



PUBLIC
2020-10-06

SAP (On-Premise) - ABAP Programming Model for SAP Fiori

Content

1	About ABAP Programming Model for SAP Fiori	6
2	Before You Start.....	9
2.1	Prerequisites.	9
3	Get Started.	11
3.1	Developing a Simple List Reporting App	11
	Define a Data Model Based on CDS Views.	13
	Generating OData Service With Auto-Exposure.	19
	Consume Business Data Using SAP Fiori Elements.	27
4	Concepts.	35
4.1	Draft Concept.	35
4.2	Draft Consistency.	40
4.3	Locking and Resume	43
4.4	Messages.	50
4.5	Validations for CDS-Based Business Objects.	58
4.6	Actions for CDS-Based BOPF Business Objects.	60
4.7	Determinations for CDS-Based Business Objects.	63
5	Develop.	67
5.1	Developing List Reporting Apps with Search and Analytical Capabilities.	67
	Data Model Without Metadata - Starting Point.	69
	Adding Metadata to Data Model.	70
	Running Resulting App	75
5.2	Developing New Transactional Apps	78
	Defining the Business Object	79
	Implementing a Service-Specific Consumption View	89
	Running the Resulting SAP Fiori App.	92
	Extending Apps with Quick Actions	96
5.3	Developing New Transactional Apps with Draft Capabilities	105
	Defining the Draft Business Object for Sales Order.	109
	Providing Additional Business Logic with a BOPF Determination.	121
	Defining the Consumption Layer.	124
	Adding UI Semantics for Consumption in Fiori UI.	127
	Running the Resulting Fiori App.	131
	Extending the Implementation for Adding New Sales Order Items.	132
	Extending the Implementation with Value Validation	148

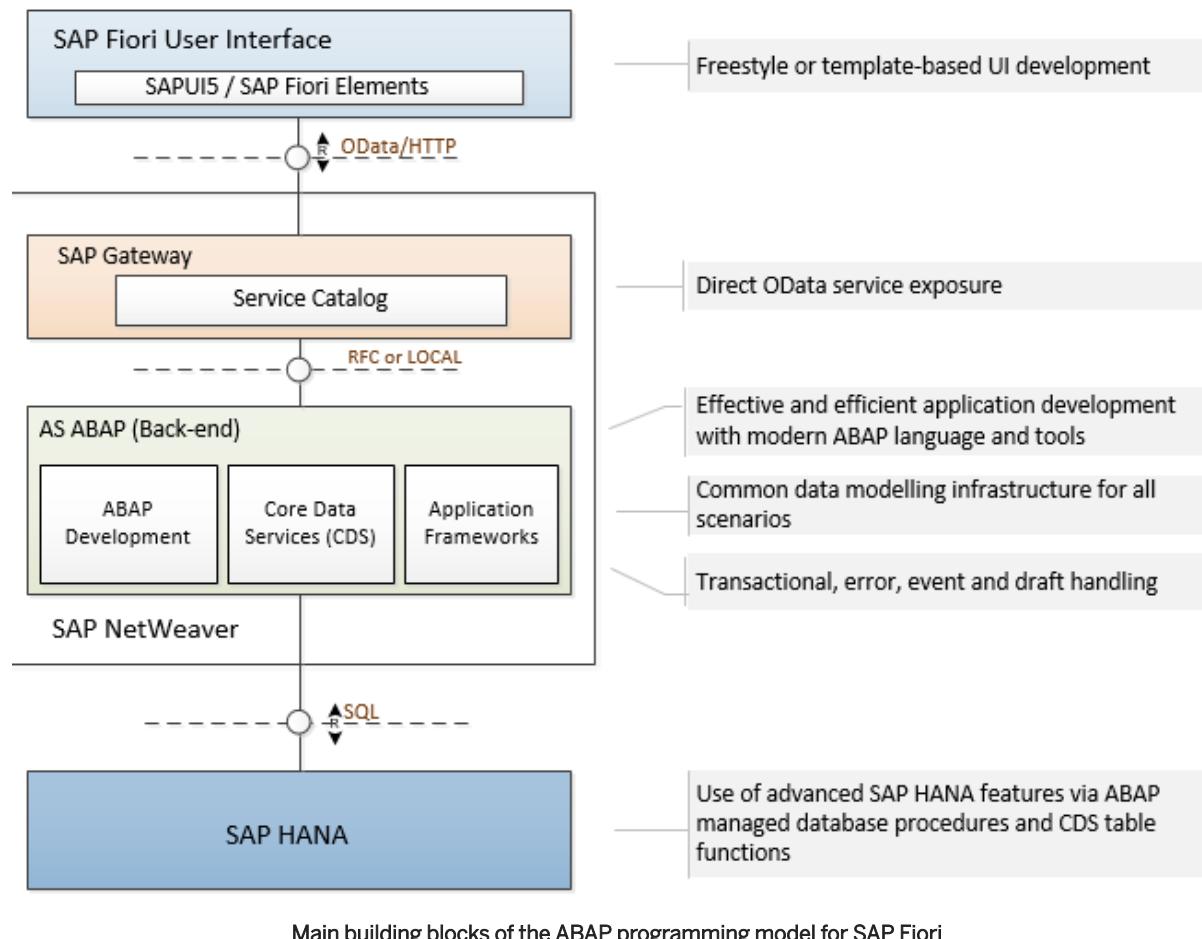
Extending the Implementation with an Action.	152
6 Add Authorizations.	153
6.1 Authorization Checks in Transactional Development Scenarios.	154
6.2 Implementing Authorizations for Read-Only Applications.	156
6.3 Implementing Authorizations for Generated Business Objects.	160
Authorization Concept in BOPF	165
Assigning Authorization Defaults to OData Services.	167
6.4 Defining Authorizations for External Service Consumption.	168
Assigning Authorizations to Roles.	169
7 Extend.	172
7.1 Extending Apps with Custom Code.	175
Creating an Appropriate CDS View Extension.	178
Adding Custom Fields to Extension View	181
Adding Fields from Association	182
Running the Resulting Fiori App.	184
7.2 Metadata Extensions.	185
8 Common Tasks.	188
8.1 Adding Field and Action Control.	189
Static Field Control.	190
Dynamic Field and Entity Control.	194
Dynamic Action Control.	201
8.2 Using Virtual Elements in CDS.	206
Adding Annotations for Virtual Elements.	208
Implementing ABAP Code Exits for Virtual Elements	212
Creating Application-Specific Exception Classes	231
Understanding the ABAP Code Exit API for Virtual Elements.	232
8.3 Consuming Temporal Data.	236
Temporal Data	237
8.4 Deriving Values from Foreign Entities.	247
Annotations for Derivations.	248
8.5 Enabling Text and Fuzzy Searches in SAP Fiori Apps	259
8.6 Using Aggregate Data in SAP Fiori Apps.	263
Annotating Aggregate Functions in CDS.	264
OData Interpretation of Aggregation Annotations.	267
8.7 Defining Text Elements	271
Language-independent Text Elements	272
Getting Text Through Text Associations.	273
8.8 Providing Value Help.	275
Simple Value Help.	277

Value Help with Additional Binding	282
8.9 Adding Field Labels and Descriptions	284
8.10 Mapping CDS View-Elements onto Table Fields	286
8.11 Implementing Draft-Enabled Associations	288
Draft-Enabling for Associations with Cardinality To-One	291
Draft-Enabling for Reverse Associations	294
8.12 Defining CDS Annotations for Metadata-Driven UIs	296
Tables and Lists	297
Detail Pages	303
Field Groups	306
Annotations Similar to dataField	307
Charts	311
Data Points	314
Contact Data	324
Navigation	325
Actions	331
Field Manipulation	333
8.13 Exposing CDS Entities as OData Service	339
Generating OData Service With Auto-Exposure	341
Creating an OData Service based on a Referenced Data Source (RDS)	342
Generating an OData Service Using the Mapping Editor	350
Exposing CDS Entities with Parameters	357
9 Reference	361
9.1 CDS Annotations	361
AccessControl Annotations	362
Analytics Annotations	367
AnalyticsDetails Annotations	379
Consumption Annotations	390
Aggregation Annotations	414
EnterpriseSearch Annotations	420
Hierarchy Annotations	422
ObjectModel Annotations	428
OData Annotations	441
Search Annotations	442
Semantics Annotations	445
UI Annotations	465
VDM Annotations	587
9.2 Understanding the BOPF Authorization API	592
Method CHECK_STATIC_AUTHORITY	594
Method CHECK_INSTANCE_AUTHORITY	596
Context Information for Implementing Authorization Checks	599

10	What's New in ABAP Programming Model	601
10.1	ABAP Platform 1809 FPS1 (7.53, SP01)601
10.2	ABAP Platform 1809 (7.53, SP00)	602
10.3	7.52, SP02	602
10.4	7.52, SP01	603
10.5	7.52, SP00	604
10.6	7.51, SP03	604
10.7	7.51, SP02	605
11	Glossary	606

1 About ABAP Programming Model for SAP Fiori

The *ABAP Programming Model for SAP Fiori* defines the architecture for efficient end-to-end development of intrinsically SAP HANA-optimized Fiori apps in SAP S/4HANA. It supports the development of all types of Fiori applications like transactional, search, analytics and planning apps and is based on customer-proven technologies and frameworks such as Core Data Services (CDS) for defining semantically rich data models, OData protocol, ABAP-based application services for custom logic and SAPUI5-based user interfaces – as shown in the figure below.



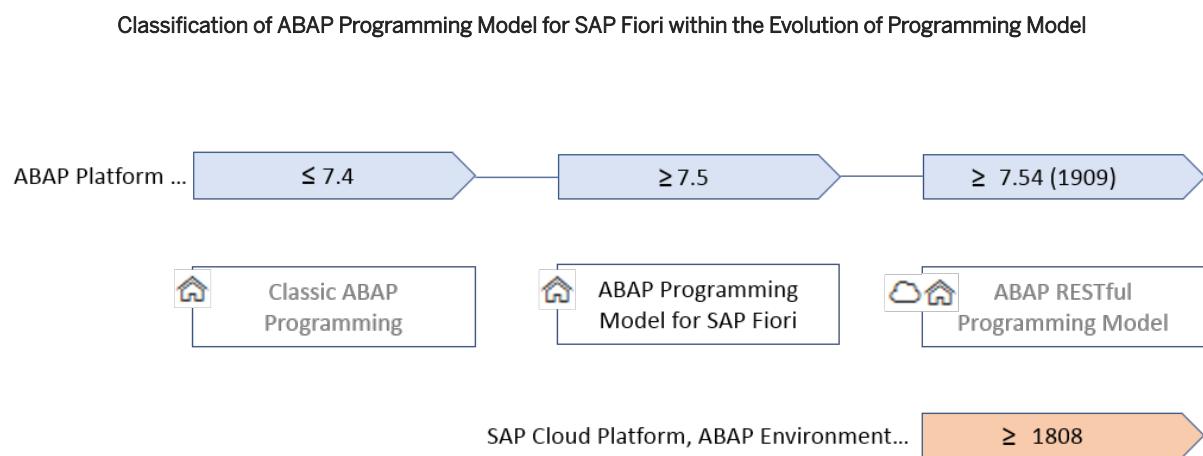
Target Audience

ABAP developers who want to provide OData services within the scope of ABAP programming model for SAP Fiori.

Validity of Documentation

This documentation refers to the range of functions that have been shipped as part of delivery for **AS ABAP 7.54, SP00**.

Evolution of the Programming Model



-
-

The *ABAP Programming Model for SAP Fiori* is based on modern and proven technologies such as CDS, BOPF and SAP Gateway. A lightweight CDS-based flavor of the BOPF framework is used in this context for the transactional processing. This programming model is generally available to customers and partners within **ABAP Platform** starting with release 7.50 SP00.

i For more information about the evolution of the ABAP programming model, read this [blog](#) on the community portal.

Contents

[Before You Start... \[page 9\]](#)

[Get Started \[page 11\]](#)

[Concepts \[page 35\]](#)

[Develop \[page 67\]](#)

[Add Authorizations \[page 153\]](#)

[Extend \[page 172\]](#)

[Common Tasks \[page 188\]](#)

[CDS Annotations \[page 361\]](#) (Reference)

[What's New in ABAP Programming Model \[page 601\]](#)

[Glossary \[page 606\]](#)

2 Before You Start...

... check the Technical Requirements and [Prerequisites \[page 9\]](#)

2.1 Prerequisites

SAP NetWeaver Releases

- SAP NetWeaver AS for ABAP 7.51 innovation package, SP00, or higher
- No specific database is required, but SAP HANA DB is recommended.

Development Environment

- We recommend to use the latest version of ABAP Development Tools for SAP NetWeaver (in short: ADT).

→ Tip

You can download the latest available ADT plugin from the update site <https://tools.hana.ondemand.com/#abap>

SAP Gateway

- SAP Gateway is properly configured in your ABAP system for activating and testing the resulting OData service. **More on this** on the SAP help portal: [SAP Gateway Foundation](#).

Authorizations

To reproduce the steps described in this guide, the user on SAP Netweaver Application Server with the following roles assigned is required:

- [SAP_BC_DWB_ABAPDEVELOPER](#)
- [SAP_BC_DWB_WBDISPLAY](#)

- [*/IWFND/RT_DEVELOPER*](#) (for SAP Gateway service development).

Knowledge

Basic knowledge of

- ABAP Core Data Services (CDS)
- BOPF programming model
- SAP Gateway (transaction SEGW).

3 Get Started

Contents

[Developing a Simple List Reporting App \[page 11\]](#)

3.1 Developing a Simple List Reporting App

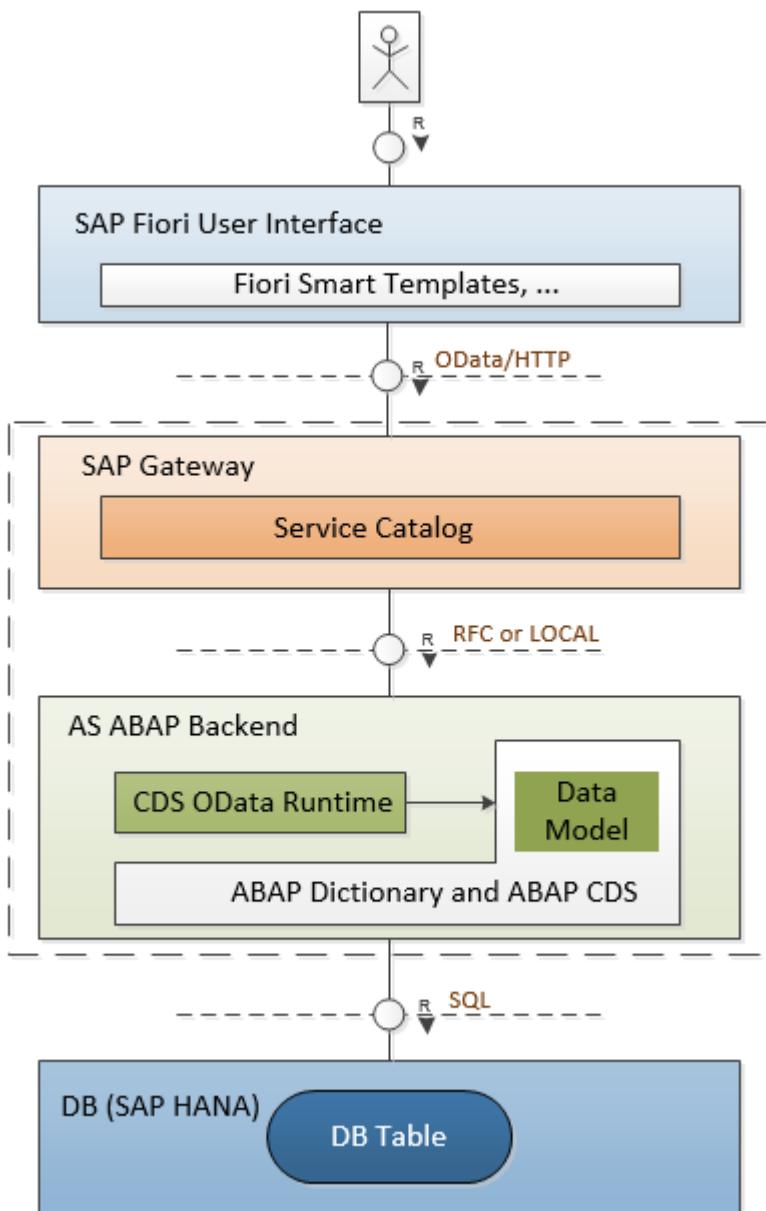
Introduction

In this **Getting Started** section, you have the opportunity - starting from an already existing data model - to develop a simple list - reporting scenario based on the standardized ABAP programming model for SAP Fiori. You will be guided step-by-step through the new development infrastructure, which includes technologies such as Core Data Services (CDS), SSDL, and SAP Gateway. In this scenario, the assumption is that the business logic is already provided with the FPM application model and that it allows users to create and manage sales orders. In the first step, you will use the well-known ABAP Development Tools to implement a CDS consumption view as a new data model layer, using a data source that is already provided with the EPM application. From within your development environment, you can continue with the next step towards consumption of the corresponding OData service. Here, you will use the CDS tools as part of the ABAP development environment to generate all artifacts that are required for service activation in the SAP Gateway hub. As soon as the OData service is activated in the SAP Gateway, it is ready for consumption through an OData client, such as an SAP Fiori app. Therefore, in our final step, we are going to build a Fiori UI using Smart Templates as Fiori building blocks. Finally, you can test the resulting list-reporting app within the SAP Fiori Launchpad environment.

Video Available

Watch the corresponding video available on our [SCN page](#)!

Architecture Overview



Architecture components of the sample application to be developed

Objectives

By the end of this Getting Started section, you will be able to...

- Create a DDL source in ABAP Development Tools for defining an ABAP CDS view
- Implement an ABAP CDS view for a simple data model based on existing data definition from EPM scenario
- Understand some basic CDS view annotations
- Perform a so-called auto-exposure process, which automatically exposes the CDS view as an OData service to the SAP Gateway hub

- To activate and test the exposed OData service in SAP Gateway
- Define a SAP Fiori Smart Template that implements the consumption layer
- Run the resulting app in Fiori Launchpad

3.1.1 Define a Data Model Based on CDS Views

To define a data model based on the ABAP CDS view concept, you first need to create a DDL source as the relevant ABAP repository object using a wizard in ABAP Development Tools.

Task 1: [Create a Data Definition for CDS View \[page 13\]](#)

In the second step, you will implement an elementary CDS view from scratch by defining a simple query for sales order items based on a single data source from EPM demo application.

Task 2: [Implement the CDS View as Data Model \[page 15\]](#)

In the last task of this section, you have the option to try out the test environment for verifying the output (result set) of the CDS view you have just implemented.

Task 3: [Verify the Result Set in the Data Preview Tool \[page 18\]](#)

3.1.1.1 Create a Data Definition for CDS View

Prerequisites

You need the standard developer authorization profile to create ABAP development objects.

Context

A Data Definition represents an ABAP development object used to define an ABAP CDS entity (for example, a CDS view). With the Data Definition you have the appropriate development object for the CDS view, which you can use to directly access the standard ABAP Workbench functionality, such as syntax check, activation, or connecting to the [Transport Organizer](#).

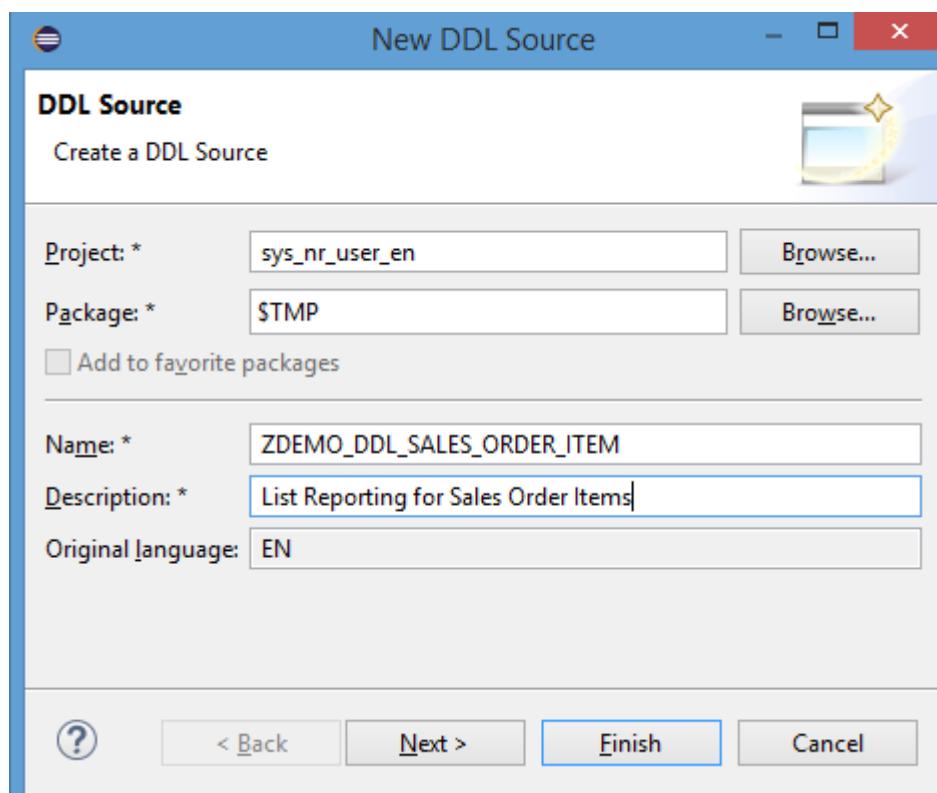
Procedure

1. Launch the [ABAP Development Tools](#).
2. In your ABAP project, select the relevant package node in the [Project Explorer](#).

3. Open the context menu and choose **New** **> Other ABAP Repository Object** **> Core Data Services** **> Data Definition** to launch the creation wizard.
4. In addition to the *Project* and *Package*, enter the *Name* (with due regard to your namespace) and the *Description* for the Data Definition to be created.

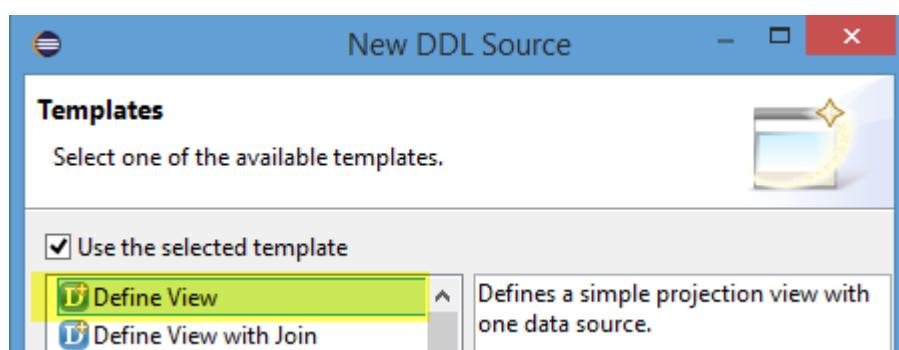
i Note

The maximum length for names of Data Definitions is 30 characters.



First wizard page when creating a Data Definition

5. Choose *Next*.
6. Assign a transport request and choose *Next*.
7. Select the *Define View* template to speed up the view definition.

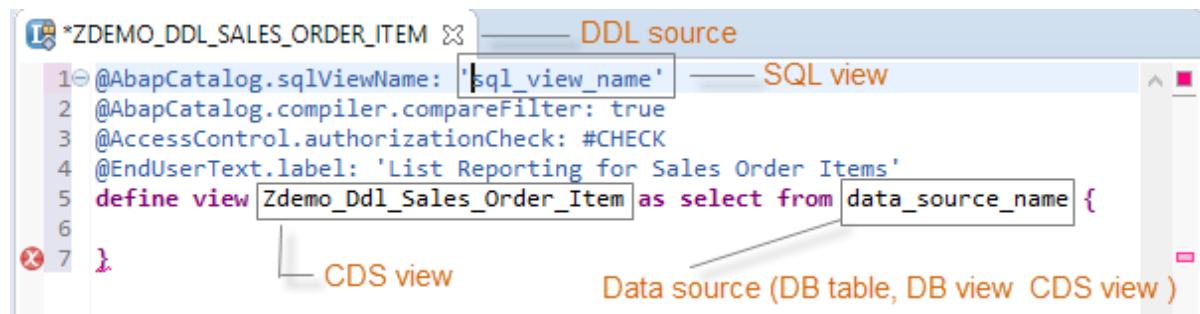


Selecting a template for CDS view with single data source

8. Choose *Finish*.

Results

In the selected package, the ABAP back-end system creates an inactive version of a Data Definition and stores it in the ABAP Repository. As a result of this creation procedure, the Data Definition editor will be opened. The generated source code already provides you with the necessary view annotations and adds placeholders for names of the Dictionary SQL view, the actual CDS view, and for the data source for query definition.



The screenshot shows the ABAP Data Definition (DDL) editor with the title bar "ZDEMO_DDL_SALES_ORDER_ITEM". The editor displays the following code:

```
1 @AbapCatalog.sqlViewName: 'sql_view_name' — SQL view
2 @AbapCatalog.compiler.compareFilter: true
3 @AccessControl.authorizationCheck: #CHECK
4 @EndUserText.label: 'List Reporting for Sales Order Items'
5 define view Zdemo_Ddl_Sales_Order_Item as select from data_source_name {
6
7 }
```

Annotations in the code highlight the following components:

- "sql_view_name" is highlighted in a box labeled "SQL view".
- "data_source_name" is highlighted in a box labeled "Data source (DB table, DB view CDS view)".
- The entire code block is labeled "CDS view".

