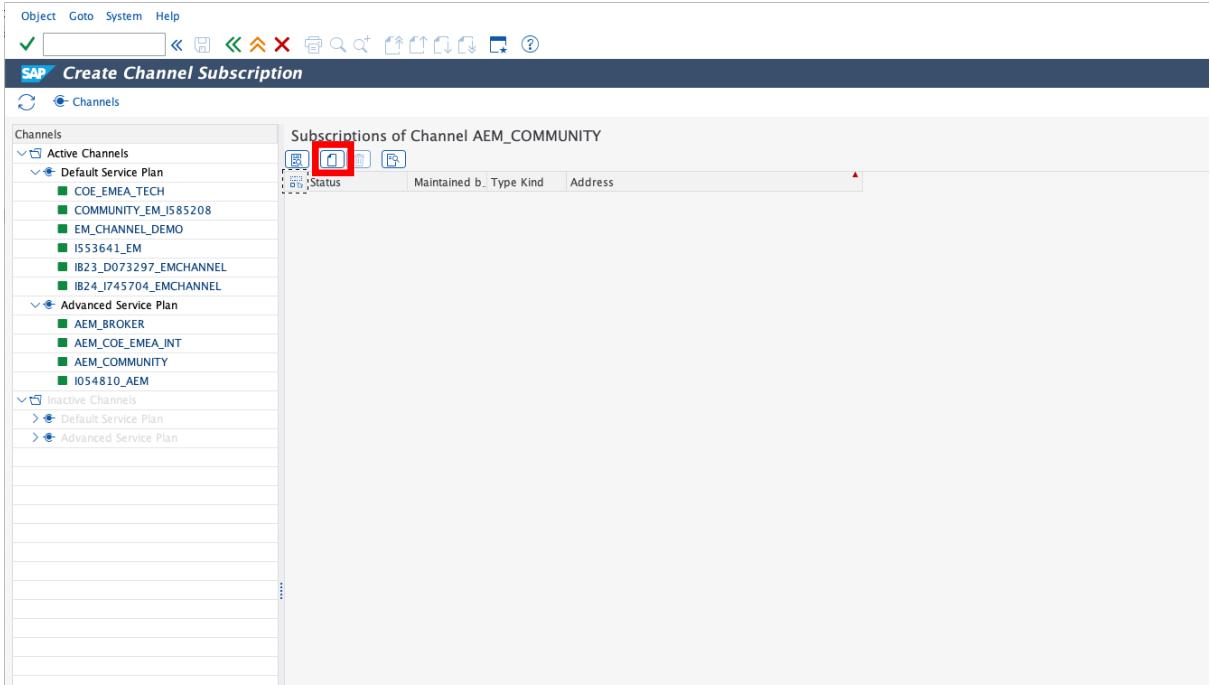
15. Enter your user name in the Create Destination dialog and press save

16. Press save on the Create Inbound Binding dialog

With the Inbound Binding configured we can configure the subscription next

17. Open transaction “/IWXB/E/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”.

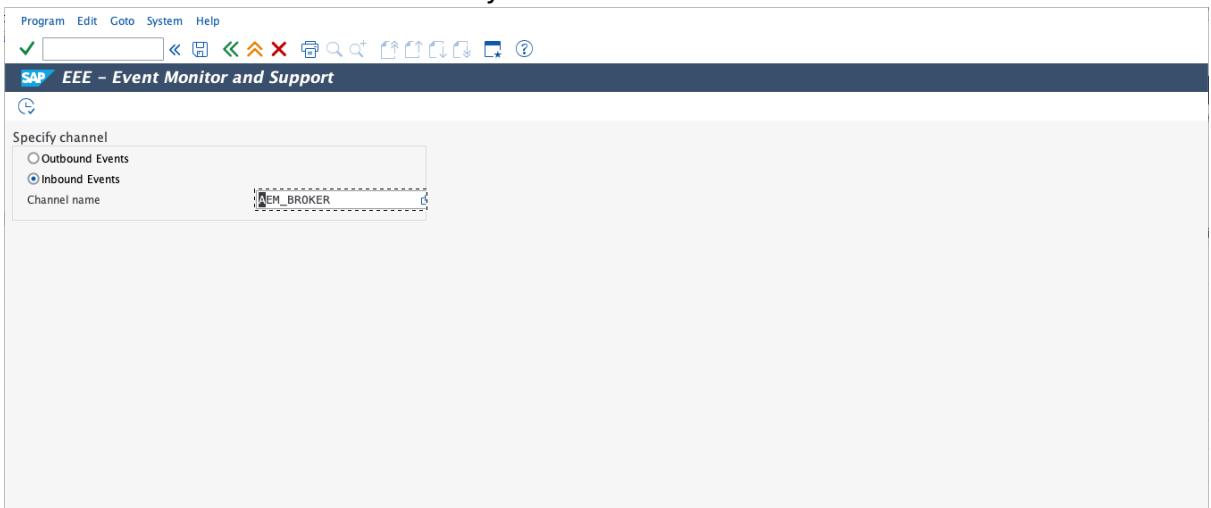


20. Save the subscription

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

1 Inbound Events of Channel AEM_BROKER -- Received										
		Event ID	Topic	Arrival Time	Payload Size [k...]	Status	Retries	Repository ID	Consumer ID	Vers...
		FA163E75AA8A1EDF9E9954D67F45B...	sap/s4/beh/businesspartner/v1/Busine...	22.09.2024 10:08...	0,27	In Process	1	DEFAULT	ZBPEVENTS0000	1 At Least Once