The generated template code in the DDL editor

3.1.1.2 Implement the CDS View as Data Model

Context

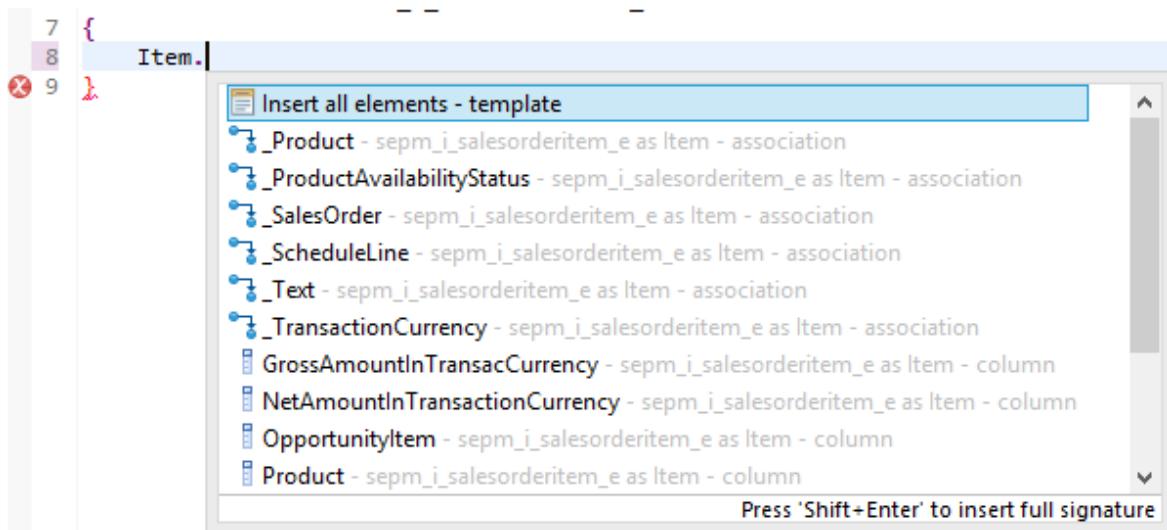
In this step, you will implement a projection view as a new data model using a data source that is already predefined in the EPM demo scenario.

Procedure

1. If you have not yet already done so, open the newly created DDL source in the text-based DDL editor.
2. Specify the names of the...
 - a. SQL view to be generated in the ABAP Dictionary: **ZDEMO_SOI**
 - b. Actual CDS view: **ZDEMO_CDS_SalesOrderItem**
3. In the SELECT statement, enter the predefined CDS view **SEPM_I_SalesOrderItem_E** as data source and define **Item** as alias name for the data source.
... select from SEPM_I_SalesOrderItem_E as Item {
4. Add the **Item** fields to the SELECT list (as they are required for new projection view) and assign the alias names to each item field as follows:

→ Tip

Whenever you insert table or view fields in the SELECT list, you can avail of the Content Assist functionality in the DDL editor.



Inserting fields by means of semantic auto-completion

```

Item.SalesOrder           as SalesOrderID,
Item.SalesOrderItem       as ItemPosition,
Item._SalesOrder._Customer.CompanyName as CompanyName,
Item._Product             as Product,
Item.TransactionCurrency as CurrencyCode,
Item.GrossAmountInTransacCurrency as GrossAmount,
Item.NetAmountInTransactionCurrency as NetAmount,
Item.TaxAmountInTransactionCurrency as TaxAmount,
Item.ProductAvailabilityStatus as ProductAvailabilityStatus

```

- To document the key semantics of the new data model, define the `SalesOrderItem` and the `ItemPosition` fields as KEY elements in the current CDS view:

```

...
key Item.SalesOrder           as SalesOrderID,
key Item.SalesOrderItem       as ItemPosition,
...

```

- Add the currency semantics to the corresponding `Item` elements.

→ Tip

To insert an annotation, type the first letter(s) of the annotation and then press **CTRL** + **Space**.

```

key Item.so_item_pos          as ItemPosition,
@Semantics.cu|                |
Item.currency                  @ currencyCode: true - annotation
Item.gross_am
Item.net_amou
Item.tax_amou

```

Press 'Shift+Enter' to insert full signature

Adding @Semantics annotation to an element

```

...
@Semantics.currencyCode: true
Item.TransactionCurrency      as CurrencyCode,
@Semantics.amount.currencyCode: 'CurrencyCode'

```

```

Item.GrossAmountInTransacCurrency           as GrossAmount,
@Semantics.amount.currencyCode: 'CurrencyCode'
Item.NetAmountInTransactionCurrency        as NetAmount,
@Semantics.amount.currencyCode: 'CurrencyCode'
Item.TaxAmountInTransactionCurrency       as TaxAmount,
...

```

Results

The resulting source code for the CDS view is the following:

```

@AbapCatalog.sqlViewName: 'ZDEMO_SOI'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'List Reporting for Sales Order Item'

define view ZDEMO_CDS_SalesOrderItem
as select from SEPM_I_SalesOrderItem_E as Item
{
key Item.SalesOrder                    as SalesOrderID,
key Item.SalesOrderItem                as ItemPosition,
Item._SalesOrder._Customer.CompanyName as CompanyName,
Item.Product                         as Product,
@Semantics.currencyCode: true
Item.TransactionCurrency              as CurrencyCode,
@Semantics.amount.currencyCode: 'CurrencyCode'
Item.GrossAmountInTransacCurrency    as GrossAmount,
@Semantics.amount.currencyCode: 'CurrencyCode'
Item.NetAmountInTransactionCurrency as NetAmount,
@Semantics.amount.currencyCode: 'CurrencyCode'
Item.TaxAmountInTransactionCurrency as TaxAmount,
Item.ProductAvailabilityStatus      as ProductAvailabilityStatus
}

```

The source code above is used to define quite a simple CDS view named `ZDEMO_CDS_SalesOrderItem`. This view is implemented by means of a query for performing a `SELECT` statement, where the predefined CDS view `SEPM_I_SalesOrderItem_E` is used as the data source. The select list includes a set of fields that are relevant for the new sales order item projection view. The `KEY` elements in the selection list are used to define the key field semantic of the CDS view. The element `CurrencyCode` is defined as currency key using the annotation `@Semantics.currencyCode: true`. The annotation `@Semantics.amount.currencyCode: 'CurrencyCode'` defines each amount element as a currency field.

When the DDL source is activated, the following objects are created in ABAP Dictionary:

- The actual entity of the CDS view `ZDEMO_CDS_SalesOrderItem`
- An SQL view `ZDEMO_SOI`

3.1.1.3 Verify the Result Set in the Data Preview Tool

Prerequisites

The DDL source is syntactically correct and has been activated.

Context

Now you have the option to launch the test environment (Data Preview tool), which will enable you to verify the result set of a CDS view.

Procedure

1. In the DDL editor, position the cursor somewhere in the CDS source code.
2. Open the context menu and choose ► *Open With* ► *Data Preview* ▾.

Results

Since the CDS view does not require any parameters, the Data Preview displays the result set of the data selection query directly.

The screenshot shows the SAP Data Preview tool interface. The title bar displays the table name 'ZDEMO_CDS_SALESORDERITEM'. Below the title bar, there is a breadcrumb navigation bar showing 'ZDEMO_CDS_SALESORDERITEM'. The main area is a grid table with the following columns: SalesOrderID, ItemPosi..., Product, Cu..., GrossAmo..., NetAm..., and Tax. The table contains five rows of data:

SalesOrderID	ItemPosi...	Product	Cu...	GrossAmo...	NetAm...	Tax
0500000000	0000000010	HT-1000	EUR	1137.64	956.00	181.64
0500000000	0000000020	HT-1001	EUR	2972.62	2498.00	474.62
0500000000	0000000030	HT-1002	USD	3736.60	3140.00	596.60
0500000000	0000000040	HT-1003	EUR	3927.00	3300.00	627.00
0500000000	0000000050	HT-1007	USD	1067.43	897.00	170.43

Result sets in the Data Preview tool

3.1.2 Generating OData Service With Auto-Exposure

OData Exposure

With the concept of auto-exposure, a new and simple way of creating OData services has been introduced. Here, the OData model definition as well as the OData service runtime is provided generically, based on *SADL* (*Service Adaptation Description Language*).

The requirement here is that the annotation `@OData.publish: true` is specified at the CDS data model (CDS consumption view) level as follows:

```
@AbapCatalog.sqlViewName: 'SQL_VIEW_SAMPLE'  
...  
@OData.publish: true  
define view CDS_VIEW_NAME as select from  
...  
}
```

→ Remember

We recommend using the **auto-exposure** option in the case of elementary data model compositions: for example, if you defined the entire CDS data model based on a root CDS view so that several other CDS views are children of this root CDS view. In addition, the CDS views of this composition might have associations to other entities. In cases like this, all the CDS views together form a quite simple data model composition that you want to expose as an OData service, together with first the level of associations.

Do not use the **auto-exposure** option if:

- Your data model composition is more complex and you need to include deeper association levels in your OData service.
- You want to generate a hybrid scenario where implementations are based on CDS views and on custom logic.

More on this: [Exposing CDS Entities as OData Service \[page 339\]](#)

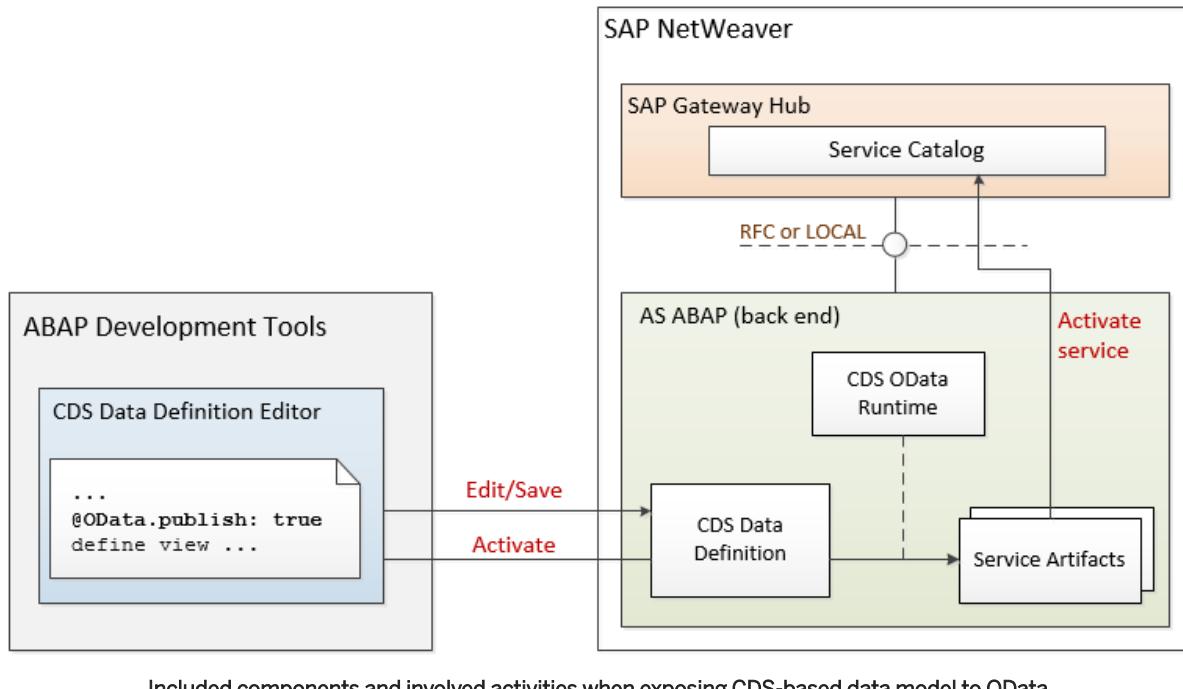
The Auto-Exposure Process

The following figure depicts the main components of the auto-exposure process and refers to the most important activities that are involved:

Starting in *ABAP Development Tools*, you open the relevant CDS data definition object where the CDS view in question is implemented. After you have added the OData annotation to the CDS view, you can trigger the activation of the entire CDS data definition (that serves as transportable development object). *ABAP Development Tools* delegates the activation request to *SADL*. *SADL* framework generates several SAP Gateway artifacts that are stored in the back end of the application server AS ABAP and are required for OData service activation in the SAP Gateway hub system.

In the next step, you need to launch the transaction `/IWFND/MAINT_SERVICE` to add the OData service to the service catalog of the SAP Gateway hub. In this way, the actual OData service is created in the SAP Gateway and the back end system is assigned as a data source using the system alias.

As soon as the OData service is activated in the SAP Gateway hub, it is ready for consumption through an OData client, such as an SAP Fiori app.



Developer-Relevant Tasks

Task 1: Generate Service Artifacts From a CDS View [page 21]

As a result of this task, several service artifacts are generated in the back end of the application server.

Task 2: Activate OData Service in the SAP Gateway Hub [page 22]

As a result of this task, the OData service is added to service catalog of the SAP Gateway hub and is ready for consumption.

Task 3: Test the Activated OData Service [page 26]

3.1.2.1 Generate Service Artifacts From a CDS View

Here, we want to expose the newly developed data model to OData as a canonical OData service using the so-called **auto-exposure** approach. In this step, you will use the CDS tools as part of the ABAP development environment to generate all artifacts that are required for service activation in the SAP Gateway hub.

Prerequisites

The following rules need to be fulfilled for a CDS view:

- The CDS view is defined and the source code of the corresponding data definition source is syntactically correct.
- At least one `KEY` element is defined in the `SELECT` list of the CDS view. Otherwise, the OData service generation will fail due to missing `KEY` elements in the CDS view.
- The name of the CDS view in question does not exceed the maximum length of 26 characters.

→ Tip

If this is not the case, use the refactoring function for renaming.

Procedure

1. In the *ABAP Development Tools*, open the editor with the relevant data definition where your CDS view is defined.
2. In editor, navigate to the source code line before the `DEFINE VIEW` statement.
3. Insert the following OData annotation to the CDS view:

```
@OData.publish: true
```

The screenshot shows the ABAP Development Tools editor with the code for a CDS view named `*ZDEMO_DDL_SALES_ORDER_ITEM`. Line 6 contains the annotation `@OData.publish: true`. A tooltip box is overlaid on the code, containing the text `@ publish: true - annotation` and the instruction `Press 'Shift+Enter' to insert full signature`.

```
1@AbapCatalog.sqlViewName: 'ZDEMO_SOI'
2@AbapCatalog.compiler.compareFilter: true
3@AccessControl.authorizationCheck: #CHECK
4@EndUserText.label: 'List Reporting for Sales Order Items'
5
6@OData.
7
8define
9    as
10{
11    key
12    key
13
```

4. Save the new editor content and activate the data definition of the CDS view.

More on this:

Results

ABAP Development Tools delegates the activation request to the SADL (Service Adaptation Description Language) framework. SADL, in turn, generates several SAP Gateway artifacts that are stored in the back end of the application server AS ABAP and are required for OData service activation in the SAP Gateway hub later on:

- The actual service object with the technical name <CDS_VIEW>_CDS. You can find it as SAP Gateway Business Suite Enablement - Service object (object type: R3TR_IWSV)
- An SAP Gateway model (object type: R3TR_IWMO) with the name <CDS_VIEW>_CDS
- An ABAP class <NAME_SPACE>CL_<CDS_VIEW> that is used to provide model metadata to the SAP Gateway service.

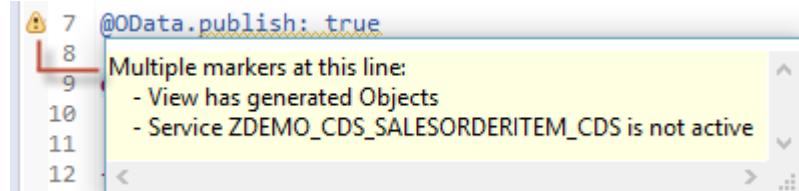
All generated service artifacts are automatically added to the same ABAP package (and also to the same transport request) as the underlying CDS view.

i Note

We recommend that you do not change the generated service artifacts.

The warning decorator  within the editor ruler indicates a successful generation process. If you move the cursor over the decorator, an info screen informs you that the service artifacts have been generated for the CDS view. However, the service must be activated in the SAP Gateway hub for exposure. This will be done manually in a separate step using the transaction /IWFND/MAINT_SERVICE.

More on this: [Activate OData Service in the SAP Gateway Hub \[page 22\]](#)



Quick info after a successful generation of service artifacts

3.1.2.2 Activate OData Service in the SAP Gateway Hub

After you have generated the SAP gateway artifacts in your development environment, you can continue with the next step towards consumption of the OData service. For this you need to activate the OData service in the SAP Gateway hub. In other words: The OData service has to be enabled in SAP Gateway which establishes a mapping between the technical OData service name and the corresponding back end service.

Prerequisites

- The service artifacts are successfully created in the back end of the application server AS ABAP.
- The SAP Gateway hub (target system for the OData service) is set up and configured for managing OData services. **More on this:** [Quick Configuration](#)
- You have the authorization for using activation functionality in the transaction /IWFND/MAINT_SERVICE.

Procedure

1. Open the SAP GUI for the relevant ABAP project by starting the SAP GUI Launcher *ABAP Development Tools* (icon  in the toolbar). Within the embedded SAP GUI, you are able to access the complete functionality of the classic ABAP Workbench.

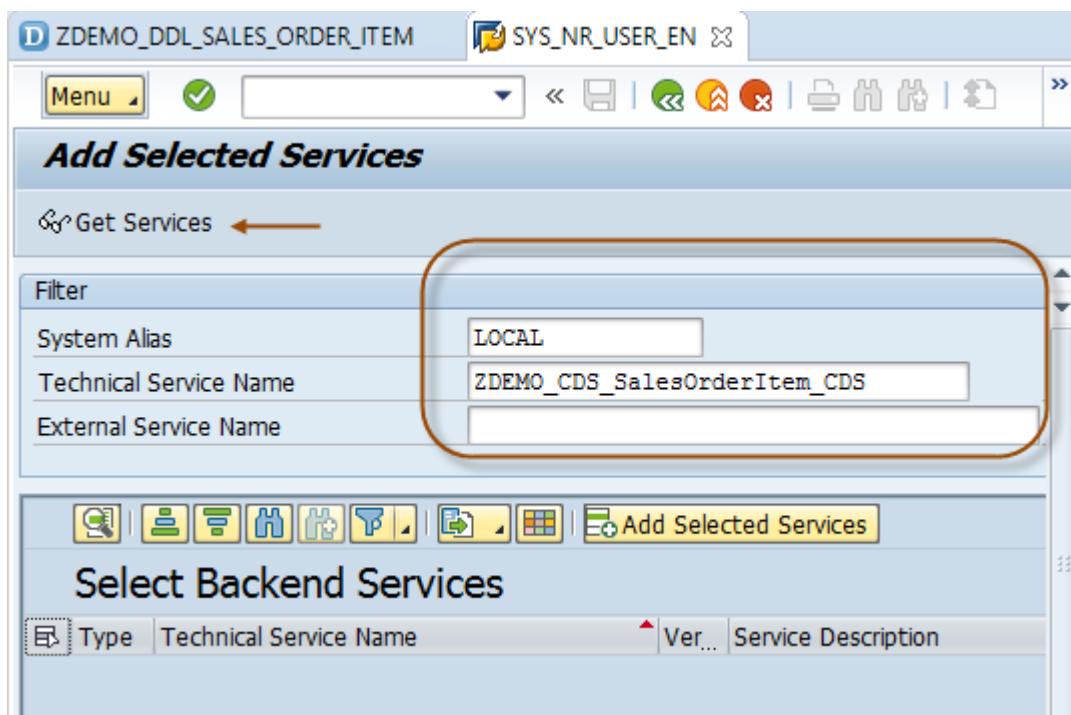
Note

Alternatively, you log in to the corresponding development system using the SAP GUI mode.

2. In the command field, enter the transaction code **/IWFND/MAINT_SERVICE**.

The entry screen of the transaction displays in the target system all activated Gateway services in the *Service Catalog* and allows you to add new services.

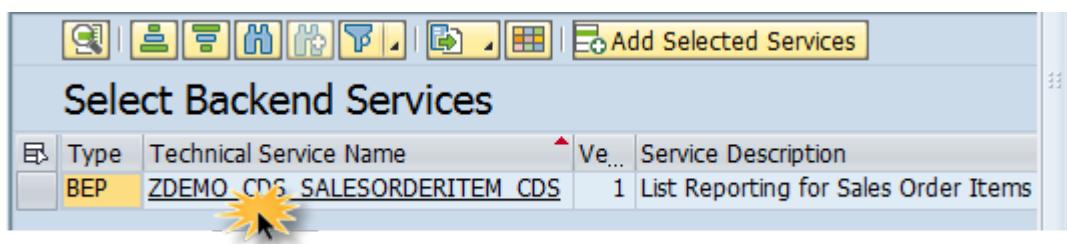
3. Click the *Add Service* button in the toolbar.
4. Enter the *System Alias* of your front-end server.
5. Enter the *Technical Service Name* (in our case: `<CDS_VIEW>_CDS`).



Editing details for the service to be activated in the Gateway hub (in our case: LOCAL system)

6. Click the *Get Services* button in the toolbar to request the services available.

As a result, the service is displayed for selection.
7. Select the service created as a result of last procedure and then choose *Add Selected Services* or alternatively click the object link for further selection.



Adding a service to the Gateway service catalog

The [Add Service](#) dialog that appears, suggests already the name <CDS_VIEW>_CDS for the *Technical Service*, and for the *Technical Model*.

i Note

We recommend that you do not change the suggested technical names <CDS_VIEW>_CDS for the service and the model.

The dialog that now appears informs you that the model metadata for the Gateway service is going to be created.

- Specify the package for service activation.

Service	
Technical Service Name	ZDEMO_CDS_SALESORDERITEM_CDS
Service Version	1
Description	List Reporting for Sales Order Items
External Service Name	ZDEMO_CDS_SALESORDERITEM_CDS
Namespace	
External Mapping ID	
External Data Source Type	C

Model	
Technical Model Name	ZDEMO_CDS_SALESORDERITEM_CDS
Model Version	1

Creation Information	
Package Assignment	\$TMP Local Object

ICF Node	
<input checked="" type="radio"/> Standard Mode	<input type="radio"/> None

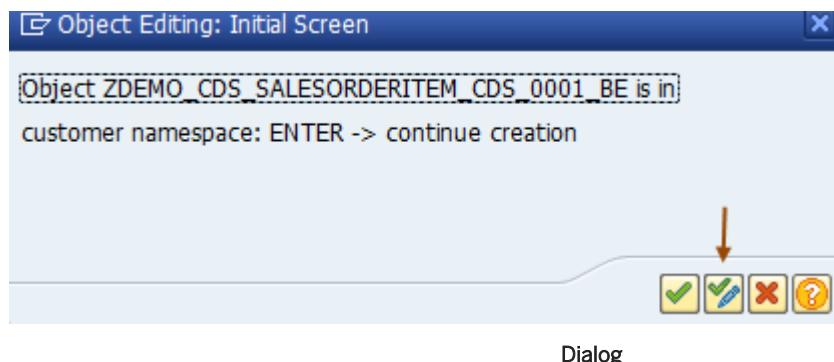
Adding a service with a \$TMP package as example

→ Remember

Assign a productive package (none \$TMP package) only if you want to transport the service activation within in your system landscape.

- Leave the other details on the dialog screen unchanged, and choose (Continue).

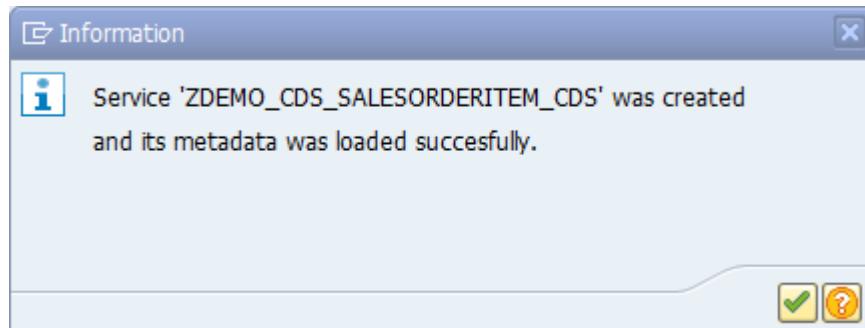
An information dialog appears.



Dialog

- In the information dialog, proceed with (Further for All).

The dialog that now appears informs you that the model metadata for the Gateway service has been created successfully in SAP Gateway.



Dialog that informs you about a successful service creation (example for \$TMP package)

- To complete the service activation, and choose (Continue)

Results

As a result of the successful procedure, the OData service has been activated in the Gateway hub. During this activation, further new objects have been created:

- SAP Gateway: Service Group Metadata object (object type: R3TR_IWSG) with the name Z<CDS_VIEW>_CDS_<VERSION> that object type IWSG that includes the service group metadata of the Gateway represents the actual OData service.
- An SAP Gateway: Model Metadata object (object type: R3TR_IWOM) with the name Z<CDS_VIEW>_CDS_<VERSION>_BE that represents the structure of the actual OData service.
- In addition, an ICF node (object type: SICF) for the OData service has been generated.

The OData service is also added to the Service Catalog of transaction /IWFND/MAINT_SERVICE and has the status *Active* in the current SAP Gateway hub.

Service Catalog				
Type	Technical Service Name	V...	Service Description	External
BEP	ZDEMO_CDS_SALESORDERITEM_CDS	1	List Reporting for Sales Order Ite...	ZDEMO

The new OData service is displayed in the Service Catalog

3.1.2.3 Test the Activated OData Service

Prerequisites

To run the newly developed and activated OData service, we assume that...

- Authorization defaults are assigned to the new service using transaction **SU24** (SAP customers) or **SU22** (SAP internal). [More on this: Assigning Authorization Defaults to OData Services \[page 167\]](#)
- Your authorization role is extended in transaction **PFCG** for using the new service. [More on this: Assigning Authorizations to Roles \[page 169\]](#)

→ Remember

When developing new OData services within the SAP internal development landscape, usually it is not necessary for you, as ABAP developer at SAP, to take care for these authorization requirements.

Starting from the Service Catalog

To test the new service from the *Service Catalog*, select the newly created service and choose *SAP Gateway Client*.

Service Catalog				
Type	Technical Service Name	V...	Service Description	External
BEP	ZDEMO_CDS_SALESORDERITEM_CDS	1	List Reporting for Sales Order Ite...	ZDEMO

ICF Nodes	
Status	ICF Node
OK	ODATA

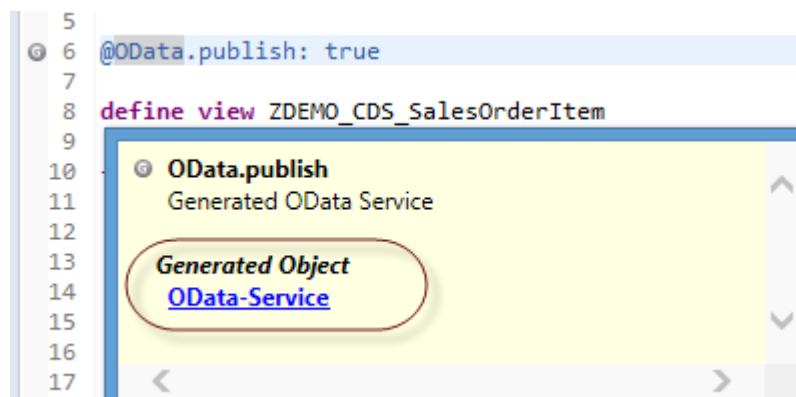
Testing the service in SAP Gateway

→ Tip

If you want to know more about the SAP Gateway Client, see [SAP Gateway Client](#).

Starting in ABAP Development Tools (in the case of an embedded Gateway)

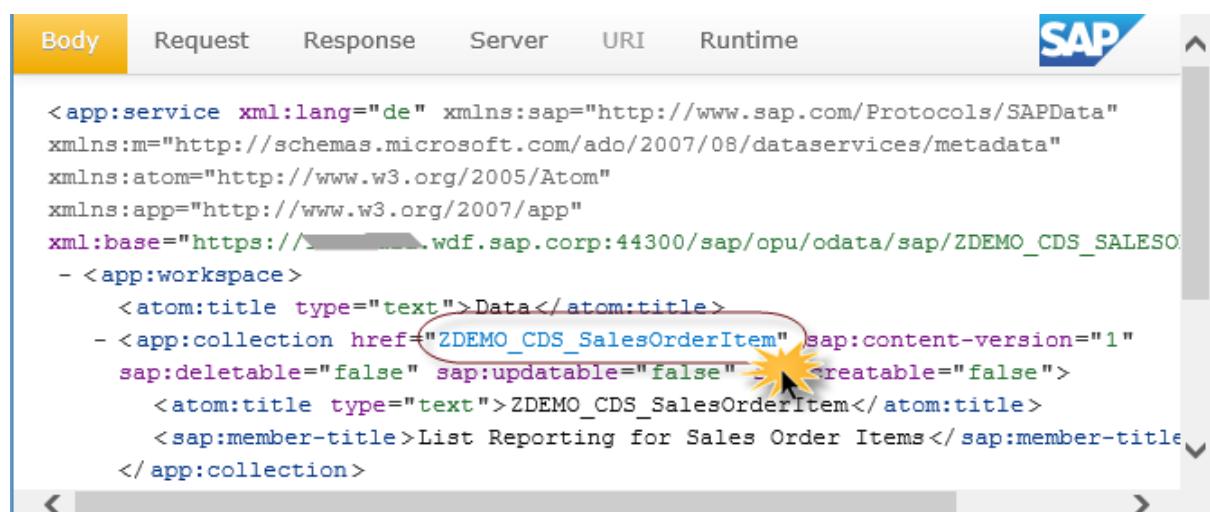
If the SAP Gateway and the back end are in the same ABAP system and you open the editor with the data definition of the relevant CDS view again, you will detect that a new decorator  indicates a successfully activated OData service. If you move the cursor over the decorator, an info screen provides you with a link to the OData service.



The screenshot shows a portion of an ABAP code editor. Line 6 contains the annotation `@OData.publish: true`. A tooltip has appeared over line 10, which contains the annotation `@ OData.publish`. The tooltip displays the message "Generated OData Service". Below the tooltip, there is a callout bubble with the text "Generated Object" and a link "OData-Service" underlined in blue. The entire tooltip and callout bubble are highlighted with a red oval.

```
5
6 @OData.publish: true
7
8 define view ZDEMO_CDS_SalesOrderItem
9
10 @ OData.publish
11   Generated OData Service
12
13 Generated Object
14   OData-Service
15
16
17
```

Quick info after successful service activation in the case of an embedded SAP Gateway



The screenshot shows the SAP Fiori Elements test interface. The top navigation bar includes tabs for Body, Request, Response, Server, URI, and Runtime, with the Body tab currently selected. The main area displays an XML representation of an OData service endpoint. A specific part of the XML is highlighted with a red oval and a yellow starburst effect, specifically the `sap:deletable="false" sap:updatable="false" sap:createable="false"` attributes on a collection element. The XML code is as follows:

```
<app:service xml:lang="de" xmlns:sap="http://www.sap.com/Protocols/SAPData"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app"
  xml:base="https://[REDACTED].wdf.sap.corp:44300/sap/opu/odata/sap/ZDEMO_CDS_SALESO
  - <app:workspace>
    <atom:title type="text">Data</atom:title>
    - <app:collection href="ZDEMO_CDS_SalesOrderItem" sap:content-version="1"
      sap:deletable="false" sap:updatable="false" sap:createable="false">
        <atom:title type="text">ZDEMO_CDS_SalesOrderItem</atom:title>
        <sap:member-title>List Reporting for Sales Order Items</sap:member-title>
      </app:collection>
```

Testing the OData service

3.1.3 Consume Business Data Using SAP Fiori Elements

In this final procedure, we are going to build a quite simple list-reporting Fiori UI without any code using the OData service (that you created in the last procedure) as the data source. In the course of the UI development,

we will make use of **SAP Fiori elements** as UI building blocks and test the resulting list-reporting app within the **SAP Fiori launchpad** environment.

More on this:

Developer-Relevant Tasks

[Task 1: Create a Project for a Fiori App in the Web IDE \[page 28\]](#)

[Task 2: Run the New App in the Fiori Launchpad \[page 31\]](#)

Related Information

[Developing Apps with SAP Fiori Elements](#)

3.1.3.1 Create a Project for a Fiori App in the Web IDE

Prerequisites

- You are registered in the cloud for using SAP Web IDE in the cloud.
More on this: [SAP web IDE - Getting Started](#)
- You have the authorization for using the SAP Web IDE (role WebIDEPPermission).
More on this: [SAP web IDE - Assigning Roles to Permissions](#)
- The system connection to the relevant SAP Gateway system is configured using the [SAP HANA Cloud Connector](#) and available from within the SAP Web IDE.

Procedure

To create a project based on a specific template, proceed as follows:

1. Launch the SAP Web IDE in a web browser.
2. From the *File* menu, choose  *New > Project from Template* .
3. Select *List Reporting Application* as the template category for project generation and choose *Next*.
4. Enter a *Project Name*, *Title*, and the *Application Component Hierarchy* and choose *Next*.

2 Basic Information

Project Name*

Web_Project_for_SOI

App Descriptor Data

Title*

Web Project for Sales Order Items

Namespace

Description

Application Component Hierarchy*

zmh

SAP Fiori ID

Path for ABAP Deployment

[Previous](#)[Next](#)

Editing basic project information in the new project wizard of the SAP Web IDE

- To specify the *Data Connection*, choose *Service Catalog*, select the Gateway system from the list, and then choose the service that you created in the last procedure. Now choose *Next*.

New Smart Template Application

Data Connection

Choose a service from one of the sources listed below.

Sources	Service Information	SAP Gateway - system destination
Service Catalog	000	ZDEMO_CDS_SALESO
Workspace		X
File System		
Service URL		

Specifying the service from SAP Gateway system

- Since the OData service received from SAP Gateway does not provide any UI annotations, the list of annotation files is empty. Therefore, this step is not relevant for your template and you can ignore the annotation selection. Now choose [Next](#).

+ Add Annotation Files

Rank	Name	Source
No data		

Annotation Selection: In our special case, the OData service does not provide any available UI-related metadata

- Assign the name of the original data model (CDS consumption view) to [Data Binding](#) template parameter. Then choose [Next](#).

Template Customization

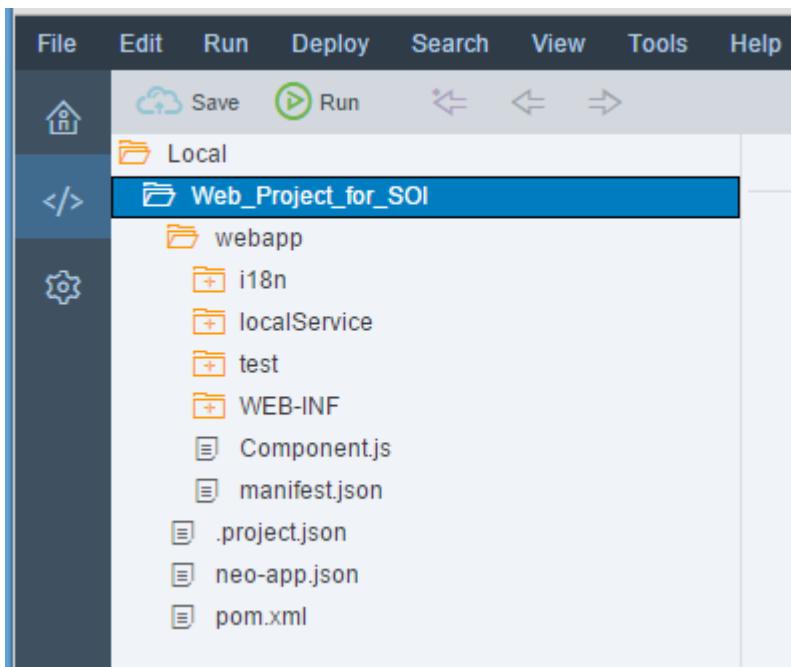
Data Binding	OData Collection*	<input type="text" value="ZDEMO_CDS_SalesOrderItem"/>
OData Navigation	<input type="text" value="OData navigation attribute to a collection"/>	

Assigning the CDS view fro consumption

- Confirm your project information in the last wizard page and choose [Finish](#).

Results

The project wizard creates the SAP Fiori app based on [Fiori Elements](#). The SAP Web IDE automatically opens the new project in the workspace under a folder with the project name that you specified and displays the project tree:



Project tree created in the SAP Web IDE

3.1.3.2 Run the New App in the Fiori Launchpad

Based on the OData service metadata, you have created a project for your SAP Fiori app in SAP Web IDE and want to run the new app in the Fiori Launchpad.

Prerequisites

To run the newly developed app, we assume that...

- Authorization defaults are assigned to the new service using transaction **SU24** (SAP customers) or **SU22** (SAP internal). **More on this:** [Assigning Authorization Defaults to OData Services \[page 167\]](#)
- Your authorization role is extended in transaction **PFCG** for using the new service. **More on this:** [Assigning Authorizations to Roles \[page 169\]](#)

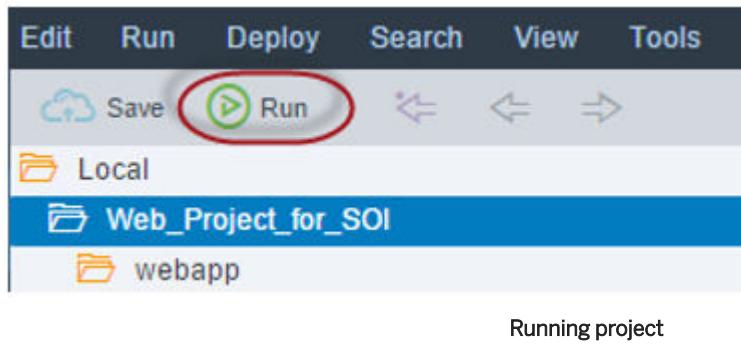
Context

An app based on **SAP Fiori elements** uses predefined template views and controllers that are provided centrally, so that no application-specific view instances are required. The SAP UI5 runtime interprets metadata and annotations of the underlying OData service and uses the corresponding views for the SAP Fiori app at start up. To run the resulting list-reporting app for sales order items, we will be using a local sandbox for the **SAP Fiori launchpad** as a simplified environment that you can use for local testing. This ensures that the app can be embedded properly into the **SAP Fiori launchpad**.

Procedure

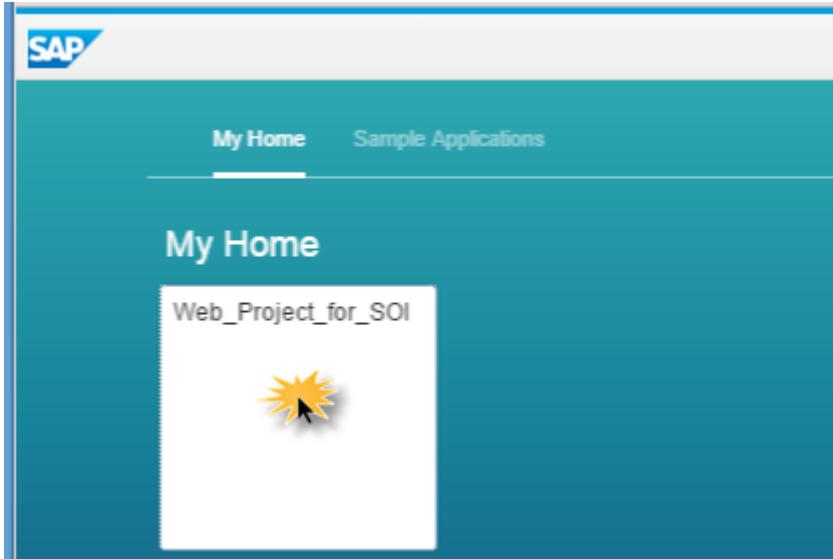
To run and test your app without defining a new run configuration, proceed as follows:

1. In the SAP Web IDE, select the project you want to run.
2. In the main toolbar, click the *Run* button.



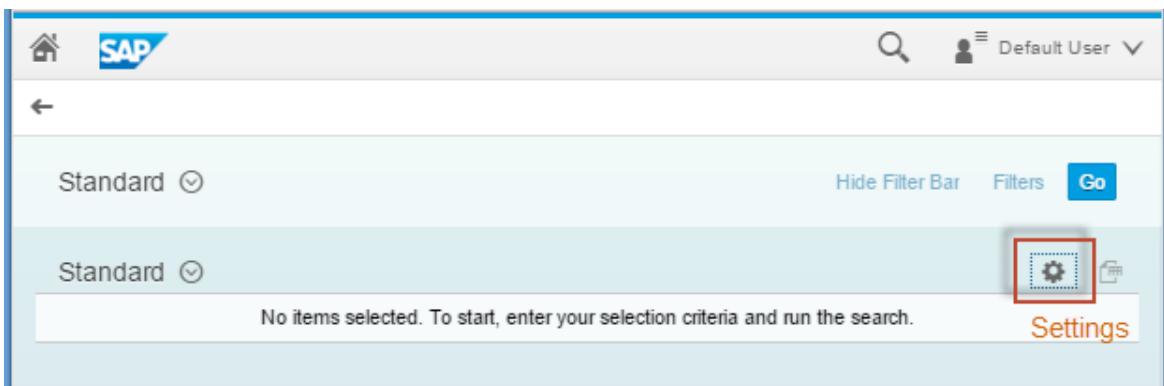
The Web IDE opens the SAP Fiori launchpad.

3. In the SAP Fiori Launchpad, select the app you want run.

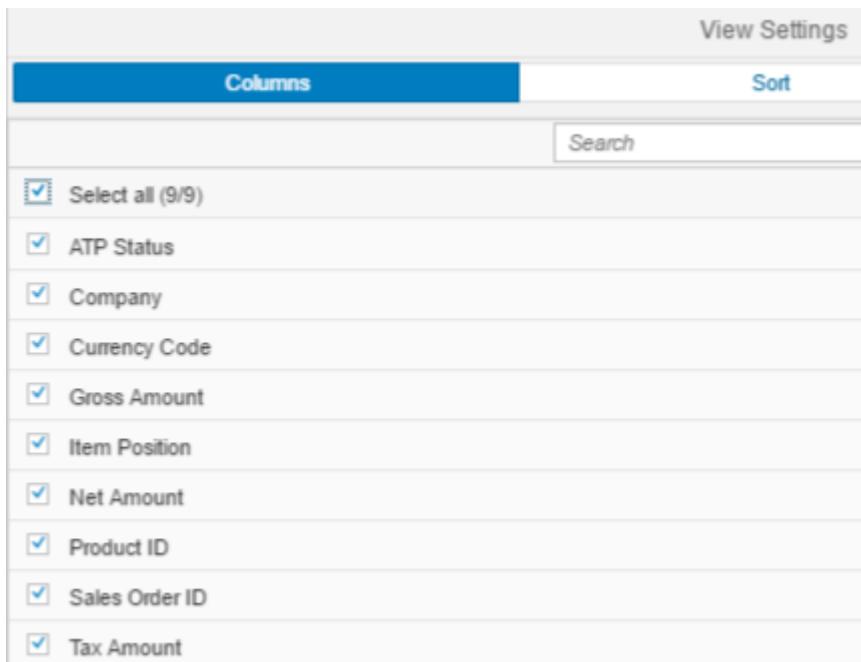


Selecting app in the launchpad

4. In the screen that appears, choose the *Settings* button.



5. In the *Settings* dialog, click on the *Select all* box and confirm with *OK*.



Selecting columns

6. Choose the *Go* button.

Results

Congratulations, you have successfully completed all the required steps!

The screenshot shows the SAP Fiori launchpad interface. At the top, there is a header bar with the SAP logo, a search icon, and a user dropdown set to "Default User". Below the header, the title "Web Project for Sales Order Items" is displayed. A filter bar is present with buttons for "Hide Filter Bar", "Filters", and a highlighted "Go" button. The main content area displays two rows of sales order item data in a table format. Each row includes columns for ATP Status, Company, Currency Code, Gross Amount, Item Position, and a detail icon. Below each row, specific item details are listed: Net Amount, Product ID, Sales Order ID, and Tax Amount.

ATP Status	Company	Currency Code	Gross Amount	Item Position	
SAP	EUR	1,137.64 EUR	10	>	
Net Amount: 956.00 EUR					
Product ID: HT-1000					
Sales Order ID: 500000000					
Tax Amount: 181.64 EUR					
SAP	EUR	2,972.62 EUR	20	>	
Net Amount: 2,498.00 EUR					
Product ID: HT-1001					
Sales Order ID: 500000000					
Tax Amount: 474.62 EUR					

The Fiori launchpad displays the sales order item data

4 Concepts

Contents

- [Draft Concept \[page 35\]](#)
- [Draft Consistency \[page 40\]](#)
- [Locking and Resume \[page 43\]](#)
- [Messages \[page 50\]](#)
- [Validations for CDS-Based Business Objects \[page 58\]](#)
- [Actions for CDS-Based BOPF Business Objects \[page 60\]](#)
- [Determinations for CDS-Based Business Objects \[page 63\]](#)

4.1 Draft Concept

Motivation

SAP's traditional applications are developed using **stateful** technologies, such as Floorplan Manager (FPM) for WebDynpro ABAP or the classic Dynpro technique. These stateful transactional applications rely on a server session along with application buffers that can fulfill client requests (user interactions with multiple backend round trips) until the user has saved the data changes and finished his/her work – as illustrated in figure (a) below. In stateful applications, data entry and data updates work on a temporary in-memory version of a business entity which is only persisted once it is sufficiently complete and consistent. The life time of this temporary version is tied to the UI session.

As an application developer, you might need to enable the end user to store changed data at any time in the backend and continue at a later point in time or to recover this data, even if the application client has crashed. This kind of scenario needs to support a **stateless** communication and requires a replacement for the temporary in-memory version of the business entity that is created or edited. This temporary version is kept on the database and is known as a "draft". As illustrated in figure (b) below, drafts are isolated in their own persistency and do not influence existing business logic until activated.

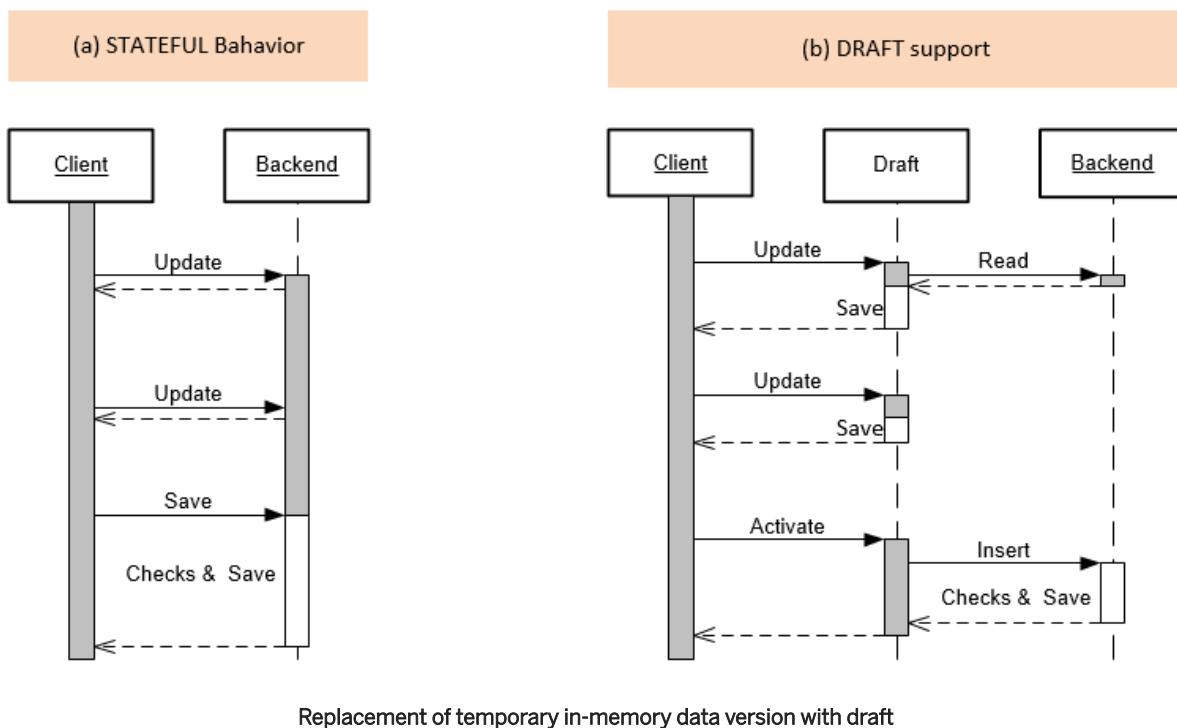
In addition to replacing UI sessions, the draft concept introduces many new opportunities:

- Draft data can be stored any time without having to meet any criteria regarding consistency or completeness, thus minimizing the risk of data loss due to network, server, or client failures.

- Work on a draft data can be suspended and resumed later, allowing work on complex entities to be spread over time without the risk of data loss.
- Work on draft data can be resumed on a different device.
- Work on draft data can be resumed by a different user.
- Multiple users can work alternatively on the same draft.
- Multiple users can collaborate on a draft and work on it concurrently.

! Restriction

At the present point in time however, SAP's implementation of the draft framework supports **exclusive draft** capabilities only. Therefore, the draft is locked for a specific (exclusive) user only and no collaborative activities are possible on one and the same draft version of data.



Draft Life Cycle (State Transitions)

As stated above, traditional business applications only store the most recent version of a business entity data in the database. This results in the following situations however:

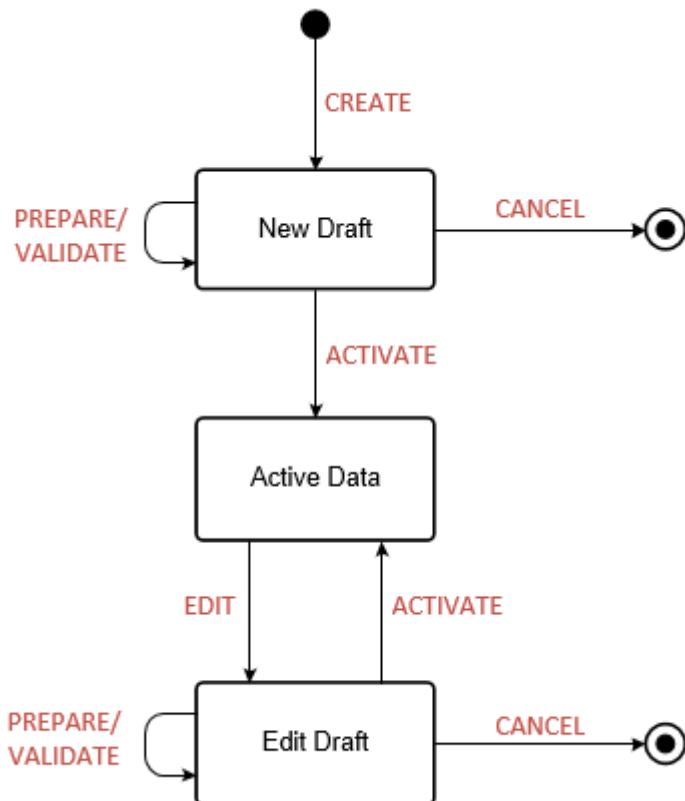
- A new business entity is created in transient memory only; no persistent version exists in the database.
- The most recent persistent version of the business entity exists in the database.
- If an existing business entity data is being edited by a user, two versions of this business entity exist in parallel: the transient version with changes made since editing started, and a persistent version with the content before editing started.

To base an application on stateless communication without feature loss, the following three states of business entities are required:

- New draft – for initial data (where no database version exists).

- Active data – corresponds to the database version of a traditional application.
- Edit draft – exists in parallel to the corresponding active data.

The figure below illustrates the states of the business entities and their transitions following various actions triggered by the user.



Life cycle of draft – state transitions

The life cycle of a business entity starts with the creation of a new draft. This initial draft can be empty or can contain default values derived from system preferences and related business entities, calculated values, or field control information. For a new sales order item that is being created for example, the item position can be calculated, and a product can be selected from the product catalog.

New user inputs or data updates for an existing business entity start with the creation of an edit draft version as a copy of active data. The new or changed data is stored in the draft persistence regardless of its validity or completeness. User inputs can also be immediately validated however. The consistency of the draft can be checked using the VALIDATION action. This will return messages, each with a reference to the critical part of the draft. The VALIDATION action is free from side-effects, meaning that it simply returns messages as a result of consistency checks. The draft can be enriched by calling a PREPARATION action. PREPARATION actions can have side-effects and modify the draft. They can also return messages as a result of consistency checks or the business logic triggered by them.

The draft version is converted into active data by calling the ACTIVATION action. This will implicitly trigger the VALIDATION action on the draft. The draft is also deleted after successfully copying its data into active data of the business entity.

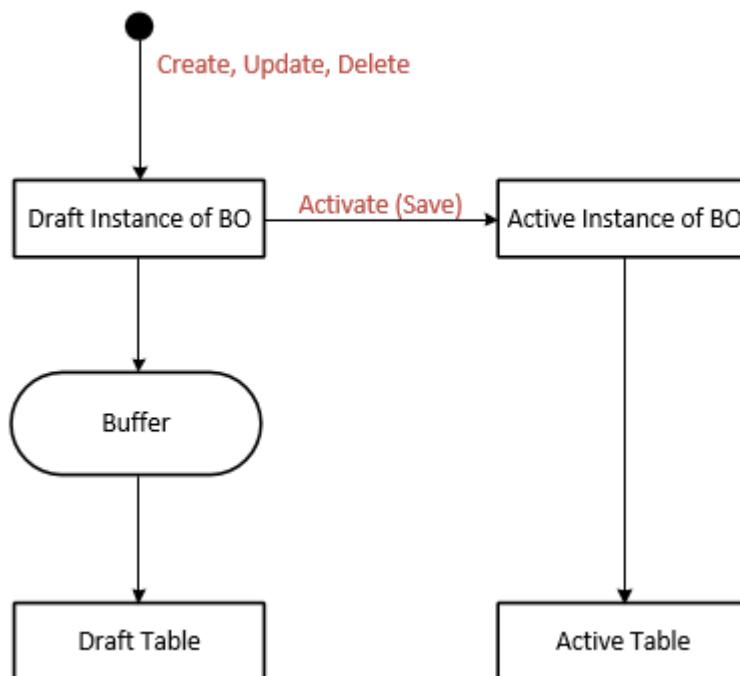
Cancelling new data entries or data updates can be achieved by deleting the draft using the `DELETE` action on the draft entity.

Draft Support When Using BOPF Business Objects

BOPF provides additional functionality for draft support at the business object level. In addition to the standard BOs, it is possible to create a specific shape of BOs that is suited for implementing draft qualities. Draft business objects provide code exits for implementing the draft contract and extending the BO functionality based on well-defined BOPF APIs.

Applications with draft support can deal with two BOPF business object instances:

- The **active instance** represents the state of the business object (sales order) instance that is stored in the active persistence (active table).
- The **draft instance** represents the temporary version (transient state) of a business object instance until it is permanently stored in the persistence layer as an active instance. The draft data is stored in the draft persistence (draft tables) until its transition to active data.



Draft support for BOPF business objects – transition from draft to active BO instance

Actions generated at root node level

In order to consume draft data in a generic manner, the BOPF framework by default creates standard draft actions (as mentioned above), such as `EDIT`, `ACTIVATION`, `PREPARATION`, and `VALIDATION`, for each draft business object.

The screenshot shows the SAP Fiori UI for the ZDEMO_I_SALESORDER_TP entity. The top navigation bar shows the path: ZDEMO_I_SALESORDER_TP > ZDEMO_I_SALESORDER_TP. Below the path is the title "Actions". At the top right of the main area are two buttons: "New..." and "Delete". Below these buttons is a search bar with the placeholder "type filter text". The main content is a table with two columns: "Name" and "Implementation Class". The table contains the following data:

Name	Implementation Class
LOCK_ZDEMO_I_SALESORDER_TP	ZCL_A_LOCK_DEMO_I_SALESORDER_T
DELETE_ZDEMO_I_SALESORDER_TP	ZCL_A_DELETE_DEMO_I_SALESORDER
EDIT	/BOBF/CL_LIB_A_EDIT
ACTIVATION	/BOBF/CL_LIB_A_ACTIVATION
VALIDATION	/BOBF/CL_LIB_A_VALIDATION
PREPARATION	/BOBF/CL_LIB_A_PREPARATION

Below the table are navigation tabs: Overview, Alternative Keys, Properties, Associations, Actions, Determinations, and Validations. The "Actions" tab is currently selected. A note below the tabs states: "Actions generated at root node level of the draft BO".

Actions generated at root node level of the draft BO

Draft-Enabled Associations

As stated above, every BO entity with draft support can have more than one version of data: The active instance and the draft instance. Consequently, when following an association from one draft-enabled entity to another one, the data returned by the association can be either the active or the draft version. Whether the association navigates to the active or the draft instance of the target entity depends on the association being draft-enabled or not.

A draft-enabled association retrieves active data if you follow it from an active source instance. It retrieves draft data if the association is followed from a draft source instance. On the other hand, an association that is not draft-enabled always retrieves active data even if the source instance is draft.

Some associations are draft-enabled by default. This is the case, when the associated entities belong to the same business object instance (identified by the same root key). All other associations need to be implemented to become draft-enabled. The implementation for draft-enablement is done by annotations on the respective association.

More on this: [Implementing Draft-Enabled Associations \[page 288\]](#)

Visualization of Draft in SAP Fiori UI

Draft versions of business objects data are made visible in SAP Fiori UI to the end user in order to distinguish them from the application's active data. The figure below displays the UI for a sales order processing app, where the end user can list all sales orders available in the system and perform all CRUD operations on sales order instances. The resulting Fiori app also provides draft-enabling capabilities. For each sales order instance for which draft data also exists, the indication *Draft* is added to the sales order ID field. An additional filter for *Editing Status* is also provided to the end user to indicate specific draft instances.

Besides the user interface design, the user interaction is adapted to draft qualities:

User input is therefore implicitly stored as a draft version. In the “minimize data loss” use case, the user has to decide at the end of an application whether s/he wants to activate or discard the draft data version. If the application ends in an uncontrolled way, the draft will be available when the user re-starts the application.

The screenshot shows a SAP Fiori application interface for managing sales orders. At the top left, there is a search bar and a dropdown menu labeled "Standard *". Below the search bar, there is a field for "Sales Order ID" with a placeholder "7000000000". To the right of the search bar, there is a "Search" button with a magnifying glass icon. On the far right of the header, there is a blue circular icon with a white arrow pointing outwards.

In the center of the screen, there is a modal window titled "Editing Status:" with a red border. This modal lists five status options: "All" (selected), "Own Draft", "Locked by Another User", "Unsaved Changes by Another User", and "Unchanged".

Below the modal, there is a table titled "Sales Orders (5) Standard". The table has three columns: "Sales Order ID", "Customer", and "Status". The table shows the following data:

Sales Order ID	Customer	Status
700000002	Talpa (100000003)	In Progress (I) >
700000004	Brazil Technologies (100000028)	Delivered (D) >
700000005 Draft	DelBont Industries (100000002)	New (N) >
700000001 Draft	HEPA Tec (100000013)	New (N) >

At the bottom of the table, there are buttons for "Delete", a plus sign for "Create", and a gear icon for "Settings".

At the bottom of the screenshot, there is a caption: "Visualization of draft in SAP Fiori UI – sales order app example".

Related Information

[Developing New Transactional Apps with Draft Capabilities \[page 105\]](#)

[Implementing Draft-Enabled Associations \[page 288\]](#)

4.2 Draft Consistency

Here we explain how the ABAP application infrastructure is used in transactional applications to ensure the data consistency for draft instances of business objects.

By reading this topic, you will understand:

- How the ABAP application infrastructure prevents a draft instance with inconsistent data from being activated

- How you as a developer can determine the consistency status and use it in your scenario for the benefit of the user.

Prerequisites

Your application scenario uses drafts. This includes draft BOPF business objects that are generated in accordance with the CDS-based data model.

Addressed Requirements

In transactional application scenarios that implement drafts, the user can, in principle, enter arbitrary data as a draft data. This means that inconsistent data can also be stored in a draft version.

However, the application infrastructure should automatically prevent draft instances of business objects with inconsistent data from being activated. Consistency validation should be able to ensure that only consistent draft data can be permanently stored in the persistence layer as an active instance. In addition, the result of rejected activations could return a list of messages, each with a reference to the problematic part of the draft, to the consumer.

→ Remember

The consistency of drafts is implicitly ensured by the ABAP application infrastructure. In some rare cases, however, you as a developer should be able to check the consistency status for each individual draft instance in your application scenario.

Consistency Status Field

To identify a draft instance that contains inconsistent data, a suitable marking is required. For this reason, each ROOT node of a draft business object has a technical field that is named `DRAFTENTITYCONSISTENCYSTATUS`. This field represents the consistency status of the entire draft instance. Activation of a draft instance is only possible if the status is consistent.

The field `DRAFTENTITYCONSISTENCYSTATUS` can have three different values that are defined in `/BOBF/IF_FRW_C`:

- PENDING (if the status is undetermined),
- CONSISTENT or
- INCONSISTENT

The screenshot shows the SAP Fiori Properties view for the entity type ZDEMO_I_SALESORDER_TP. The top navigation bar shows the path: ZDEMO_I_SALESORDER_TP > ZDEMO_I_SALESORDER_TP >. Below the path is a section titled "Properties". Under "Properties", there is a tab labeled "DRAFT". A table is displayed with the following columns: "Element Name", "Enabled", "ReadO...", and "Manda...". The table contains three rows, each with a checked "Enabled" box and an unchecked "ReadO..." and "Manda..." box. The rows are: "DRAFTENTITYCREATIONDATETIME", "DRAFTENTITYLASTCHANGEDATETIME", and "DRAFTENTITYCONSISTENCYSTATUS". The last row, "DRAFTENTITYCONSISTENCYSTATUS", is highlighted with a yellow background.

Element Name	Enabled	ReadO...	Manda...
DRAFTENTITYCREATIONDATETIME	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
DRAFTENTITYLASTCHANGEDATETIME	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
DRAFTENTITYCONSISTENCYSTATUS	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Overview Alternative Keys Properties Associations Actions Determinations

The field DRAFTENTITYCONSISTENCYSTATUS is defined for each draft business object

The value of the consistency status field is automatically defined by the ABAP application infrastructure (BOPF) using built-in status handling, namely as follows:

- Status evaluation (inconsistent or consistent) occurs on demand by the PREPARATION action.
- Each change to the draft instance resets the status to *pending*.

As a developer, you have the option of checking the consistency status for each individual draft instance.

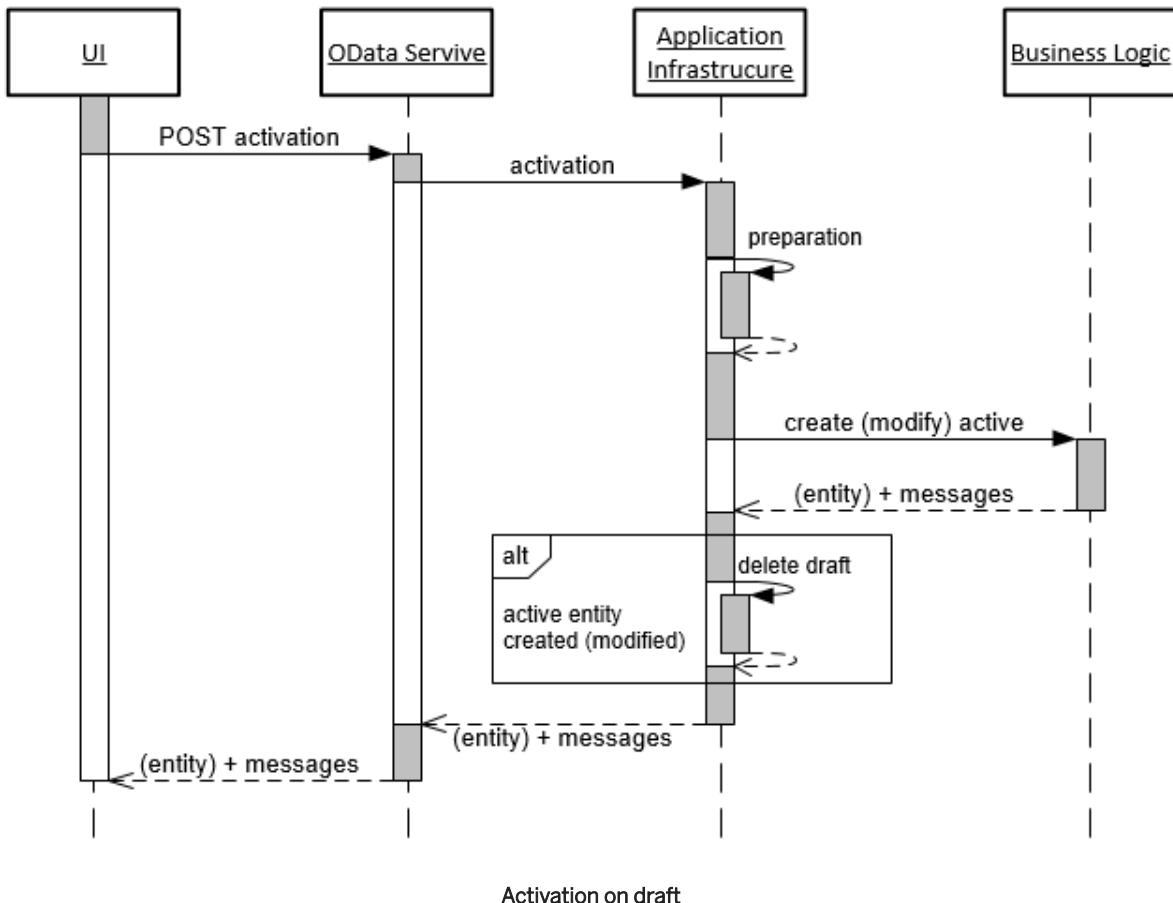
Preventing Inconsistent Draft Entities from Being Activated

The diagram below describes the process of draft activation. As you can see in the life-cycle diagram of the [draft topic \[page 37\]](#), the activation can be applied to the *New Draft* and the *Edit Draft* states.

If the user interface sends an OData POST request for draft activation to the service layer, the request is forwarded to the BOPF framework that is part of the ABAP application infrastructure. This infrastructure evaluates the consistency status of the draft instance. The consistency validation implicitly executes the PREPARATION action and sets the value of the draft consistency field. (The implicit execution of PREPARATION, however, requires that the POST request is part of a single change set and no other changes are included in this change set - besides those that result from the ACTIVATION action.)

If this status is pending or inconsistent, the activation action is rejected. In this case, because of consistency checks or other business logic triggered by it, only a list of messages is returned, each with a reference to the problematic part of the draft.

Otherwise, the draft can be converted into an active instance: either a new active instance is created when activating a *New Draft* or the existing active instance is modified when activation is applied for an *Edit Draft*. Also, the draft is deleted after its data is copied to an active instance.



Related Information

[Draft Concept \[page 35\]](#)

[Validations for CDS-Based Business Objects \[page 58\]](#)

4.3 Locking and Resume

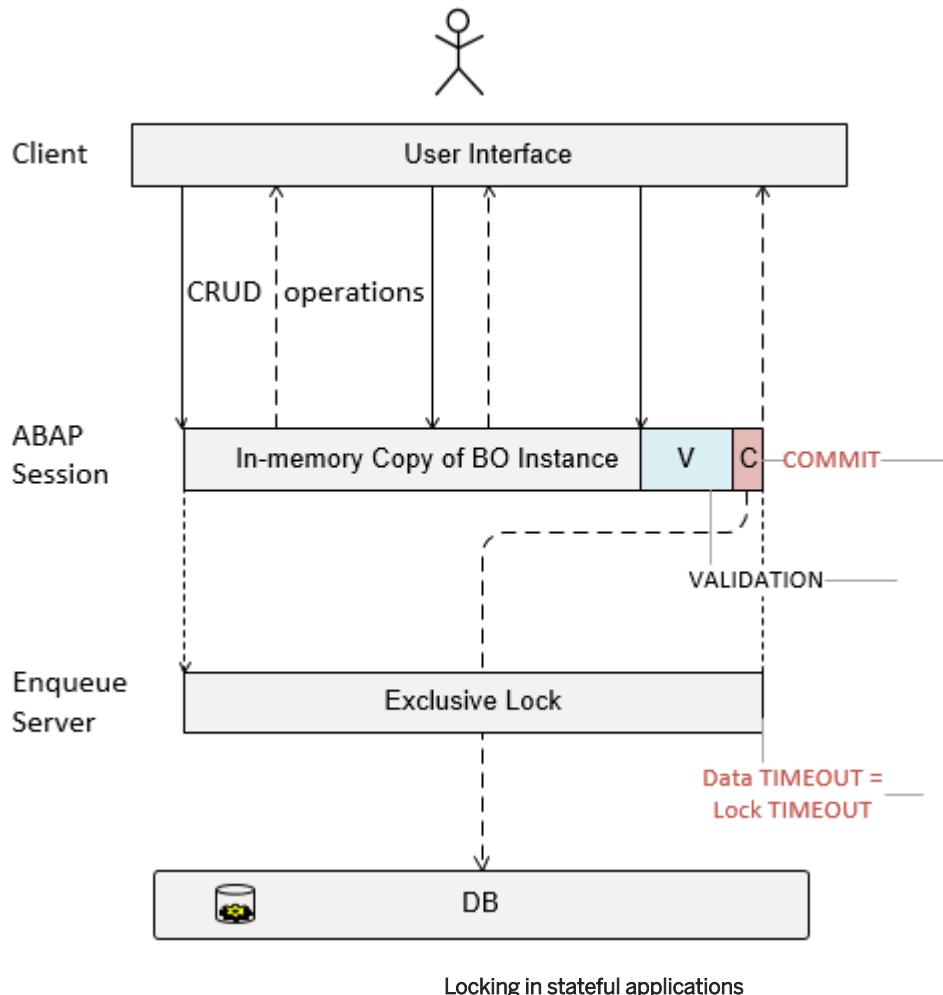
Here, we explain the relationship between the draft and locking behavior in a stateless programming model.

Locks in Traditional Applications

Traditional transactional ABAP applications are developed on a stateful programming model. These applications rely on the server session, along with application buffers that can fulfill client requests until the user has successfully saved or discarded the data changes and finished his/her work.

The stateful programming model supports the processing of in-memory copying of business entities (such as business objects instances) in an incremental way, involving user interactions with multiple round trips

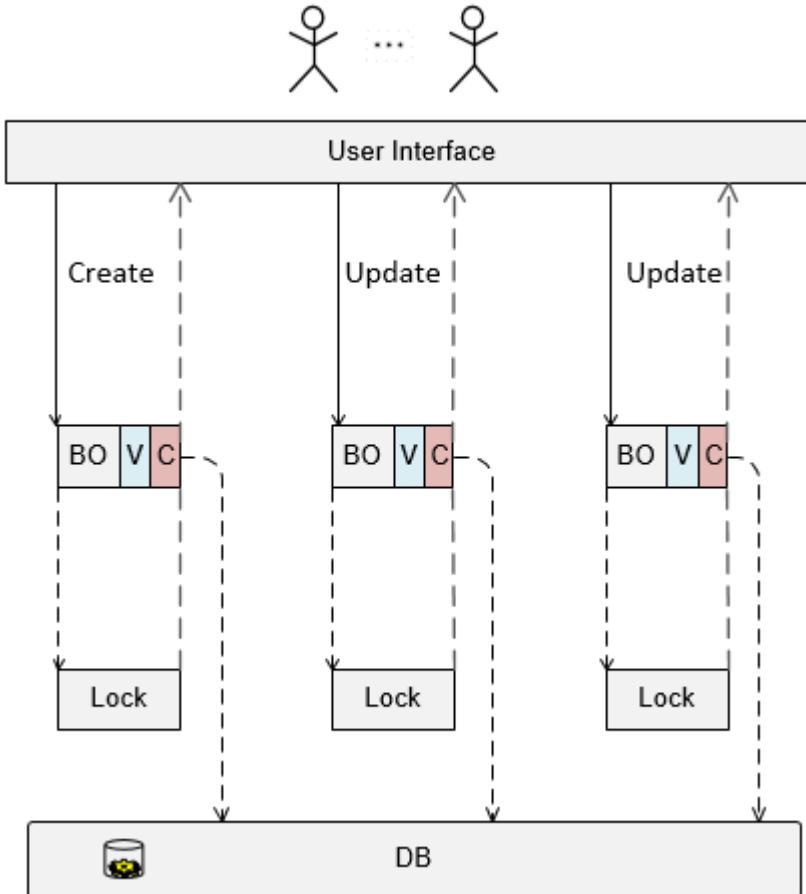
between the UI and the back end. The lifetime of this copy is tied to the session lifecycle, which also controls pessimistic locks. The state is bound to a single ABAP session, which starts with the first request and ends with the commit. The currently set exclusive lock (bound to an individual user) is canceled when the commit is executed. The lifetime of this lock is thus tied to the lifetime of the ABAP session, and the lock expiration coincides with the timeout of the ABAP session.



Locks in Stateless Applications

Newer business applications follow a stateless programming paradigm, which offers better scalability by avoiding a difficult behavior of application states on the application server. The application scenario might require the data from business entities to be held across multiple interaction steps that can be triggered by multiple users.

Since the communication in this case is based on a RESTful protocol, there is no ABAP session continuously available. In a stateless approach without a continuous session, a persistent counterpart needs to replace the in-memory copy of the business entity data to keep the state across requests. Moreover, locks are decoupled from the session and bound to the lifecycle of the draft – as depicted in the figure below.



CRUD operations and locking in stateless applications – schematic diagram

This approach has certain disadvantages however:

- Each atomic writing operation (create, update, update, and so on) executed during the interaction phase will result in a separate commit, preceded by corresponding validations. Editing a single entity (business object instance) therefore typically involves multiple validations.
- Missing isolation - Since multiple users can collaborate on the same business entity, this can result in concurrent execution of operations that affect the state of the entity. In other words, a single user can create intermediate states on entities that do not appear for other users.
- Each writing operation must result in a consistent state, since validation might reject writing. It is therefore not possible to save an intermediate state during the creation of a new business entity.

Solution: Locking Based on Draft

For some specific applications, it would suffice to shift the application logic to the UI layer. With this approach however, the development infrastructure would have to ensure that all required data from the back end is replicated to the front end after the incoming request. In this case, the state of the data would be kept at the front end. Only the end states would be passed on to the back end.

The effort involved for data transfer in typical business applications can become quite high however. Calculations that take place at the back end could sometimes transfer large result sets to the UI. In addition,

the clear majority of the business logic exists as a legacy code. Rebuilding this on the UI is not easy, and includes enormous effort when using JavaScript for example. Finally, this kind of UI session concept would make it necessary to pay greater attention to security issues.

Draft Entities and Durable Locks

To tackle the requirements of locking in stateless applications, the concept of draft entities and an associated infrastructure is implemented in order to persist intermediate states of a business entity that might contain incomplete or even erroneous data.

The data from business entities that is to be held over multiple interaction steps is therefore kept in a draft version. This draft is a temporary collection of changes made to the data. At the user's activation request, the changes can be applied as active data, and the draft deleted. With this draft concept, all data changes are persisted in draft tables, which are shadow tables of the original data, to ensure that the draft data is not accidentally processed by any kind of reporting for example. [More on this: Draft Concept \[page 35\]](#)

In addition, regular enqueue locks are extended to allow decoupling locks from ABAP session lifecycle: **durable locks**. In this context, the term durable expresses the capability to preserve an enqueue lock even after ending the session or when committing data in an intermediate step to persist the state of a draft entity. Durable locks can therefore be kept across multiple OData requests of a client, each processed in its own session.

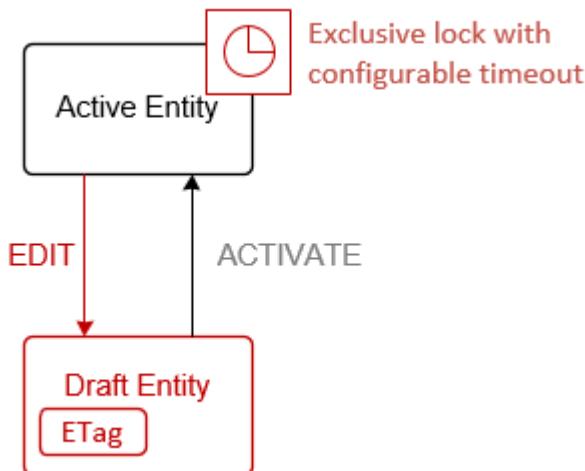
This approach has many benefits: Since the state is continuously transferred to the back end, the business logic can be invoked to provide the end user with feedback related to the data entries. Furthermore, additional UX qualities such as continuous work, device switch and data loss preventions can be achieved, since the persisted state in the draft tables is decoupled from the ABAP session, from the end user, and even from the device.

As shown in the figure below, the draft acts as a locking instance in its own right:

New user inputs or data updates (subsumed in the EDIT action) for an existing business entity start with the creation of a draft entity as a copy of active entity. The EDIT action creates an exclusive lock on the active entity so that no further changes can be made to the active data. Since the draft lifecycle does not have a short timeout behavior (otherwise we would not achieve the draft qualities), the lock needs to be removed to ensure a proper workflow. To enable collaborative work on the business entity, the expiration period of this exclusive lock is therefore limited. It starts with the first modification of the draft entity and ends with a **lock timeout** (which is configurable).

i Note

In comparison to the ABAP session with a single timeout, the draft makes it possible to separate the data timeout from the lock timeout.



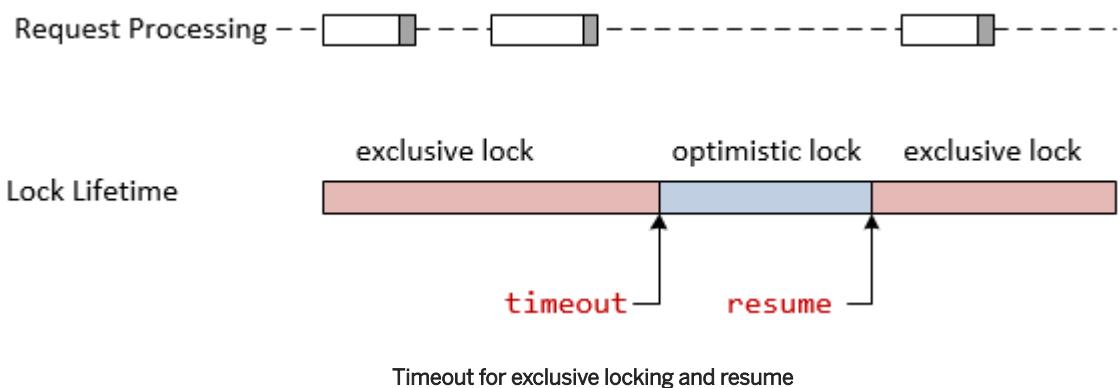
Relationship between the life cycle of draft and locking in stateless programming

Locking Timeout and Resume

Once the timeout of the exclusive lock has occurred (if the user has been inactive for a substantial period of time for example), the durable lock is removed, and the draft is operated in an optimistic lock mode.

The corresponding **ETag** value of the active entity (time stamp of the last data modification, for example) is included in additional administrative data of the draft entity. It is used to determine whether two representations of a business entity, such as an active instance and the corresponding draft BO instance, are the same. If the representation of the entity ever changes, a new and different ETag value is assigned. An ETag check is used to determine whether an optimistically locked draft still matches the current version of the related active instance.

Once the user continues working on draft data, the draft state must be resumed. As depicted in the figure below, **resume** re-acquires the exclusive locks.



There is not guarantee however that resume will be successful, since changes might have been applied in the meanwhile (as could always be the case in an optimistic lock phase). Another user (U2) might have changed the related entity for example. In this case, the first user U1 loses his/her draft D1, and a newer version D2 will be created. If user U1 tries to access D1, s/he gets the response: *resource not found*. (If on the other hand user U2 does not change the related entity, the original draft D1 of the entity is retained and can be further processed after resumption.) Optimistic mode avoids a lock conflict in any case.

→ Remember

Before resuming a draft, the ETag of the active instance is checked. If the ETag has been changed, this results from changes to the active instance during the optimistic lock phase. The old draft then cannot be resumed, and a new draft from the active instance must be created.

Durable Locks and Enqueue Context

Durable locks are required to protect an active version of a business entity - which is edited as a draft in a stateless manner - against changes from legacy applications (for example, SAP GUI-based components).

But how do they work?

A durable lock is requested using an enqueue context. This context serves as the unique identifier of a lock phase during which cross-session locks are assigned to this context. A draft is associated with its enqueue context through an ID. If, a legacy application uses function modules to acquire its own locks on certain resources for example, the locks from the legacy code must be made durable for as long as the draft exists. The enqueue context therefore integrates locks from the legacy code into the lock phase of the durable locks.

Note that the enqueue context, and all locks assigned to it, will be removed if the corresponding draft is not edited for a certain period of time (timeout). Once the user continues working on the draft, the draft state must be resumed.

For editing a specific draft, this means that the `resume()` method of the framework interface `/BOBF/IF_FRW_DRAFT` is called whenever the lock on the active entity is reacquired after being lost because of a timeout of the assigned enqueue context (or for other technical reasons).

→ Remember

Resumptions of locks on active instances is performed by the framework. To this, it executes the LOCK action and runs the ETag check. Certain application scenarios can require additional locks however. Only then is implementation of the `resume` method necessary. Implementation of the `resume` method is therefore optional and is only necessary if, in addition to the lock for the active instance, other locks have to be added in the same context for durable locks (as described above). This can therefore be considered an exception.

To connect to/disconnect from a draft's enqueue context before/after acquiring enqueue locks, the methods `attach()` and `detach()` of framework class `/BOBF/CL_LIB_ENQUEUE_CONTEXT` are used – as demonstrated in the implemented example below:

Code Sample

The following code sample demonstrates how you can implement the `resume` method (from `/BOBF/IF_FRW_DRAFT`) that is used to keep the application-specific data of a draft instance in case the enqueue context has been lost.

Using this code sample, you can understand the basic implementation steps of handling enqueue context for drafts in stateless programming model.

a). Requesting a Durable Enqueue Context

The following call returns an instance of the enqueue context for durable locks:

```
DATA(enqueue_context) = /bobf/cl_lib_enqueue_context=>get_instance( ).
```

b). Activating the Specified Enqueue Context

The enqueue context is used to call the `attach(iv_draft_key)` method. This method call activates the enqueue context in the current ABAP session. As a result, all subsequent lock operations will use this enqueue context of the draft until `detach()` is called. Note that the importing parameter `iv_draft_key` should only be the root instance keys of the current draft instance. This means in particular that it is not possible to activate the context of a foreign business object from outside.

c). Integrating Locks from Legacy Code

The specified enqueue context is also used to make locks from the legacy code durable.

With the call `locks_from_legacy_code->lock_person()`, the legacy application uses function modules to acquire its own locks on resources (in our example, on the person object).

d). Deactivating the Previously Activated Enqueue Context

The application calls must detach after an attach. Otherwise it will be followed by a dump. The `detach()` method deactivates the currently active enqueue context and restores the original enqueue context of the current ABAP session. All subsequent lock operations will then use the original enqueue context of the ABAP session. Unlike the `attach()` method, there is no draft key required for this method as an importing parameter.

Listing: Implementation of durable locks in the resume method

```
METHOD /bobf/if_frw_draft~resume.

...
*** (a) Create a durable enqueue context
DATA(enqueue_context) = /bobf/cl_lib_enqueue_context->get_instance( ).
TRY.
  LOOP AT lt_person ASSIGNING FIELD-SYMBOL(<s_person>)
    WHERE personuuid IS NOT INITIAL.
    *** (b) Attach the specified enqueue context
    enqueue_context->attach( <s_person>-root_key ).
    TRY.
      *** (c) Activate locks in legacy code
      locks_from_legacy_code->lock_person( <s_person>-personuuid ).
      CATCH cm_my_message INTO DATA(lm_exception).
        lm_exception->set_origin_location( is_location = VALUE #( 
          node_key      = is_ctx-node_key
          key           = <s_person>-key
          attributes    = lt_attributes
        ) ). 
        eo_message->add_cm( lm_exception ). 
        INSERT VALUE #(`key = <s_person>-root_key )
        INTO TABLE et_failed_key.
    ENDTRY.
    *** (d) Deactivate the previously acquired enqueue context
    enqueue_context->detach( ). 
  ENDLOOP.
  CATCH /bobf/cx_frw_core.
    et_failed_key = it_key.
  ENDTRY.
...
ENDMETHOD.
```

Related Information

SAP Lock Concept

ETag: Object Model Annotations > ObjectModel.entityChangeStatId [page 428]

Object Model Annotations > ObjectModel.lifecycle.enqueue.expiryInterval [page 428]

Object Model Annotations > ObjectModel.lifecycle.enqueue.expiryBehavior [page 428]

4.4 Messages

This topic describes the message concept for stateless applications that is part of the ABAP application infrastructure.

Messages and Failed Keys

There are a lot of situations in the life cycle of an application in which messages play a crucial role. For example, incorrect input from the end user detected in validation as faulty input or as invalid data entry, is one reason to send corresponding feedback to the user so that a correction can be made.

Messages can be created during the execution of the application logic, such as in an action, a validation, or a determination. They arise during the execution of these operations to inform the end user about success, problems, or even failure.

At the technical level, messages are returned to the consumer using message containers. The message container is returned by the application logic by assigning the reference to the exporting parameter `EO_MESSAGE`.

Signature of /BOBF/IF_FRW_VALIDATION->EXECUTE

importing	...		
exporting	<code>EO_MESSAGE</code>	<code>type ref to</code>	/BOBF/ IF_FRW_MESSAGE
	<code>ET_FAILED_KEY</code>	<code>type</code>	/BOBF/T_FRW_KEY
raising	...		

The validation may return state or transition messages (see also section *Message Types* below) to inform the consumer about any issues. The behavior, however, depends on the state of the business object's instance:

Draft Instance

For draft instances, it is possible to return both STATE and TRANSITION messages.

Active Instance

In draft-enabled applications, it is not possible to create messages with lifetime STATE for active instances. In this case, a short dump occurs.

Messages returned by a validation are not able to prevent active instances from being saved nor can they reject activations of draft instances. This is only possible using failed keys. An instance is considered invalid when at least one failed key is raised by at least one of the executed consistency validations, that is, when the exporting parameter `et_failed_key` of the `EXECUTE` method (of a validation, an action, or a determination) contains the key of the instance.

If the issue detected by the validation requires an action from the consumer to make the instance consistent, a save of the active instance (or an activation of the draft instance) can be rejected.

Draft Instance

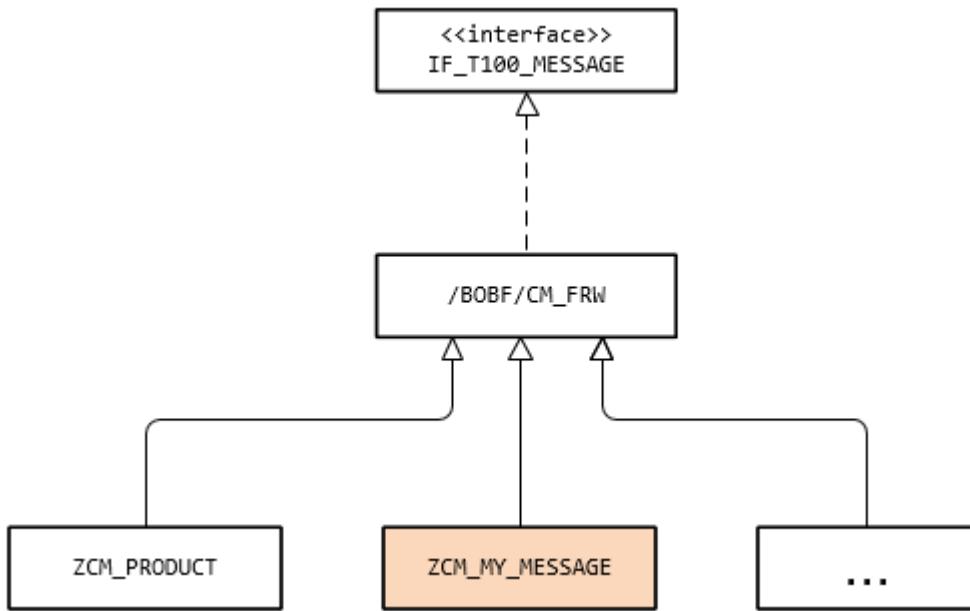
If `et_failed_key` contains keys of draft instances, activation of the draft is prevented using the draft consistency status. **More on this:** [Draft Consistency \[page 40\]](#)

Active Instance

If `et_failed_key` contains keys of active instances, changes cannot be committed to the database.

Class-Based Message Framework

The message concept of the ABAP application infrastructure is based on ABAP class-based messages (CM classes). That means that each individual message is an instance of an ABAP exception class. As depicted in the figure below, message-related classes inherit from `/BOBF/CM_FRW` as the superclass.



Class-based messages inherit from the framework class `/BOBF/CM_FRW`

The super class `/BOBF/CM_FRW` already provides all basic features of the message concept of the ABAP application infrastructure. These features can be used to implement application-specific message-related exceptions.

→ Remember

To keep an overview, it is recommended that you create your own message-related exception class for each set of messages that are related to each other. Each exception class can include multiple messages.

Activities Relevant to the Developer

1. Create an ABAP exception class with the superclass `/BOBF/CM_FRW`.
More on this:
2. Use the code template `textIdExceptionClass` to add the constant for the exception ID.
More on this:
3. Create a message class and add the appropriate messages.
More on this:

C ZCM_MY_MESSAGE		ZCM_MY_MESSAGE X		
Message Class : ZCM_MY_MESSAGE [Total Messages: 1, With Long text: 0]				
	New	Delete	Long Text	Preview long text
type filter text				
Locked	Message Number	Short Text	Self Explanatory	
<input type="checkbox"/>	001	Start year &1 must be before end year &2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	<Enter new value>			

Message class ZCM_MY_MESSAGE (in the message editor of ADT)

- In the *Properties* view of the exception class, assign the related message class (as depicted in figure below):

The screenshot shows two views of the SAP ABAP IDE. The top view is the ABAP editor for the class `ZCM_MY_MESSAGE`. It contains the following code:

```

1 class zcm_my_message definition
2   public
3     inheriting from /bobf/cm_frw
4     final
5     create public .
6
7   public section. | Exception ID
8
9   constants:
10    begin of INVALID_START_END_YEAR,
11      msgid type symsgid value 'ZM_MY_MESSAGE' Message Class
12      msgno type symsgno value '001',
13      attr1 type scx_attrname value 'START_YEAR',
14      attr2 type scx_attrname value 'END_YEAR',
15      attr3 type scx_attrname value '',
16      attr4 type scx_attrname value '',
17    end of INVALID_START_END_YEAR .

```

The bottom view is the properties view for the same class. The `Message Class:` field is highlighted in yellow and contains the value `ZCM_MY_MESSAGE`.

Application-specific exception class `ZCM_MY_MESSAGE` refers to the same-titled message text class - ADT editor and Properties view

Attributes of Messages

Whenever a message is issued, a new object of the message-related exception class is created. When a message is created, for example in a validation's `EXECUTE` method, the attributes of this object must be specified.

The following attributes are required to initialize a message object:

Attribute	Meaning
TEXTID	Key for identifying the message text
SEVERITY	<p>Severity of the issue</p> <p>Severity values are defined as constants in /BOBF/CM_FRW; they are used to indicate the message as ERROR, WARNING, SUCCESS, or INFORMATION.</p>
SYMPTOM	<p>Optional symptom values</p> <p>Symptom values are used to automatically determine how to deal with the message; they are defined as constants in /BOBF/IF_FRW_MESSAGE_SYMPTOMS.</p>
LIFETIME	<p>Specifies the message type, of either a transition message or a state message</p> <p>Messages with lifetime value TRANSITION are just returned to the consumer; they do not appear again in subsequent round-trips (<i>fire and forget</i> semantics).</p> <p>Messages with lifetime value STATE (valid for draft instances only) are returned to the consumer and also persisted on the database (durable messages). They are available in subsequent round-trips (including read operations) without re-executing the business logic that created the message until the message is invalidated.</p>
MS_ORIGIN_LOCATION	<p>Indicates which field is responsible for triggering the message</p> <p>This attribute points out the business object, node, instance key, and attribute referred by the message.</p>
<p>i Note</p> <p>This specifies that the referred attribute has a red border on a Fiori UI.</p>	
(MV_CURRENCY_CODE)	Application-specific property (as example)

i Note

If necessary, additional attributes can be created, for example to pass specific values for the messages.

Message Types

Depending on the value of the `lifetime` parameter of validations, there are two different message types available:

- **Transition messages** – Events that are related to the consumer's request or a temporary system state and hence do not need to be persisted
- Transition messages remain in the message container and are passed to the consumer by the standard BOPF message channel. If the consumer repeats the same request later on, it might happen that the issue will not occur any more. This kind of message follows the *fire and forget* semantics.
- A common way to visualize transition messages is a *message toast* that appears only briefly on the user interface. **More on this:** [Message Toast](#)

Examples:

- A requested resource is locked by another user and is therefore temporarily not available.
- The correct semantics of the importing parameter of an action are not specified by the user.

- **State messages** (relevant for draft only) – Events that are related to the state of a node instance

This state can only be changed by a modification of the instance changing its state. State messages are therefore persisted and need to be deleted if the message is not valid anymore. Hence, state messages are durable events. Usually, the user interface shows the [message box](#) until the issue is fixed.

Examples:

- Wrong business partner ID
- The currency code entered by the user is not allowed

Invalidation of Messages

State messages are automatically stored in the backend until the activity that created the message for a certain instance is called again without this message being created again. Therefore, state messages need to be deleted if they are not valid anymore.

A persisted message can be deleted if...

- The corresponding business object instance is deleted, or
- The reason justifying the message creation has expired.

From the runtime perspective, the second point becomes true if the check logic that raised the persisted message earlier is re-executed.

Example

A message regarding an invalid business partner ID becomes invalid as soon the corresponding validation CHECK_VALID_ID is re-executed. The validation either creates the same message again, if the ID is still invalid or the check is successful and the message is not raised again.

Code Sample

The following code sample demonstrates how both messages and failed keys originate from the application logic and how they are forwarded to the application framework. The starting point is the implementation of a consistency validation that is used in a sales order application to check whether the year entries (`start_year` and `end_year`) specified by the end-user are valid or not.

Using this code sample, you can already see the basic process of handling messages during execution of business process operations:

a).Creating a Message Container

The message container `eo_message` is created using the factory method:

```
eo_message = /bobf/cl_frw_message_factory->create_container( ).
```

This container object is used to collect the messages that originate from the implementation of an action, determination, or validation (as in the listing below).

b). Retrieving the Relevant Data

When checking the validity of data, we must first retrieve data of the sales order instance that was changed by the user. The standard method call `io_read->retrieve()` is used to retrieve the data of the root node. This

method call imports the data of the requested node instances and exports the keys of failed node instances (set of node instance keys that do not match the validation criteria) in the parameter `et_failed_key`.

c). Checking Validity of Data

The actual check whether the year entered by the end user is valid takes place in an `IF` statement. If the data entry for a particular node instance is incorrect or invalid, then its technical key is added to the key table `et_failed_key`.

d). Adding New Messages to the Message Container

The new message container `eo_message` provides access to all (class-based) messages that are issued during a roundtrip. Each individual message that is created as an instance of the message class `cm_my_message` is finally added to the message container in the call `eo_message->add_cm()`.

Listing: Retrieving Data from Draft Instance

```
METHOD /bobf/if_frw_validation~execute.  
...  
*** Constants interface of the generated business object: if_my_bo_c  
DATA(lt_attributes) = VALUE stringtab( ( if_my_bo_c=>sc_node_attribute-root-  
start_year )  
                                         ( if_my_bo_c=>sc_node_attribute-root-  
end_year ) ).  
*** (a) Create a message container  
eo_message = /bobf/cl_frw_message_factory=>create_container( ).  
*** (b) Retrieve relevant data and return failed keys  
io_read->retrieve(  
    EXPORTING  
        iv_node           = is_ctx-node_key  
        it_key            = it_key  
        it_requested_attributes = lt_attributes  
    IMPORTING  
        et_data           = <t_data>  
        et_failed_key     = et_failed_key  
    ).  
...  
LOOP AT lt_data ASSIGNING FIELD-SYMBOL(<s_data>).  
*** (c) Check whether the entered data is valid  
    IF <s_data>-end_year IS NOT INITIAL AND <s_data>-start_year > <s_data>-  
end_year.  
        INSERT VALUE #( key = <s_data>-key ) INTO TABLE et_failed_key.  
*** (d) Add new messages to the message container  
eo_message->add_cm( NEW cm_my_message(  
    textid           = cm_my_message=>invalid_start_end_year  
    start_year       = <s_data>-start_year  
    end_year         = <s_data>-end_year  
    severity         = cm_my_message=>co_severity_error  
    lifetime         = cm_my_message=>co_lifetime_state  
    ms_origin_location = VALUE #(  
        bo_key          = is_ctx-bo_key  
        node_key        = is_ctx-node_key  
        key             = <s_data>-key  
        attributes      = lt_attributes  
    )  
    ) ).  
ENDIF.  
ENDLOOP.  
ENDMETHOD.
```

4.5 Validations for CDS-Based Business Objects

A validation is an entity of a business object node that is triggered in certain situations to check various aspects of a given set of node instances. It is used to validate whether a set of node instances is consistent.

→ Remember

Validations never modify any node instance data but they do return the messages and keys of failed (inconsistent) node instances.

How Do Validations Generally Work for Active and Draft BO Instances?

Validations check whether an active or draft node instance of business object is consistent with respect to the consistency criteria imposed by the business requirements. Such validations are called at predefined points within the BO transaction cycle to ensure that BO nodes are persisted in a consistent state. These points can be configured for validations for both active and draft business objects. Each such validation configuration contains a trigger condition that is checked by BOPF runtime at several times during the transaction. If the trigger condition is fulfilled, the consistency validation is executed. Otherwise, if there are inconsistent node instances, a consistency validation behaves in one of the following ways:

- The validation sends messages to the consumer – for both active and draft instances
- The validation sends messages to the consumer and prevents the transaction from being saved until the inconsistency is corrected - for active instances
- The validation sends messages to the consumer and changes the draft consistency status – for draft instances.

Example

The *CUSTOMER_INVOICE* business object may contain multiple validations to check whether the items of an invoice are in a consistent state. For example, one consistency validation checks whether the quantity and the price of each item are defined. Another validation checks whether the address of the buyer is valid. If it is not, the consistency validation prevents the transaction from being saved to ensure that only consistent invoice instances can be saved.

Implicit Validations

As a developer, you have the option of configuring the trigger condition for validation. If this condition is fulfilled, the BOPF runtime automatically executes the validation. We call this implicit validation.

Implicit Validation - After Modifications

Implicit validations are executed after each modification of node data, depending on the configured request nodes.

This kind of implicit validation is based on the configuration of the trigger condition that is defined for the requested business object's node. The trigger condition describes the node (called *request node*) and the kind

of change (create/update/delete) that should automatically trigger the validation execution by the BOPF runtime.

For each request node, independently, the triggers are configured to decide on which kind of change the validation will be automatically executed. For validations, these change operations are:

- *Create*: validation after a node instance is created
- *Update*: validation after a node instance is updated
- *Delete*: validation after a node instance is deleted

→ Remember

Note that validation trigger on *Delete* only makes sense for sub nodes (with respect to the node where the validation is defined). A deleted instance of a node does not exist anymore at this point in time and cannot be validated. Therefore, the trigger on *Delete* for the same node or the parent node will not execute a validation. This fact is taken into account when you check the BO by issuing a corresponding warning message.

Triggers

Information on Triggers

Node	Association	Create	Update	Delete
I_AIVS_MDBU_ARTISTTP		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Defining trigger condition for validation in the BO editor

→ Remember

At least one of these operations must be flagged for each request node. *Create* and *Update* are flagged by default.

Example: If an instance of node I_AIVS_MDBU_ARTISTTP from the figure above is created or updated, the validation is automatically invoked by the BOPF infrastructure.

Draft Instances

The execution of the validation on draft instances is done immediately after the modification.

Active Instances

In the case of a stateless session, the execution of the validation on active instances is delayed when the transaction is saved (instead of being executed immediately after modification).

Implicit Validation – While Activation (for Draft Instances) and Save (for Active Instances)

Draft Instances	Active Instances
In the current release, the BOPF infrastructure implicitly triggers PREPARATION in cases where the ACTIVATION action is called by the consumer.	In the case of a stateless session, the execution of the validation on active instances is delayed from immediately after modification to when the transaction is saved (instead of being executed immediately after modification). This is because the modification and the save happen in the one and only roundtrip in that session.

Explicit Consumer Request for Validation Execution

In addition to implicit consistency validations, a consumer (for example, the UI) may explicitly run validations (explicit validation).

Draft Instances	Active Instances
To explicitly trigger the validation execution, the consumer can invoke the PREPARATION action for a draft instance at any time during a transaction cycle.	To explicitly trigger the validation execution, the consumer can invoke the CHECK_CONSISTENCY, CHECK_AND_DETERMINE on an active instance at any time during a transaction cycle.

Related Information

[Extending the Implementation with Value Validation \[page 148\]](#)

4.6 Actions for CDS-Based BOPF Business Objects

An action is an entity assigned to an individual node of a business object that is used to implement a service (operation or behavior) of the business object.

Examples

You can use the RELEASE action to change the status of an invoice instance of the CUSTOMER_INVOICE business object. As a result of this action execution, the status of the invoice can be set to RELEASED. The following are other examples of an action:

- **COPY SHEET**
- **PUBLISH SALES_QUOTE**
- **SET TO PAID SALES_ORDER**

Triggers for Actions

For an action to be executed, the corresponding triggers are required. Actions are called actively by one of the following:

- A service consumer, for example by a user interface
- Another business object
- Internally, for example by another action or by a determination.

Configuring Actions with Instance Multiplicity

The user can perform an action on one or more node instances, or on no instance (static action) at all. For this purpose, you can assign the instance multiplicity to an action. The action's multiplicity defines how many node instances the action can operate on during one action call. You can choose the following cardinality types for actions:

- Multiple node instances
Action operates on one or more node instances.
This option enables the end user to select multiple instances (for example, multiple sales order items) of a business object and trigger an action. The action is then executed sequentially for the selected items.
- Single node instance
Action operates on exactly one single node instance.
When using this option, we do not want to allow the end user to execute the action on a set of multiple node instances.
- No node instances (static action)
The action does not operate on any node instances and defines a static action. For example, creation procedures are defined as static actions.

Configuration Actions with Exporting Type

In addition, you can configure each action with an exporting type. Using the exporting type parameter, you can specify whether the action is able to return data to the caller during the action execution and if so what kind of data.

There are three options available:

- **None** - This default option indicates that the action does not return any data to the consumer, even if a related action can cause changes to the business object.
- **Type** - This option allows the action to return a data table defined by the row type entered in the field Exporting Parameter Structure and the corresponding table type entered in the field Exporting Parameter Table Type.
- **Node** - This option allows the action to return instances of a (foreign) node that is defined by the field Exporting Parameter Node and Exporting Parameter BO.

Action Overview

General Information

Name: * SET_PAID

Description: set to paid action

Instance Multiplicity: Multiple Node Instances

Implementation Details

Implementation Class: * ZCL_DEMO_A_SET_PAID

Parameter Structure:

Exporting Type: Node

Exporting Parameter BO: ZDEMO_I_SALESORDER_TP

Exporting Parameter Node: ZDEMO_I_SALESORDER_TP

Exporting Multiplicity: 1

Overview

Action configuration - Specifying exporting type

Implementing a Custom Action

When implementing your own custom actions, you must first add a new action to the corresponding BO node and implement the action logic. The action logic is encapsulated within a global ABAP class that implements the corresponding action interface /BOBF/IF_FRW_ACTION.

When calling an action, the consumer must provide the following specifications:

- The **key of node instances** on which the action operates. (This specification only applies when the action operates on one or more instances.) An action can only be performed with exactly the number of instances that is configured in the multiplicity of the action (see section above).

i Note

To prevent an action from operating on certain specific instances, you can define an action validation.

- Any **input parameters** that the action requires.

i Note

In general, an action can have multiple importing parameters, such as multiple structure components (but always of one structure!), that the consumer passes to the implementation class at runtime. All parameters are included in a structure known as a parameter structure.

Related Information

[Extending the Implementation with an Action \[page 152\]](#)

[Enabling Actions for OData Consumption \[page 101\]](#)

[Dynamic Action Control \[page 201\]](#)

4.7 Determinations for CDS-Based Business Objects

In the ABAP application infrastructure, a determination is an entity of a business object node that is used to provide functions that are automatically executed as soon as a certain **trigger condition** is fulfilled. A determination is triggered internally on the basis of changes made to the node instance of a business object. The trigger conditions are checked by the ABAP application infrastructure at different points during the transaction cycle, depending on the determination times and the changing operations on the relevant node instances. For each determination, it is necessary to specify both the points in time and the changes that form the trigger condition. Changes can include creating, updating, deleting, or loading node instances. As a developer, you can implement your own custom determination primarily to compute data that is derived from the values of other attributes. The determined attributes and the determining attributes of the trigger condition either belong to the same node or to different nodes.

Example

If there is a change to the quantity of multiple products (items) in an invoice object instance, the amount attribute (the overall price) for all relevant products has to be calculated again. To calculate this amount automatically, you have to extend the standard change process for the invoice business object instances using a custom determination. The determination can then react to the creation, modification, deletion, or loading of an invoice item by deriving new values for the amount field. For such an example, the solution might look as follows:

After the quantity of invoice items is changed, the trigger conditions of the `CALCULATE_ITEM_AMOUNT` determination and the `CALCULATE_TOTAL_AMOUNT` determination are both fulfilled. The `CALCULATE_ITEM_AMOUNT` determination calculates the amount of the changed item (price x quantity), whereas the `CALCULATE_TOTAL_AMOUNT` determination sums up the amounts of all items to the total amount of the invoice.

Determination Time and Triggers

Depending on the use case in question, the ABAP application infrastructure checks the triggers of a determination at several points during the transaction cycle.

A determination time defines at what time in the transaction cycle the trigger condition of that determination should be evaluated. For example, the re-calculation of the invoice amount should take place every time after a modification is performed (category: **After Modify**), but only if there are instances of the ITEM node that were updated (trigger: **Update**).

The following list shows which trigger operations are evaluated at which determination times (categories):

The determination time and the triggers defines the trigger condition

Trigger >

Category	Create	Update	Delete	Load	Determine
React after modification	X	X	X		
React on check and determine					X
React during save	X	X	X		

Determination Dependencies

If, in the example above, you do not define any dependencies between the determinations, the system might calculate the total amount before the item amount. Therefore, you must specify the CALCULATE_ITEM_AMOUNT determination as a predecessor of the CALCULATE_TOTAL_AMOUNT determination.

To specify whether a determination is a predecessor or a successor of another determination, you define a determination dependency. If there is more than one determination to be executed, the dependencies of the determinations are respected when the infrastructure checks the trigger conditions of determinations.

Example

The following dependency is defined in such a way that the item amount is calculated before the total amount.

Node: Determination	Dependency
ROOT: CALCULATE_TOTAL_AMOUNT	Determination depends on CALCULATE_ITEM_AMOUNT
ROOT > ITEM: CALCULATE_ITEM_AMOUNT	Determination must be executed before CALCULATE_TOTAL_AMOUNT

ZDEMO_I_SALESORDER_TP_DR ► ZDEMO_I_SALESORDER_TP_DR ► CAL

Dependency

[New](#) [Delete](#)

type filter text

Node	Determination	Dependency
ZDEMO_I_SALESORDERITEM	CALCULATE_ITEM_AMOUNT	Predecessor

Overview Dependency

Defining dependency at the ITEM level in the BO editor

Dependency

[New](#) [Delete](#)

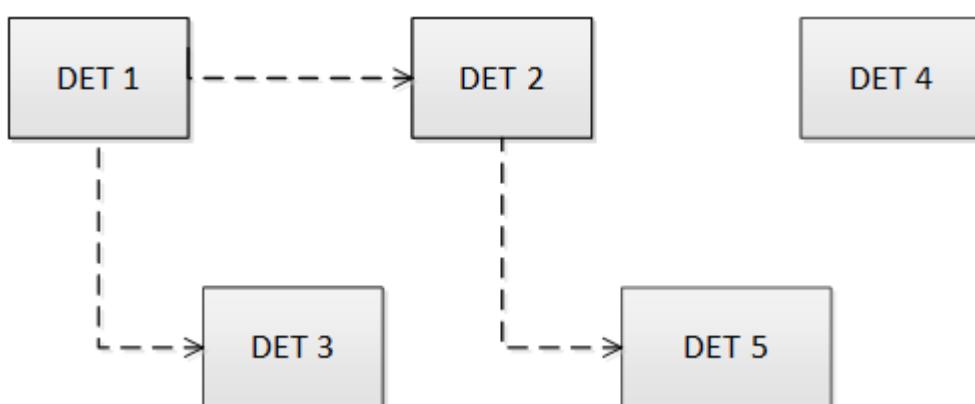
type filter text

Node	Determination	Dependency
ZDEMO_I_SALESORDER	CALCULATE_TOTAL_AMOUNT	Successor

Overview Dependency

Defining dependency at the ROOT level

Dependency defines the execution order of determinations:



Dependencies between determinations

Examples of valid execution orders are:

4, 1, 3, 2, 5 (for example, the determination DET 2 is always executed before the execution of DET 5)

1, 2, 5, 4, 3 (for example, DET 1 is always executed before the execution of DET 2).

Implementing a Custom Determination Class

When implementing your own custom determination, you first must add a new determination to the corresponding BO node and then implement the determination logic.

The determination logic is encapsulated within an ABAP class that implements the determination interface / BOBF/IF_FRW_DETERMINATION.

Related Information

[Extending the Implementation for Creating Sales Order Items \[page 138\]](#)

5 Develop

Contents

[Developing List Reporting Apps with Search and Analytical Capabilities \[page 67\]](#)

[Developing New Transactional Apps \[page 78\]](#)

[Developing New Transactional Apps with Draft Capabilities \[page 105\]](#)

5.1 Developing List Reporting Apps with Search and Analytical Capabilities

Introduction

Starting from the elementary list reporting scenario that was introduced in the Getting Started section, you may want to add some further list reporting functions. For example, if your table or list contains many rows, it becomes difficult for end users to find the information they need. To facilitate finding the desired information, you can provide selection fields (filters) to specify the range of information that the end user is looking for. In addition, you may want to specify the positioning of columns or prioritize, or even hide, specific fields in your table or list, or even enrich the presentation of data with aggregation capabilities.

This is implemented using specific CDS annotations, which you – as a Fiori app developer – add to the source code of the respective CDS view. A CDS annotation (in short: annotation) adds metadata to the view that expands the syntax options of SQL.

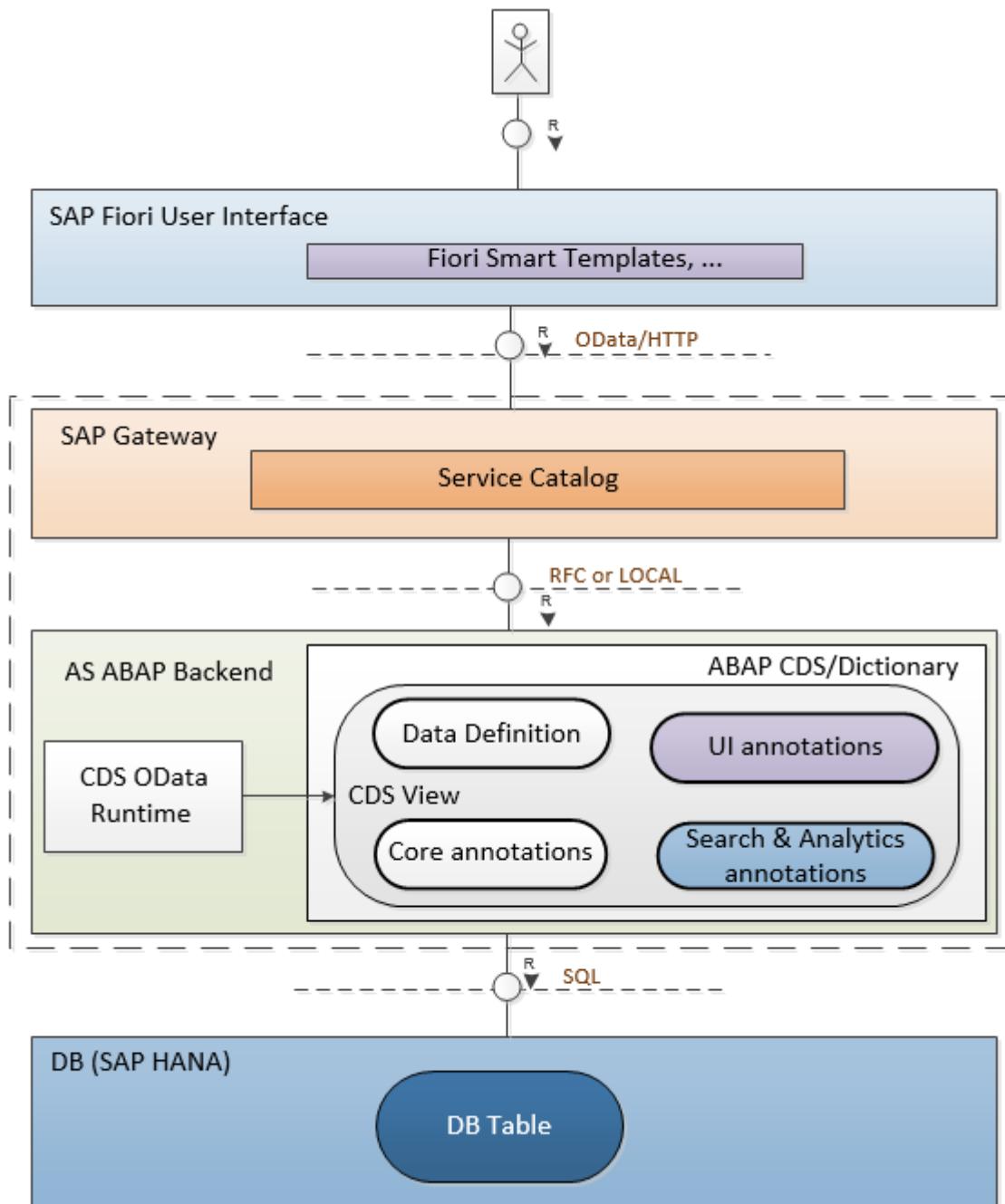
In this development scenario, you will use different types of CDS annotations:

- **Core annotations** - can be specified in the source code of any ABAP CDS object and are evaluated by ABAP runtime environment.
- **UI annotations** - allow you to focus OData UI vocabulary on usage patterns of data in UIs representing certain semantic views of business data. The UI annotations used are evaluated by the SSDL framework, which means that SSDL translates CDS annotations into the corresponding OData annotations.
- **Search annotations** - allow you to define search fields as filters or . The annotations used are also evaluated by the SSDL framework.
- **Analytics annotations** - provide data elements with aggregation behaviour in analytical scenarios.

Video Available

Watch the corresponding video available on our [SCN page](#)!

Architecture Overview



Architecture components in a list reporting scenario that expands the data definition with metadata

Contents

[Data Model Without Metadata - Starting Point \[page 69\]](#)

[Adding Metadata to Data Model \[page 70\]](#)

[Running Resulting App \[page 75\]](#)

Related Information

[Using Aggregate Data in SAP Fiori Apps \[page 263\]](#)

5.1.1 Data Model Without Metadata - Starting Point

Starting Point

Let us assume you have already created a DDL source as a development object and implemented it as simple data model for EPM sales order items - as shown in the listing below.

```
@AbapCatalog.sqlViewName: 'ZDEMO_SOI_ADV'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Advanced List Reporting for Sales Order Item'

@OData.publish: true

define view ZDEMO_CDS_SalesOrderItem_A
as select from SEPM_I_SalesOrderItem_E as Item
{
  key Item.SalesOrder           as SalesOrderID,
  key Item.SalesOrderItem       as ItemPosition,
  Item._SalesOrder._Customer.CompanyName as CompanyName,
  Item.Product                  as Product,
  Item.TransactionCurrency     as CurrencyCode,
  Item.GrossAmountInTransacCurrency as GrossAmount,
  Item.NetAmountInTransactionCurrency as NetAmount,
  Item.TaxAmountInTransactionCurrency as TaxAmount,
  Item.ProductAvailabilityStatus as ProductAvailabilityStatus
}
```

The source code above defines quite a simple CDS view called `ZDEMO_CDS_SalesOrderItem_A`. This view is implemented by means of a query for performing a `SELECT` statement, where the predefined CDS view `SEPM_I_SalesOrderItem_E` (that originates from the EPM demo application) is used as the data source. The select list includes a set of fields that are relevant for the new sales order item consumption view. The `KEY` elements in the selection list are used to define the key field semantic of the CDS view.

When the DDL source is activated, the following objects are created in ABAP Dictionary:

- The actual entity of the CDS view `ZDEMO_CDS_SalesOrderItem_A`
- An SQL view `ZDEMO_SOI_ADV`.

Next: [Adding Metadata to Data Model \[page 70\]](#)

5.1.2 Adding Metadata to Data Model

This topic demonstrates how you can expand the syntax of the CDS view to include metadata with the aim of providing some additional list reporting capabilities.

Adding the List Title

You have the option to work with a title in your result list. For this, add the `@UI.headerInfo` annotation to the CDS view and specify the header information in singular form for a single instance (single sales order item) and in plural form for multiple instances of sales order items.

```
@UI.headerInfo: { typeName: 'Sales Order Item', typeNamePlural: 'Sales Order Items' }
define view ZDEMO_CDS_SalesOrderItem_A as
  ...
```

Preview:



Adding a Standard Search Filter

If a CDS view is annotated as `@Search.searchable`, the UI runtime automatically takes this into consideration, so that no additional UI annotations are required to expose a standard search field. However, for a meaningful search, at least one field must be defined as the default search field by annotating this field as `@defaultSearchElement: true`. The search is then initiated solely effected by those fields that are defined as default search fields. In the listing below, the `ProductCategory` is defined as the default search field.

```
@Search.searchable: true
define view ZDEMO_CDS_SalesOrderItem_A as
  ...
@Search.defaultSearchElement: true
  Item.Product          as Product,
  ...
```

Preview:

The screenshot shows a Fiori application interface. At the top, there is a filter bar with fields for "Sales Order ID" and "Company". The "Sales Order ID" field contains "HT-1060" and has a red oval around it. To the right of the filter bar is a "Hide Filter Bar" link. Below the filter bar is a table titled "Sales Order Items (11)". The table has columns for "Sales Order ID", "Company", "Product ID", and "Gross Amo..". All rows in the table have "HT-1060" in the "Product ID" column, which is also highlighted with a red oval.

Sales Order ID	Company	Product ID	Gross Amo..
500000008	Telecomunicaciones Star	HT-1060	21.42 EUR
500000033	Telecomunicaciones Star	HT-1060	21.42 EUR
500000068	Telecomunicaciones Star	HT-1060	21.42 EUR
500000093	Telecomunicaciones Star	HT-1060	21.42 EUR
500000118	Vente Et Réparation de Ordinateur	HT-1060	21.42 EUR

To facilitate finding the desired data, you can provide the standard filter for the list reporting UI.

Adding a Filter Bar for Field-Specific Selection

You can use the following UI annotations to enable specific elements for selection, for example, when providing a filter bar:

The first filter allows the end user to search for *Sales Order IDs*. As the second filter, the customer's *Company* search field is defined. In addition, the *Company* field should be considered as the default search field in the standard search filter.

i Note

If an element is annotated as `SelectionField` and has a value-help, the value-help is exposed automatically to the UI. This is independent from the `@Search.searchable` annotation.

```
@UI.selectionField.position: 10  
key Item.SalesOrder  
      as SalesOrderID,  
...  
@UI.selectionField.position: 20  
@Search: { defaultSearchElement: true, fuzzinessThreshold: 0.7 }  
Item._SalesOrder._Customer.CompanyName as CompanyName,  
...
```

Preview:

The screenshot shows a Fiori application interface with a filter bar containing two fields: "Sales Order ID" and "Company". Both fields have small square icons next to them, indicating they are search fields.

Filter bar with predefined position of search fields

Positioning and Prioritizing UI Elements

Now you can specify which field should be displayed in the list report and at which position.

To define the **order of fields** in the UI, you can use the `position` property of `dataField`-like annotations. Only the positioning order is relevant, so you can use any decimal number as the value for the positioning property.

To define the **priority of elements**, you can use the `importance` property of `dataField`-like annotations. This information, in particular, is relevant for adaptive UIs. If an UI is displayed on a small screen, elements with low priority can automatically be hidden. To define importance, you can choose the following values:

- #HIGH
- #MEDIUM
- #LOW
- None (Default)

It is not required for the `ItemPosition` field to appear as a separate field in the result Sales Order list. Therefore, this field should be hidden in the list and therefore not available for the personalization settings dialog.

```
...
@UI.hidden: true
key Item.SalesOrderItem
as ItemPosition,
...
```

In the list, the `Sales Order ID` field should get a specific label (`Sales Order`) and be displayed at the first position within the result list.

```
...
@UI.lineItem: { importance: #HIGH, label: 'Sales Order', position: 10 }
key Item.SalesOrder
as SalesOrderID,
...
```

The order of fields is continued with currency and both amount fields:

```
...
@UI.lineItem.position: { position: 40, importance: #HIGH }
Item.GrossAmountInTransacCurrency
as GrossAmount,

@UI.lineItem.position: 50
Item.NetAmountInTransactionCurrency
as NetAmount,

@UI.lineItem.position: 60
Item.TaxAmountInTransactionCurrency
as TaxAmount
...
```

The customer's `Company` name should also appear in the list:

```
...
@UI.lineItem.position: 20
Item._SalesOrder._Customer.CompanyName
as CompanyName,
...
```

The *Product* appears as third field in the list:

```
...
@UI.lineItem.position: 30
Item.Product           as Product,
...
```

Preview:

	Sales Order ID	Company	Product ID	Gross Amount	Net Amount	Tax Amount
	500000168	Compostela	HT-1060	21.42 EUR	18.00 EUR	3.42 EUR
	500000015	AVANTEL	HT-1060	10.71 EUR	9.00 EUR	1.71 EUR

Positioning of fields in the result list

Specifying Metadata for Currency Field

It is not required for the currency field to appear as a separate field in the result list. The currency code has only to be displayed in connection with the amount data instead. This is implemented by the annotation `@Semantics.currencyCode: true`. The annotation `@Semantics.amount.currencyCode: 'CurrencyCode'` defines each amount element as a currency field.

The `@Aggregation.Default: #SUM` annotation is used for analytical capabilities: The annotated amount fields are specified as so-called *measures*, that is, fields that can be aggregated.

i Note

If no `@Aggregation.Default` annotation is assigned to an element, the engine assumes no aggregation takes place.

```
...
@Semantics.currencyCode: true
Item.currency_code      as CurrencyCode,
...
@Semantics.amount.currencyCode: 'CurrencyCode'
@Aggregation.Default: #SUM
Item.GrossAmountInTransacCurrency      as GrossAmount,
...
@Semantics.amount.currencyCode: 'CurrencyCode'
@Aggregation.Default: #SUM
Item.NetAmountInTransactionCurrency    as NetAmount
...
@Semantics.amount.currencyCode: 'CurrencyCode'
@Aggregation.Default: #SUM
Item.TaxAmountInTransactionCurrency    as TaxAmount,
...
```

For more detailed information about analytical annotations and their interpretation, please refer to [Using Aggregate Data in SAP Fiori Apps \[page 263\]](#).

Preview:

Gross Amount	Net Amount	Tax Amount
246.33 USD	207.00 USD	39.33 USD
19,635 JPY	16,500 JPY	3,135 JPY
12,794.88 EUR	10,752.00 EUR	2,042.88 EUR
103.53 EUR	87.00 EUR	16.53 EUR
21.42 EUR	18.00 EUR	3.42 EUR

Currency is displayed in connection with the amount data

The aggregated fields display the sum.

Standard * ☰	HT-1040	X	Hide Filter Bar	Filters (2)	Go
Sales Order ID:	Company:				
	SAP	□			
Sales Order Items (4) Standard * ☰					
Sales Order ID	Company	Product ID	Gross Amount	Net Amount	Tax Amount
500018564	SAP	HT-1040	1,975.40 EUR	1,660.00 EUR	315.40 EUR
500018585	SAP	HT-1040	1,975.40 EUR	1,660.00 EUR	315.40 EUR
500000009	SAP	HT-1040	1,975.40 EUR	1,660.00 EUR	315.40 EUR
500000069	SAP	HT-1040	1,975.40 EUR	1,660.00 EUR	315.40 EUR
			7,901.60 EUR	6,640.00 EUR	1,261.60 EUR

The aggregated fields display the sum

Implementing Currency Conversion for Fields to be Aggregated

In order to display an amount in unitary currency, ABAP CDS provides the currency conversion function. In our example, EURO is specified as the target currency. The value 'EUR' is passed to the formal parameter `target_currency` to be linked with the `amount` value of the conversion function. In the case of an error, for example when a currency does not exist, the result is set to null value.

```
...
@Semantics.currencyCode: true
cast( 'EUR' as abap.cuky ) as TargetCurrency,
@UI.lineItem:{ label: 'Gross Amount in EUR', position: 45 }
@DefaultAggregation: #SUM

@Semantics.amount.currencyCode: 'TargetCurrency'

CURRENCY_CONVERSION(
amount          => Item.GrossAmountInTransacCurrency,
source_currency => Item.TransactionCurrency,
target_currency  => cast( 'EUR' as abap.cuky ),
exchange_rate_date => cast( '20160101' as abap.dats ),
```

```

    error_handling      => 'SET_TO_NULL' )           as ConvertedGrossAmount
...

```

Preview:

Sales Order	Company	Product ID	Gross Amount	Gross Amount in EUR
500000000	SAP	HT-1000	1,137.64 USD	1,210.26 EUR
500000000	SAP	HT-1257	1,306.62 USD	1,390.02 EUR
500000000	SAP	HT-1251	2,353.82 USD	2,504.06 EUR
500000000	SAP	HT-1252	1,544.62 USD	1,643.21 EUR
6,115,444.29 USD			6,505,791.67 EUR	

The aggregated fields display the sum for the target currency

Next: [Running Resulting App \[page 75\]](#)

5.1.3 Running Resulting App

The resulting source code that includes the metadata for the CDS view looks like this:

```

@AbapCatalog.sqlViewName: 'ZDEMO_SOI_ADV'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Advanced Reporting for Sales Order Items'
@Search.searchable : true
@OData.publish: true
@UI.headerInfo:{ typeName: 'Sales Order Item', typeNamePlural: 'Sales Order
Items' }
define view ZDEMO_CDS_SalesOrderItem_A
    as select from SEPM_I_SalesOrderItem_E as Item
{
    @UI.lineItem: { label: 'Sales Order', position: 10, importance: #HIGH }
    @UI.selectionField.position: 10
    @UI.identification: { label: 'Sales Order', position: 10, importance:
#HIGH }

    key Item.SalesOrder
        as SalesOrderID,
    @UI.hidden: true
    key Item.SalesOrderItem
        as ItemPosition,
        @UI.lineItem.position: 20
        @UI.selectionField.position: 20
        @Search: { defaultSearchElement: true, fuzzinessThreshold: 0.7 }
        @UI.identification: { label: 'Company Name', position: 20, importance:
#HIGH }
        Item._SalesOrder._Customer.CompanyName as CompanyName,
    @UI.lineItem.position: 30
    @Search.defaultSearchElement: true
    @UI.identification.position: 30
    Item.Product
        as Product,
    @Semantics.currencyCode: true
}

```

```

Item.TransactionCurrency           as CurrencyCode,
@UI.lineItem: { position: 40, importance: #HIGH }
@Semantics.amount.currencyCode: 'CurrencyCode'
@UI.identification: { position: 40, importance: #HIGH }
@DefaultAggregation: #SUM
Item.GrossAmountInTransacCurrency      as GrossAmount,
@UI.lineItem.position: 50
@Semantics.amount.currencyCode: 'CurrencyCode'
@UI.identification.position: 50
@DefaultAggregation: #SUM
Item.NetAmountInTransactionCurrency    as NetAmount,
@UI.lineItem.position: 60
@Semantics.amount.currencyCode: 'CurrencyCode'
@UI.identification.position: 60
@DefaultAggregation: #SUM
Item.TaxAmountInTransactionCurrency    as TaxAmount,
Item.ProductAvailabilityStatus        as ProductAvailabilityStatus,
@Semantics.currencyCode: true
cast( 'EUR' as abap.cuky ) as TargetCurrency,
@UI.lineItem:{ label: 'Gross Amount in EUR', position: 45 }
@DefaultAggregation: #SUM
@Semantics.amount.currencyCode: 'TargetCurrency'

CURRENCY_CONVERSION(
    amount          => Item.GrossAmountInTransacCurrency,
    source_currency => Item.TransactionCurrency,
    target_currency => cast( 'EUR' as abap.cuky ),
    exchange_rate_date => cast( '20160101' as abap.dats ),
    error_handling     => 'SET_TO_NULL' )           as
ConvertedGrossAmount
}

```

To make the CDS view ready for consumption, you have to activate the resulting OData service in the SAP Gateway hub.

Service Catalog				
Type	Technical Service Name	V...	Service Description	External
BEP	ZDEMO CDS SALESORDERITEM A CDS	1	Advanced Reporting for Sales Order Items	ZDEMO

The activated service is part of the Service Catalog in the SAP Gateway.

More on this: [Generating OData Service With Auto-Exposure \[page 19\]](#)

If you finally run the new app in the SAP Fiori Launchpad, the list reporting app provides you with various search filters, followed by the result list:

Sales Order	Company	Product ID	Gross Amount	Gross Amount in EUR	Net Amount
500000020	SAP	HT-1111	24.63 USD	26.20 EUR	20.70 USD
500000045	SAP	HT-1111	24.63 USD	26.20 EUR	20.70 USD
500000270	SAP	HT-1111	24.63 USD	26.20 EUR	20.70 USD
500000295	SAP	HT-1111	24.63 USD	26.20 EUR	20.70 USD
			98.52 USD	104.80 EUR	82.80 USD

Resulting overview page of the list reporting app

GENERAL INFORMATION

General Information

Sales Order ID: [Details Page](#)

500000020

Company:

SAP

Product ID:

HT-1111

Gross Amount:

24.63 USD

Net Amount:

20.70 USD

Tax Amount:

3.93 USD

...ondemand.com/.../flpSandbo...

Navigating to Details page

More on this: [Consume Business Data Using SAP Fiori Elements \[page 27\]](#)

5.2 Developing New Transactional Apps

Introduction

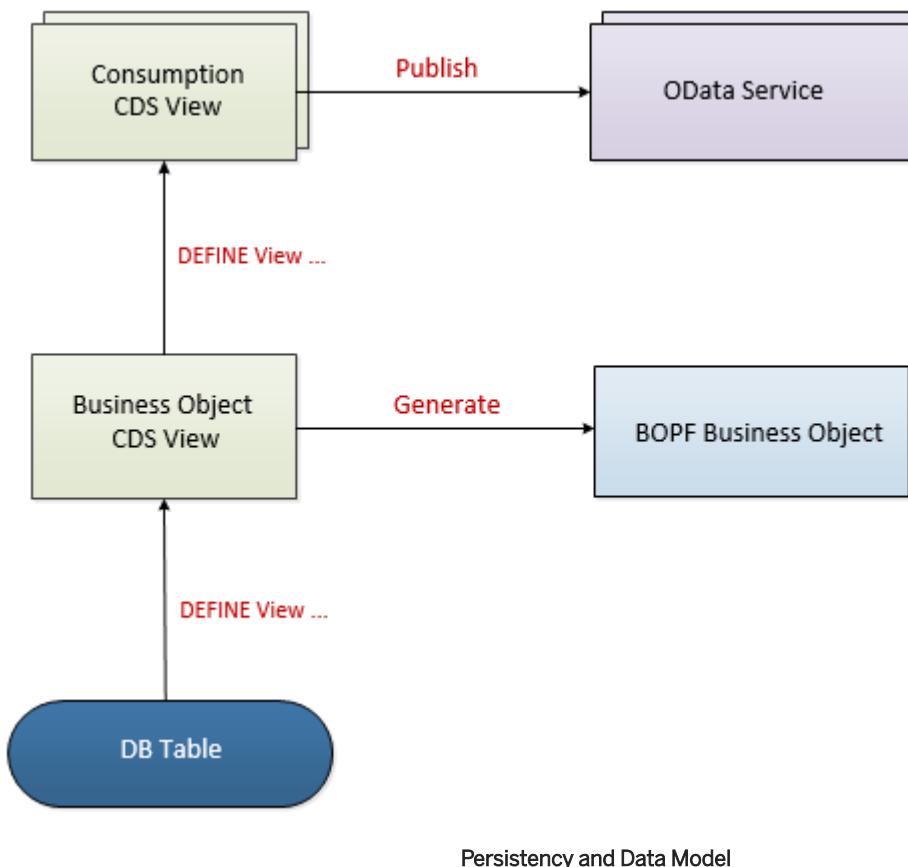
Let us assume you have to develop a completely new transactional application for SAP Fiori UI in such a way that the persistence model does not provide any GUIDs as primary keys to the consumer, but application-specific semantic keys.

In this chapter, we will develop - step by step - quite a simple sales order application, starting with the creation of a basic persistence model. We will then define a normalized data model, followed by the provision of a service-specific consumption view. After this, we are going to build a Fiori UI using Smart Templates as Fiori building blocks and test the resulting app within the SAP Fiori Launchpad. In our final step, we will extend the application's business logic with actions that are implemented with the help of BOPF API.

Persistency and Data Model

As with any other business application, a set of database tables and other Dictionary objects defines the persistence model of a transaction application. As a data source for a business object to be generated, a database table needs to be provided for each so-called business object CDS view (also referred to as business object views). Generally, business object views are normalized CDS views that define the hierarchy of entities by using associations and `@ObjectModel` annotations. A business object view is defined on top of the underlying database table or the CDS view and exposes all elements that are defined in the `SELECT` list, including the key elements that correspond to the primary keys of the underlying database table. In the transactional scenario of the programming model for SAP Fiori, the business object view is required for business object generation on the basis of the BOPF framework.

A consumption CDS view (also referred to as consumption view) allows the consumption of a business object view in a different manner by different OData services. Each service-specific view is defined on top of a business object view and exposes its fields. A consumption view enhances the data model with metadata, additional associations, or even with transient fields that fit to a given consumption use case. In particular, the consumption views are used to enhance the data model with UI-specific annotations for later consumption in Fiori UIs.



Contents

- [Defining the Business Object \[page 79\]](#)
- [Implementing a Service-Specific Consumption View \[page 89\]](#)
- [Running the Resulting SAP Fiori App \[page 92\]](#)
- [Extending Apps with Quick Actions \[page 96\]](#)

5.2.1 Defining the Business Object

For our sample scenario, we will first create a suitable database table for Sales Order header data using ABAP Dictionary tools and then use this as a data source for a new data model.

In order to be closer to the classical data model of the Suite, we introduce normalized data models -defined on top of the respective database tables - that have already been shipped as part of the EPM sample data model. These models expose semantic primary keys that consist of one or more fields. To be able to access data from other entities, a set of associations will be defined in the new data model.

After activation of the data definition, the BOPF runtime generates a corresponding business object.

Using the BOPF test environment, we can test the core services such as creating, updating, or deleting instances (CRUD operations) of the generated business object.

Procedure

1. [Modelling Normalized Data Persistence \[page 80\]](#)
2. [Adding Business Object Semantics to Data Model \[page 82\]](#)
3. [Testing CRUD Operations in the BOPF Test Shell \[page 87\]](#)

5.2.1.1 Modelling Normalized Data Persistence

Sales order data can be distributed generally across multiple database tables: a table for the sales order (header data), a table for business partners, or for currency and status information. With the provision of normalized persistence, each individual database table serves for arranging data into logical groupings such that each one describes a small part of the whole. In the context of business applications, each table field corresponds to an attribute of the business object that is represented by the entire database table. Each table row represents a unique instance of that business object and must be different in some way from all other rows (that is, no duplicate rows are possible).

Starting Point and Prerequisites

To provide data persistence for our sample Sales Order scenario, we assume that you...

- Have the standard developer authorization profile to create ABAP development objects with ABAP Development Tools.
- Reuse the data elements from EPM sample data model ([SNWD_*](#)) when creating table fields for Sales Order (header) table.
- Access related business data from associated views or tables that originate in EPM sample data model.

For our sample scenario, we will first create a suitable database table (ZTAB_SO) for sales order data using ADT ABAP Dictionary tools and then use this table as a data source for a new CDS-based data model.

Database Table ZTAB_SO for Sales Order Data

Field	Data Element
CLIENT (key)/(initial value)	MANDT
SALESORDER (key)/(initial value)	SNWD_SO_ID
BUSINESSPARTNER	SNWD_PARTNER_ID

Field	Data Element
CURRENCYCODE	SNWD_CURR_CODE
GROSSAMOUNT	SNWD_TTL_GROSS_AMOUNT
NETAMOUNT	SNWD_TTL_NET_AMOUNT
BILLINGSTATUS	SNWD_SO_CF_STATUS_CODE
OVERALLSTATUS	SNWD_SO_OA_STATUS_CODE
.INCLUDE	/BOBF/S_LIB_ADMIN_DATA

→ Tip

The standard administration data, such as the respective user or the time of creation, are included in the table by means of a predefined BOPF structure /bobf/s_lib_admin_data.

Procedure

Since SAP NetWeaver AS for ABAP 7.52 SP00, you can create and work with database tables in ABAP Development Tools (ADT) using the source-based editor.

1. Open the source-based ABAP Dictionary editor in ADT and create the database table **ZTAB_SO** with the fields as they are listed below. **For further information, see:**
2. Edit the database table as follows:

```

@EndUserText.label : 'Demo: Sales Order Table'
@AbapCatalog.enhancementCategory : #EXTENSIBLE_CHARACTER_NUMERIC
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #L
@AbapCatalog.dataMaintenance : #ALLOWED
define table ztab_so {
    @AbapCatalog.ForeignKey.screenCheck : true
    key client      : mandt not null
    with foreign key [0..*,1] t000
        where mandt = ztab_so.client;
    key salesorder   : snwd_so_id not null;
    businesspartner : snwd_partner_id;
    currencycode    : snwd_curr_code;
    @Semantics.amount.currencyCode : 'ztab_so.currencycode'
    grossamount     : snwd_ttl_gross_amount;
    @Semantics.amount.currencyCode : 'ztab_so.currencycode'
    netamount       : snwd_ttl_net_amount;
    billingstatus   : snwd_so_cf_status_code;
    overallstatus   : snwd_so_oa_status_code;
    include /bobf/s_lib_admin_data;
}

```

3. **Activate** the table.

Results

Using the time-proven ABAP Dictionary tools, you created and activated a database table for Sales Orders ([ZTAB_SO](#)) as a part of normalized data persistence.

On the basis of the table [ZTAB_SO](#) we will generate a corresponding business object. Each table field corresponds to an attribute of the Sales Order business object (to be created) represented by the entire table.

Related Information

[Database Tables](#)

5.2.1.2 Adding Business Object Semantics to Data Model

This topic demonstrates how you can implement a Business Object view to provide a normalized data model for a simple Sales Order application.

Business Object View

Each business object view must contain a specific set of `@ObjectModel` CDS annotations, indicating - for example - the root of the given entity. The business object view of the root entity needs additional CDS annotations in order to trigger the automatic generation of the respective BOPF business object, which is named after the respective business object view.

The annotations (required) for each **root entity** are:

Annotation and Values	Effect
<code>@ObjectModel.modelCategory: #BUSINESS_OBJECT</code>	Serves for semantic categorization only (the CDS view represents a business object) in the context of the Virtual Data Model (VDM). The model category is an optional annotation and has no runtime effect. We recommend adding this annotation to views representing the root node of a business object.
<code>@ObjectModel.compositionRoot: true</code>	Defines the root of the compositional hierarchy for the business object to be created
<code>@ObjectModel.transactionalProcessingEnabled: true</code>	Enables transactional runtime support

In addition, the following annotations are required for **all editable entities** (including the root entity):

Annotation and Values	Effect
@ObjectModel.writeActivePersistence: '<database_table/view>'	Specifies the database table or the database view for storing BO data changes that result from transactional behavior
@ObjectModel.createEnabled: true	Allows you to create new business object instances
@ObjectModel.deleteEnabled: true	Allows you to delete business object instances
@ObjectModel.updateEnabled: true	Allows you to update existing business object instances

More on this: [OData Annotations \[page 441\]](#)

Starting Point and Prerequisites

For our Sales Order scenario, we assume that...

- The DDL-based data definition, as the corresponding development object, is already created in the ABAP package of your choice. In our case, both the data definition and the business object view are named as ZDEMO_I_SALESORDER_TX.
- The persistence model is already defined. In particular, you have already created and activated the database table ZTAB_SO that serves as data source for the business object view.

⚠ Caution

Whenever the table field names and the corresponding element names in the business object view are different (except for camel-case), this will cause problems when activating the view. For further information on how to avoid such an issue, see also: [Mapping CDS View-Elements onto Table Fields \[page 286\]](#)

- The key in the data model corresponds to the primary key (SALESORDER) of the underlying database table.

Implementing the Data Model

The following listing provides you with the implementation of a rather elementary Sales Order data model, where the database table ZTAB_SO is defined as the data source for the corresponding CDS view ZDEMO_I_SalesOrder_TX (camel case notation!). In addition, the table ZTAB_SO also serves as a storage place for business object data changes that result from transactional behavior. Therefore, this table is specified in the @ObjectModel.writeActivePersistence annotation. To be able to access business data from other entities, a set of associations is defined as part of the data model. These associations refer to CDS views that are already provided as part of the EPM data model.

```
@AbapCatalog.sqlViewName: 'ZDEMO_I_SO'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Sales Order for transactional app'
```

```

@ObjectModel.semanticKey: 'SalesOrder'

@ObjectModel.modelCategory: #BUSINESS_OBJECT
@ObjectModel.compositionRoot: true
@ObjectModel.transactionalProcessingEnabled: true
@ObjectModel.writeActivePersistence: 'ZTAB_SO'

@ObjectModel.createEnabled: true
@ObjectModel.deleteEnabled: true
@ObjectModel.updateEnabled: true

define view ZDEMO_I_SalesOrder_TX

    as select from ZTAB_SO as SalesOrder -- the sales order table is the data
    source for this view

        association [0..1] to SEPM_I_BusinessPartner           as _BusinessPartner
        on $projection.BusinessPartner = _BusinessPartner.BusinessPartner

        association [0..1] to SEPM_I_Currency                 as _Currency
        on $projection.CurrencyCode = _Currency.Currency

        association [0..1] to SEPM_I_SalesOrderBillingStatus as _BillingStatus
        on $projection.BillingStatus = _BillingStatus.SalesOrderBillingStatus

        association [0..1] to Sepm_I_SalesOrdOverallStatus   as _OverallStatus
        on $projection.OverallStatus = _OverallStatus.SalesOrderOverallStatus

    {
        key SalesOrder.salesorder           as SalesOrder,
        @ObjectModel.foreignKey.association: '_BusinessPartner'
        SalesOrder.businesspartner         as BusinessPartner,

        @ObjectModel.foreignKey.association: '_Currency'
        @Semantics.currencyCode: true
        SalesOrder.currencycode           as CurrencyCode,

        @Semantics.amount.currencyCode: 'CurrencyCode'
        SalesOrder.grossamount            as GrossAmount,

        @Semantics.amount.currencyCode: 'CurrencyCode'
        SalesOrder.netamount              as NetAmount,

        @ObjectModel.foreignKey.association: '_BillingStatus'
        SalesOrder.billingstatus          as BillingStatus,

        @ObjectModel.foreignKey.association: '_OverallStatus'
        SalesOrder.overallstatus          as OverallStatus,

        /* Associations */
        _BusinessPartner,
        _Currency,
        _BillingStatus,
        _OverallStatus
    }

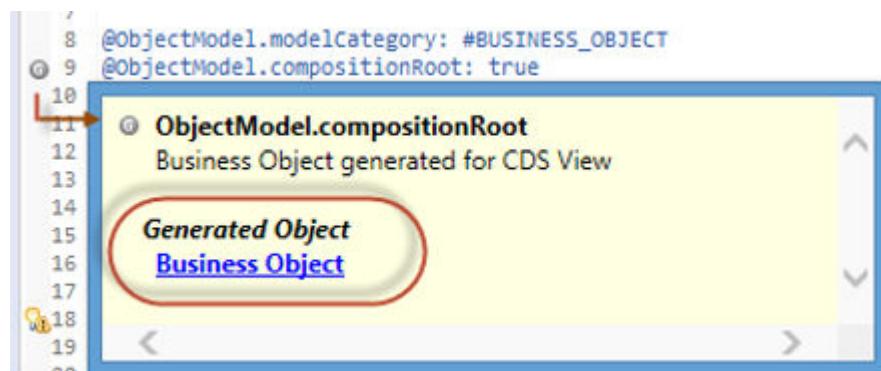
```

Activating the Data Definition

After ensuring that the syntax of the CDS view is complete and correct, activate the data definition (as the corresponding development object).

Checking the Generated Business Object

As a result of successful activation, the BOPF runtime creates a business object `ZDEMO_I_SALESORDER_TX` (upper case notation!) that is named after, and is stored in, the same package as the underlying CDS view, in our case: `ZDEMO_I_SalesOrder_TX` (camel case notation!). The decorator within the editor ruler indicates that the generation process has been successful. If you move the cursor over the decorator, an info screen informs you that the business object is generated for the CDS view. In addition, the info screen provides you with a link to the business object editor.



Quick info screen after successful creation of the business object

On the entry page of the BO editor, you can identify the Constants Interface that has been generated together with the business object.

→ Remember

Constants Interface is an ABAP interface that is dedicated to a specific business object. The interface includes constants for each business object's entity like nodes, attributes, actions and so on. We will be using it later on when we add an action to the business object generated.

Business Object Overview

General Information

Name:	ZDEMO_I_SALESORDER_TX
Source CDS View:	ZDEMO_I_SALESORDER_TX
Description:	#GENERATED#
Category:	Business Object
Constants Interface:	ZIF_DEMO_I_SALESORDER_TX_C

What's Next

- [Go to the nodes of this BO](#)
- [Go to the ROOT node](#)

Overview Nodes

Entry page of the BO editor

If you now choose [Go to the ROOT node](#), you can view some more details at the entity level of the business object. You must know these when you proceed to use the BOPF API - for example, when implementing an action. Here you can find the names of the ABAP Dictionary objects that have been generated together with the business object.

Node Overview

General Information

Name: *	ZDEMO_I_SALESORDER_TX
Source CDS View:	ZDEMO_I_SALESORDER_TX
Description:	#GENERATED#
Persistent Structure:	ZSDEMO_I_SALESORDER_TX_D
Transient Structure:	
Combined Structure:	ZSDEMO_I_SALESORDER_TX
Combined Table Type:	ZTDEMO_I_SALESORDER_TX
Database Table:	ZTAB_SO

Create Enabled Update Enabled Delete Enabled

Text Node

Generated ABAP Dictionary Objects

Overview Alternative Keys Properties Associations Actions Determinations

Root node of the BO – Entity Level

If you click on the *Properties* tab, you can view the status of each element (attribute) related to the business object created. This information is important for you when you proceed to define the static field control in the business object view later on.

The screenshot shows the 'Properties' tab of a business object configuration interface. At the top, there's a header bar with tabs: Overview, Alternative Keys, Properties (which is selected and highlighted in blue), Associations, Actions, Determinations, and a help icon. Below the header is a table with four columns: Element Name, Enabled, ReadOnly, and Mandatory. The table lists various attributes with their respective status checkboxes checked or unchecked. The attributes listed are: KEY, PARENT_KEY, ROOT_KEY, SALESORDER, BUSINESSPARTNER, CURRENCYCODE, GROSSAMOUNT, NETAMOUNT, BILLINGSTATUS, OVERALLSTATUS, CREA_DATE_TIME, CREA_UNAME, LCHG_DATE_TIME, and LCHG_UNAME.

Element Name	Enabled	ReadOnly	Mandatory
KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
PARENT_KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ROOT_KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SALESORDER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BUSINESSPARTNER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CURRENCYCODE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GROSSAMOUNT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
NETAMOUNT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BILLINGSTATUS	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
OVERALLSTATUS	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CREA_DATE_TIME	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CREA_UNAME	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
LCHG_DATE_TIME	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
LCHG_UNAME	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure: Root node of the BO – Properties level

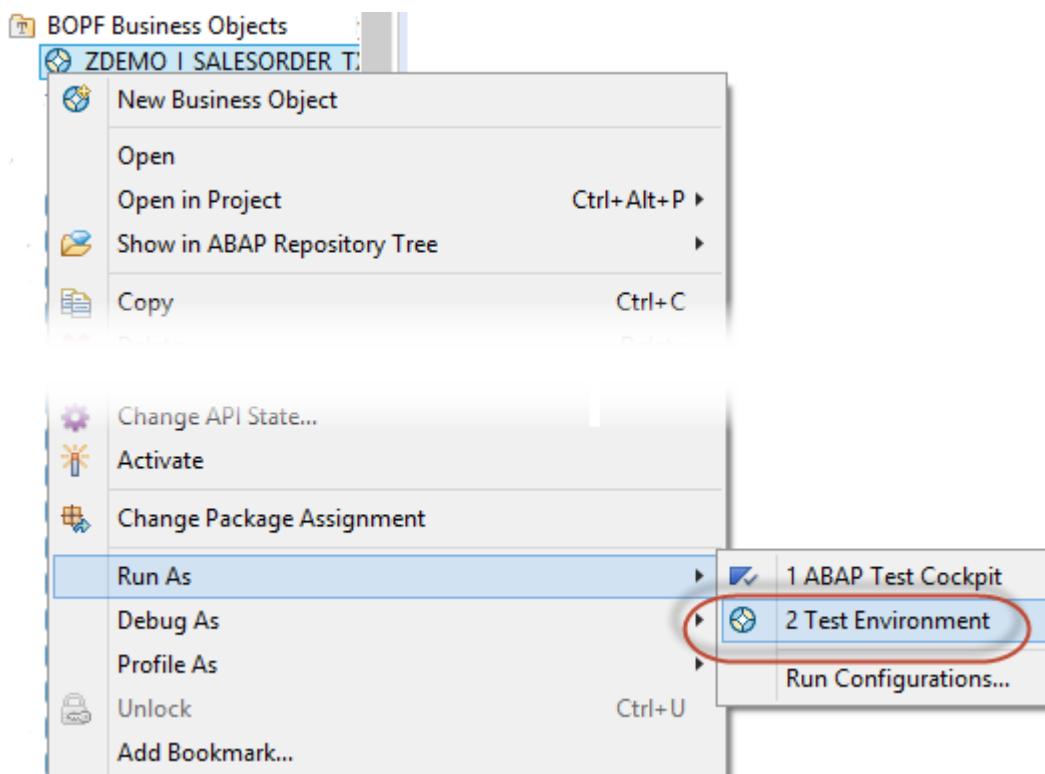
5.2.1.3 Testing CRUD Operations in the BOPF Test Shell

Starting from your ABAP Development Tools (ADT) client, you can test the generated business object in the integrated BOPF test environment. Using the BOPF test environment, you can test the core services such as creating, updating, or deleting business object instances without writing any test code.

Procedure

Launching the BOPF Test Environment in ABAP Development Tools

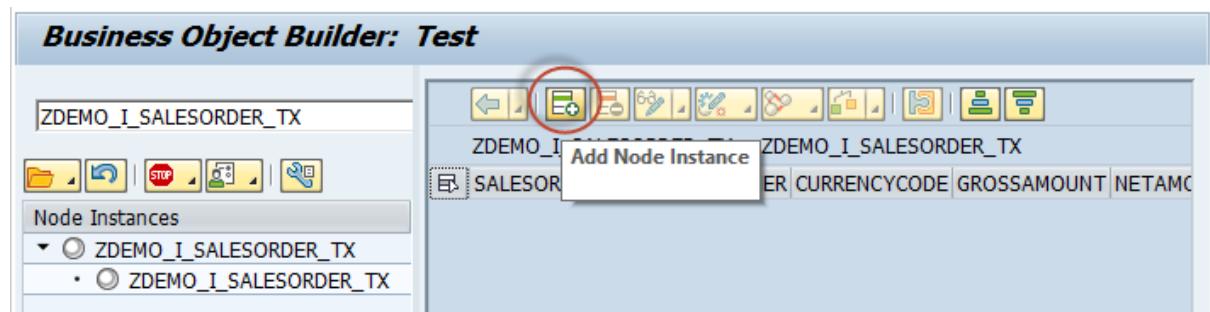
To launch the BOPF test environment, select the node of the generated business object in the Project Explorer tree, open the context menu, and choose **Run As > Test Environment**.



Launching the BOPF test environment

The BOPF test shell appears in the SAP GUI window and opens the test session.

Creating a Node Instance



Creating a node instance in the BOPF test shell

To create a new node instance, select the corresponding node in the *Node Instances* navigation pane and click the (*Add Node Instance*) push button from the toolbar of the editor pane – as depicted in the figure below.

Business Object Builder: Test

The screenshot shows the SAP Business Object Builder Test interface. On the left, there's a tree view under 'Node Instances' with several nodes expanded, including 'ZDEMO_I_SALESORDER_TX'. On the right, a table displays sales order data with columns: SALESORDER, BUSINESSPAR..., CURR..., GROSSAMO..., NETAMO..., BILL..., and OVER... . A specific row (SALESORDER 5000000111) is highlighted and circled in red.

SALESORDER	BUSINESSPAR...	CURR...	GROSSAMO...	NETAMO...	BILL...	OVER...
5000000004	1000000002	USD	1.001,23	945,34		N
5000000031	1000000000	EUR	345,34	289,31		
5000000111	1000000007	EUR	985,00	0,00		N

Entering field values of the new node instances

Finally you have the option to save your Sales Order data by choosing [Save](#) from the toolbar. The data is saved in the database table that is assigned to the node of the *Sales Order* business object.

Results

You can either save the changes you have made or discard them. To finish a test session, close the test environment.

Within a few minutes you have verified that the generated sales order business object works correctly, without writing any test code

5.2.2 Implementing a Service-Specific Consumption View

Based on the Sales Order BO view, we are now going to implement the data model for consumption with the help of the consumption view.

Consumption View

The `ObjectModel` annotations that are used in the consumption view provide transactional related aspects of the business data model.

The annotations (required) for each **root entity** are:

Annotation and Values	Effect
<code>@ObjectModel.transactional ProcessingDelegated: true</code>	Indicates that transactional access to the consumption view is delegated to the transactional runtime of the underlying view (which is annotated with <code>@ObjectModel.transactionalProcessingEnabled: true</code>).

In addition, the following annotations are required for **all editable entities** (including the root entity):

Annotation and Values	Effect
@ObjectModel.createEnabled : true	Allows the end user to create new business object instances
@ObjectModel.deleteEnabled : true	Allows the end user to delete business object instances
@ObjectModel.updateEnabled : true	Allows the end user to update existing business object instances

More on this: [ObjectModel Annotations \[page 428\]](#)

→ Remember

CDS rule: Remember to double-maintain the annotations that have the `VIEW` scope. In CDS views, only the annotations with `ELEMENT` and `ASSOCIATION` scope are inherited from the business object view.

Prerequisites

For our Sales Order scenario, we assume that...

- The DDL-based data definition, as the corresponding development object, is already created. In our case, the DDL data definition and the CDS view are named `ZDEMO_C_SALESORDER_TX` and `ZDEMO_C_SalesOrder_TX` respectively.
- The business object CDS view that you implemented in the previous step serves as the data source for the consumption view.
- The key in the data model in the consumption view corresponds to the primary key of the underlying database table.

Implementing the Data Model for Consumption

In this implementation, the business object view is the data source of the consumption view `ZDEMO_C_SalesOrder_TX`. The `@ObjectModel.transactionalProcessingDelegated: true` annotation indicates that transaction requests to the consumption view are delegated to the underlying business object view.

To expose the data model and its metadata to the OData service, the consumption view is annotated with `@OData.publish: true`.

The SELECT list includes a set of fields that are relevant for consumption in a user interface (UI). The key is specified as a semantic key `SalesOrder`. This is implemented by the annotation `@ObjectModel.semanticKey: 'SalesOrder'`.

```
@AbapCatalog.sqlViewName: 'ZDEMO_C_SO'
@AccessControl.authorizationCheck: #NOT_REQUIRED
```

```

@EndUserText.label: 'Sales Order for transactional app'
@ObjectModel.semanticKey: 'SalesOrder'
@ObjectModel.transactionalProcessingDelegated: true
@ObjectModel.createEnabled: true
@ObjectModel.deleteEnabled: true
@ObjectModel.updateEnabled: true

@UI.headerInfo: { typeName: 'Sales Order', typeNamePlural: 'Sales Orders' }
@OData.publish: true

define view ZDEMO_C_SalesOrder_TX
    as select from ZDEMO_I_SalesOrder_TX as Document
{
    @UI.lineItem.position: 10
    @UI.identification.position: 10
    key Document.SalesOrder,
    @UI.lineItem.position: 20
    @UI.identification.position: 20
    Document.BusinessPartner,
    Document.CurrencyCode,
    @UI.lineItem.position: 50
    @UI.identification.position: 50
    Document.GrossAmount,
    @UI.lineItem.position: 60
    @UI.identification.position: 60
    Document.NetAmount,
    @UI.lineItem.position: 30
    @UI.selectionField.position: 30
    @UI.identification.position: 30
    Document.BillingStatus,
    @UI.lineItem.position: 40
    @UI.selectionField.position: 40
    @UI.identification.position: 40
    Document.OverallStatus,
    /* Exposing value via associations */
    @UI.lineItem: { value: '.CompanyName', position: 15 }
    Document._BusinessPartner,
    Document._Currency,
    Document._BillingStatus,
    Document._OverallStatus
}

```

Activating the Data Definition of Consumption View

After successful activation of the corresponding data definition development object, the ABAP Development Tools generates several SAP Gateway artifacts that are stored in the back end of the application server AS ABAP and are required for OData service activation in the SAP Gateway hub. These artifacts include:

- The actual service artifact with the technical name <CDS_VIEW>_CDS. You can find it at SAP Gateway Business Suite Enablement - Service object (object type: *R3TR IWSV*).
- An SAP Gateway model (object type: *R3TR IWMO*) with the name <CDS_VIEW>_CDS.
- An ABAP class *ZCL_<CDS_VIEW>* (in case of local objects) or *<Namespace>CL_<CDS_VIEW>* (other objects) that is used to provide model metadata to the SAP Gateway service.

All generated service artifacts are automatically added to the same ABAP package (and also to the same transport request) as the underlying consumption CDS view.

Activating the OData Service

To make your data model ready for consumption, you only have to activate the resulting OData service in the SAP Gateway hub.

More on this: [Generating OData Service With Auto-Exposure \[page 19\]](#)

Service Catalog				
Type	Technical Service Name	V	Service Description	External Service Name
BEP	ZDEMO_C_SALESORDER_TX_CDS	1	Sales Order for transactional app	ZDEMO_C_SALESORDER_TX

Activated OData service is listed in the Service Catalog of the SAP Gateway

→ Tip

To verify that the activated OData service works correctly, you can test the service.

More on this: [Test the Activated OData Service \[page 26\]](#)

5.2.3 Running the Resulting SAP Fiori App

To make sure that all parts of the application work correctly for the end user, we are now going to run the resulting application.

Prerequisites

For the corresponding OData service we assume that...

- The consumption view is exposed as an OData service using the so-called auto exposure approach (`@OData.publish: true`).

More on this: [Generating OData Service With Auto-Exposure \[page 19\]](#)

- Smart Templates as Fiori building blocks are used in the course of UI development.
- More on this:** [Consume Business Data Using SAP Fiori Elements \[page 27\]](#)

Procedure

If you run the new app based on Fiori Smart Templates in the Fiori Launchpad, the resulting UI provides you with a list of sales orders, including all fields that you exposed for consumption.

Sales Order ID	Company	Business Partner ID	Confirmation Status	Overall Status	Gross Amount	Net Amount
5000000004	DelBont Industries	DelBont Industries 100000002	Initial	New N	1,001.23 USD	945.34 USD >
5000000031	SAP	SAP 100000000	Initial		345.34 EUR	289.31 EUR >
5000000111	Laurent	Laurent 100000007	Initial	New N	985.00 EUR	0.00 EUR >

Resulting UI screen with the list of sales orders

1. To create a new Sales Order instance, click the + icon and edit all the fields to specify the Sales Order header.

i Note

Each field that results from an association's path provides you with value help options. **More on this:**

The screenshot shows the SAP Fiori interface for creating a new sales order. The top navigation bar includes the SAP logo, a search icon, and a user dropdown set to "Default User". The main title is "Sales Order" with a back arrow. Below it, the sub-title is "New Sales Order". A section header "GENERAL INFORMATION" is followed by several input fields:

- "General Information"
- "*Sales Order ID": Input field containing "5000000444".
- "Business Partner ID": Input field containing "100000022".
- "Confirmation Status": Input field with a red circle drawn around it.
- "Overall Status": Input field containing "N".
- "Gross Amount": Input field containing "22.21" next to a currency selector showing "EUR".
- "Net Amount": Input field containing "19.13" next to a currency selector showing "EUR".

At the bottom right are "Save", "Cancel", and a refresh icon.

Object page for editing individual values

2. To update individual fields of a Sales Order record, click the relevant row.

Sales Orders (4) Standard						
Sales Order ID	Company	Business Partner ID	Confirmation Status	Overall Status	Gross Amount	Net Amount
5000000004	DelBont Industries	DelBont Industries 100000002	Initial	New N	1,001.23 USD	945.34 USD >
5000000031	SAP	SAP 100000000	Initial		345.34 EUR	289.31 EUR >
5000000111	Laurent	Laurent 100000007	Initial	New N	985.00 EUR	0.00 EUR >
5000000444	Quimica Madrilenos	Quimica Madrilenos 100000022	Initial	New N	22.21 EUR	19.13 EUR >

Selecting a data record for editing

- In the screen that now appears, click the *Edit* button and finally save the changed entries.

Sales Order

GENERAL INFORMATION

General Information

Sales Order ID:
5000000444

Business Partner ID:
Quimica Madrilenos (100000022)

Confirmation Status:
Initial

Overall Status:
New (N)

Gross Amount:
22.21 EUR

Net Amount:
19.13 EUR

Switching to edit mode

5.2.4 Extending Apps with Quick Actions

You might wish to extend the scenario that was developed in the previous chapter to include additional functions. For example, you might be interested in extending the business logic so that status changes to individual instances of the Sales Order business object can be executed, without the end user having to switch to edit mode. In this case, the so-called quick actions come in to play

What is a Quick Action About?

Quick actions are “one-click” actions with optional parameters. They are used for performing an action on each individual record without the need for the end user to switch to edit mode. For quick actions, no state needs to be kept on the client or back-end server. A quick action triggers the business logic in the back end that is immediately executed (in our case: written to the database. In a UI screen, the detail page (object page) might show the data as read-only and also offer quick actions without the need to switch to edit mode.

→ Tip

You have the option to provide quick actions also in non-transactional areas, that is, in purely list-reporting development scenarios.

How do I Implement Quick Actions?

Action implementation and action control requires business logic that is implemented on the basis of the BOPF API. Therefore, quick actions are implemented as standard BOPF actions that are assigned to the related business object node. At runtime, a (quick) action can modify a node instance of the assigned business object or node instance of other associated business objects.

More on this:

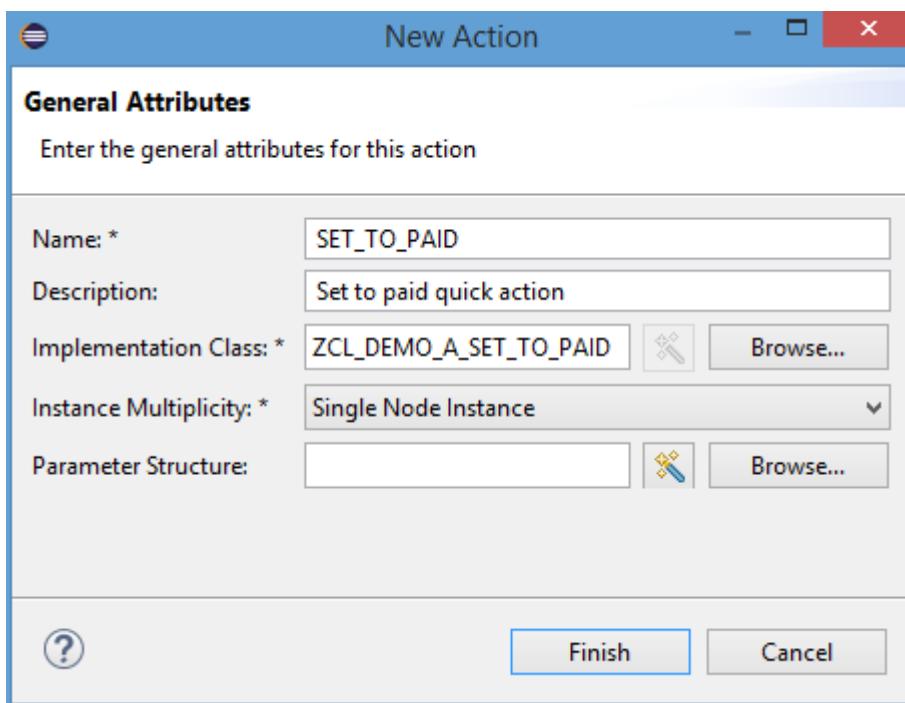
5.2.4.1 Adding a New BOPF Action

Context

To define the behavior of a sales order business object at runtime, we need to add a corresponding action entity to the generated business object. For our sample scenario, we are going to create an action `SET_TO_PAID` to change the status of the sales order item to `PAID` after a buyer pays for a product provided by a seller.

Procedure

1. To create an action of the business object generated in the step [Defining the Business Object \[page 79\]](#), open the editor of this business object and select the *Actions* tab.
2. Then choose *New* to launch the *New Action* wizard.
3. Specify the details:
 - *Name*: SET_TO_PAID
 - *Description*: Enter a short description that describes the purpose of the action.
 - *Implementation Class*: Accept the name for the action class that is suggested by the wizard.
 - *Instance Multiplicity*: Select *Single Node Instance* from the list box. This option allows the action to operate on a single sales order node instance. In other words: In the resulting app, we don't want to allow the end user to execute the action on a set of multiple node instances.
 - *Parameter Structure*: We leave this field empty since we are not going to use action parameters in our scenario.



New action wizard

4. Complete the creation procedure with *Finish*.
5. Activate the business object's actions class.

Results

The BOPF framework assigns the new action to the business object node and creates an action class that implements the BOPF action interface /BOBF/IF_FRW_ACTION.

Name	Implementation Class	Instance Multiplicity
LOCK_ZDEMO_I_SALESORDER_TX	/BOBF/CL_LIB_A_LOCK	Multiple Node Instances
SET_TO_PAID	ZCL DEMO A SET TO PAID	Single Node Instance

Added action in the BO editor

Related Information

[Actions for CDS-Based BOPF Business Objects \[page 60\]](#)

5.2.4.2 Implementing the Action

Each action in BOPF must implement the execute method of the action interface /BOBF/IF_FRW_ACTION.

Implementing the SET_TO_PAID Action

The listing below is used to implement the SET_TO_PAID action of the generated business object ZDEMO_I_SALESORDER_TX. A consumer can use this action to mark the status of the sales order as paid.

```

CLASS zcl_demo_a_set_to_paid DEFINITION
  PUBLIC
    INHERITING FROM /bobf/cl_lib_a_supercl_simple
    FINAL
    CREATE PUBLIC .

  PUBLIC SECTION.

    METHODS /bobf/if_frw_action~execute REDEFINITION .
    PROTECTED SECTION.
    PRIVATE SECTION.
  ENDCLASS.

  CLASS zcl_demo_a_set_to_paid IMPLEMENTATION.

    METHOD /bobf/if_frw_action~execute.
  
```

```

" Typed with node's combined table type
DATA(lt_sales_order) = VALUE ztdemo_i_salesorder_tx( ).

" READING BO data -----
" Retrieve the data of the requested node instance
io_read->retrieve(
  EXPORTING
    iv_node      = is_ctx-node_key
    it_key       = it_key
  IMPORTING
    et_data      = lt_sales_order
).

" WRITING BO data -----
LOOP AT lt_sales_order ASSIGNING FIELD-SYMBOL(<s_sales_order>).

" Set the attribute billing_status to new value
<s_sales_order>-billingstatus = 'P'. " PAID

" Set the attribute overall_status to new value
IF
  <s_sales_order>-overallstatus = 'N' OR <s_sales_order>-overallstatus =
  .
  .
  <s_sales_order>-overallstatus = 'P'. " PAID
ENDIF.

" Update the changed data (billig_status) of the node instance
io_modify->update(
  EXPORTING
    iv_node      = is_ctx-node_key
    iv_key       = <s_sales_order>-key
    iv_root_key  = <s_sales_order>-root_key
    is_data      = REF #( <s_sales_order>-node_data )
    it_changed_fields = VALUE #(
      ( zif_demo_i_salesorder_tx_c=>sc_node_attribute-zdemo_i_salesorder_tx-
        billingstatus )
      ( zif_demo_i_salesorder_tx_c=>sc_node_attribute-zdemo_i_salesorder_tx-
        overallstatus )
      )
    ).
  ENDLOOP.
ENDMETHOD.
ENDCLASS.
```

The `SET_TO_PAID` action is carried out by the `execute` method. As in most cases, the first step within the `execute` method is to retrieve relevant data of those node instances that are going to be changed. Remember, all attributes of the root node are represented by the combined structure's component.

Node Overview

General Information		
Name: *	ZDEMO_I_SALESORDER_TX	
Source CDS View:	ZDEMO_I_SALESORDER_TX	
Description:	#GENERATED#	
Persistent Structure: *	ZSDEMO_I_SALESORDER_TX_D	
Transient Structure:		
Combined Structure: *	ZSDEMO_I_SALESORDER_TX	
Combined Table Type: *	ZTDEMO_I_SALESORDER_TX	
Database Table: *	ZTAB_SO	

Dictionary artifacts referred in the action implementation

For reading BO data, a data reference `lt_sales_order` is used, where `lt_sales_order` refers to the combined table type `ztdemo_i_salesorder_tx`. It contains the keys and data of all node instances on which the action should be applied. This enables you to access the root node attribute `billing_status` and then set a new value `P` that indicates that the node instance is set to status `PAID`.

For each relevant node instance, the update method of the access object `io_modify` is called. Here the internal table `it_changed_fields` is populated with attributes that are going to be updated. Like all other entity names, the attribute names, too, are available through the [Constants Interface ZIF_DEMO_I_SALESORDER_TX_C](#). As result of the method call, the node instance is updated in the database.

5.2.4.3 Testing the Action in the BOPF Test Shell

Using the BOPF test environment, you can test not only the CRUD of business object instances, but also execute the action (which you implemented and activated in the previous step) to check whether it runs correctly.

Procedure

1. Launch the BOPF Test Environment in ABAP Development Tools
More on this: [Testing CRUD Operations in the BOPF Test Shell \[page 87\]](#)
2. Test the Action
 - a. To test the action that you implemented in the previous step, create a node for the business object in the editor pane.
 - b. Then choose [Execute Action](#) from the toolbar and click the proposed action `SET_TO_PAID`.

ZDEMO_I_SALESORDER > ZDEMO_I_SALESORDER_TX						
SALESORDER	BUSINESSPARTNER	CURR...	GRO...	NETAM...	BILLINGSTAT...	OVERALLSTA...
5000000003	1000000000	EUR	110...	92,09		

Testing action SET_TO_PAID in the BOPF test shell

As a result of action execution, the attributes *billing_status* and *overall_status* display the new value *P* (*PAID*).

ZDEMO_I_SALESORDER_TX > ZDEMO_I_SALESORDER_TX						
SALESORDER	BUSINESSPA...	CURR...	GR...	NETAM...	BILLING...	OVERALLSTATUS
5000000003	1000000000	EUR	110...	92,09	P	P

Results of action execution

Results

With only a few clicks, you have verified that the action works fine and you can now save the changes you have made or discard them. To finish a test session, close the test environment.

5.2.4.4 Enabling Actions for OData Consumption

Actions are often directly related to BO instances that you can see in a table with sales order records, for example. End users can select as a line item (that represents a BO instance) and execute certain actions on the selected item.

You can use the following UI annotation to expose actions to the consumer:

Syntax (for LineItem)

```
...
define view <CDS_VIEW> as select from <DATA_SOURCE> as ... {
  @UI.lineItem: [
    { type: '#FOR_ACTION', dataAction: 'BOPF:<BO_ACTION_1>', label: '<ACTION_LABEL_1>' },
    { type: '#FOR_ACTION', dataAction: 'BOPF:<BO_ACTION_2>', label: '<ACTION_LABEL_2>' },
    ...
  ]
  ...
}
```

```
}
```

i Note

The `dataAction` element references the technical name of an action of the BOPF. The string pattern is `BOPF:<technical name of action in BOPF>`.

Adding Annotations for Quick Action in the Consumption View

In our example, we add '`BOPF:SET_TO_PAID`' as `dataAction` element to the consumption view:

```
...
define view ZDEMO_C_SalesOrder_TX
    as select from ZDEMO_I_SalesOrder_TX as Document
{
    @UI.lineItem.position: 10
    @UI.lineItem:
    [
        { type: #FOR_ACTION, position: 1, dataAction: 'BOPF:SET_TO_PAID', label:
        'Set to Paid' }
    ]
    key Document.SalesOrder,
    ...
}
```

Reactivating the Data Definition of Consumption View

To publish the new metadata to the OData service, activate the data definition that implements the consumption view (in our case: `ZDEMO_C_SalesOrder_TX`).

Configuring Action Parameters for OData Consumption

To make the action available for consumption in the OData service, the following configuration steps in the BO editor are required. Otherwise, the action would be unknown in the OData consumption layer.

Procedure

1. If you have not yet already done so, open the editor of the business object (in our case: `ZDEMO_I_SALESORDER_TX`) and click the `Actions` tab.
2. Select the action `SET_TO_PAID` and choose `CTRL` + `Click` to open the `Action Overview`.

Actions	
New...	Delete
type filter text	
Name	Implementation Class
SET_TO_PAID	ZCL_DEMO_A_SET_TO_PAID
LOCK_ZDEMO_I_SALESORDER_TX	/BOBF/CL_LIB_A_LOCK
Ctrl+click to open	

Opening the action overview to add configuration details

3. Specify the following details:

- *Exporting Type*: Select **Node** from the list box, since the action operates on the business object's node. This option allows the action to return instances of a node that is defined by the following two fields:
 - *Exporting Parameter BO*: Enter the name of the business object.
 - *Exporting Parameter Node*: Enter the name of the business object node (in our case: the same name as for the business object).
- *Exporting Multiplicity*: Select **1** from the list box. For the resulting OData service we want to allow the action to operate on a single sales order node only.

ZDEMO_I_SALESORDER_TX

ZDEMO_I_SALESORDER_TX > ZDEMO_I_SALESORDER_TX > SET_TO_PAID

Action Overview

[Go to the actions of the node](#)

General Information

Name:*	SET_TO_PAID
Description:	Set to paid quick action
Instance Multiplicity:	Single Node Instance
Annotation:	

Implementation Details

Implementation Class: *	ZCL_DEMO_A_SET_TO_PAID	New...	Browse...
Parameter Structure:		New...	Browse...
Exporting Type:	Node		
Exporting Parameter BO:	ZDEMO_I_SALESORDER_TX	Browse...	
Exporting Parameter Node:	ZDEMO_I_SALESORDER_TX	Browse...	
Exporting Multiplicity:	1		

Overview

Added details for configuration

- Leave the other details unchanged and save the changed entries.
- Activate* the business object.

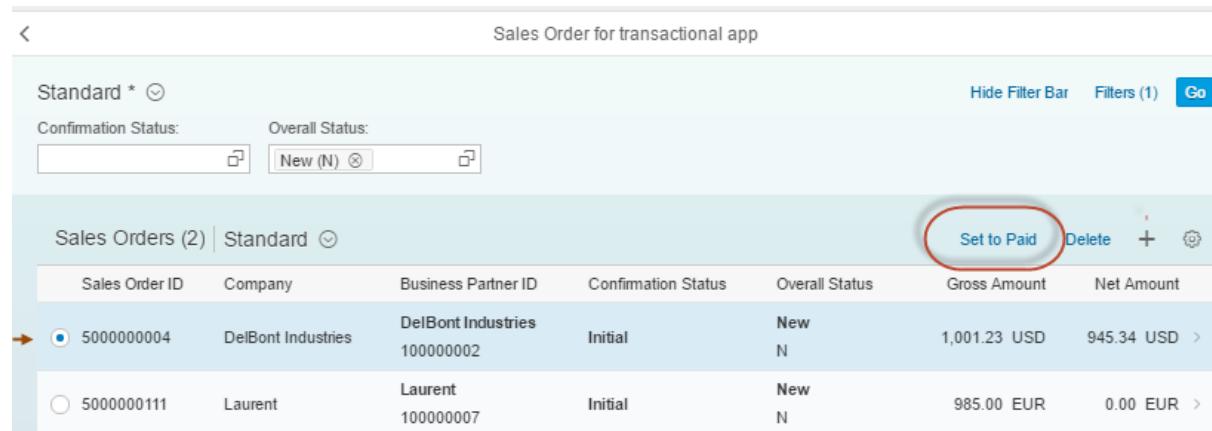
Results

The BOPF framework assigns the configuration details to the action. As a result of this configuration procedure, the action will be exported as an additional function to the OData consumption layer.

5.2.4.5 Running the Resulting SAP Fiori App

Again, you have the option to run the changed app based on *Fiori Smart Templates* in the *Fiori Launchpad* to check the action execution.

If you run the app, the resulting UI screen provides you with the label *Set to Paid* for the new action - as shown in the figure below.



Sales Order for transactional app

Standard * ☰

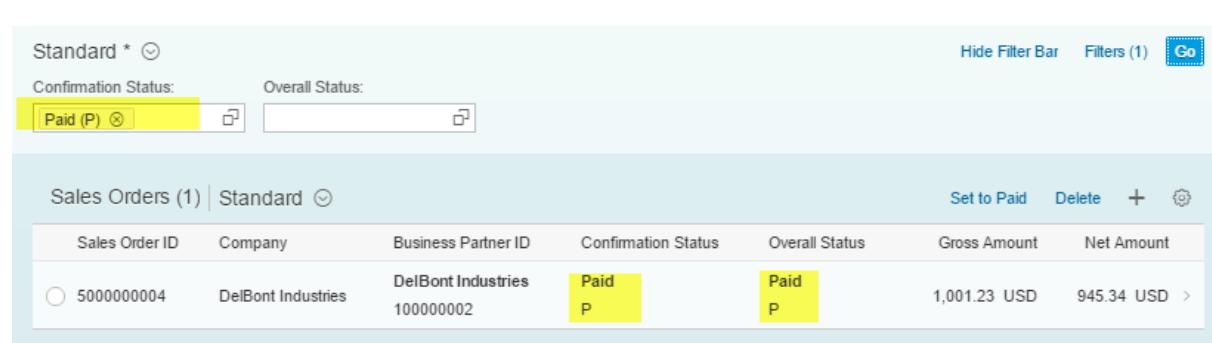
Confirmation Status: Overall Status:

New (N) ☰

Sales Orders (2) | Standard ☰

Sales Order ID	Company	Business Partner ID	Confirmation Status	Overall Status	Gross Amount	Net Amount
5000000004	DelBont Industries	DelBont Industries 100000002	Initial	New N	1,001.23 USD	945.34 USD
5000000111	Laurent	Laurent 100000007	Initial	New N	985.00 EUR	0.00 EUR

Set to Paid action is added to the app



Standard * ☰

Confirmation Status: Overall Status:

Paid (P) ☰

Sales Orders (1) | Standard ☰

Sales Order ID	Company	Business Partner ID	Confirmation Status	Overall Status	Gross Amount	Net Amount
5000000004	DelBont Industries	DelBont Industries 100000002	Paid P	Paid P	1,001.23 USD	945.34 USD

Results after action execution

5.3 Developing New Transactional Apps with Draft Capabilities

Based on an end-to-end development example, you will, from scratch, create and implement all requisite artifacts that combine transactional processing with draft capabilities.

Prerequisites

In addition to the general [Prerequisites \[page 9\]](#), the development of new transactional apps with draft capabilities requires the following:

- ABAP Application Server as of SAP NetWeaver AS for **ABAP 7.51 innovation package SP02** or higher
- Client installation of ABAP Development Tools (ADT) with the **client version 2.73** or higher

i Note

We recommend that you use the latest available ADT client, which can be downloaded from the public [update site](#).

- For draft enabling, the key of the data model must be a UUID-based key, as it is native in the BOPF programming model
- Expert knowledge of the BOPF programming model.

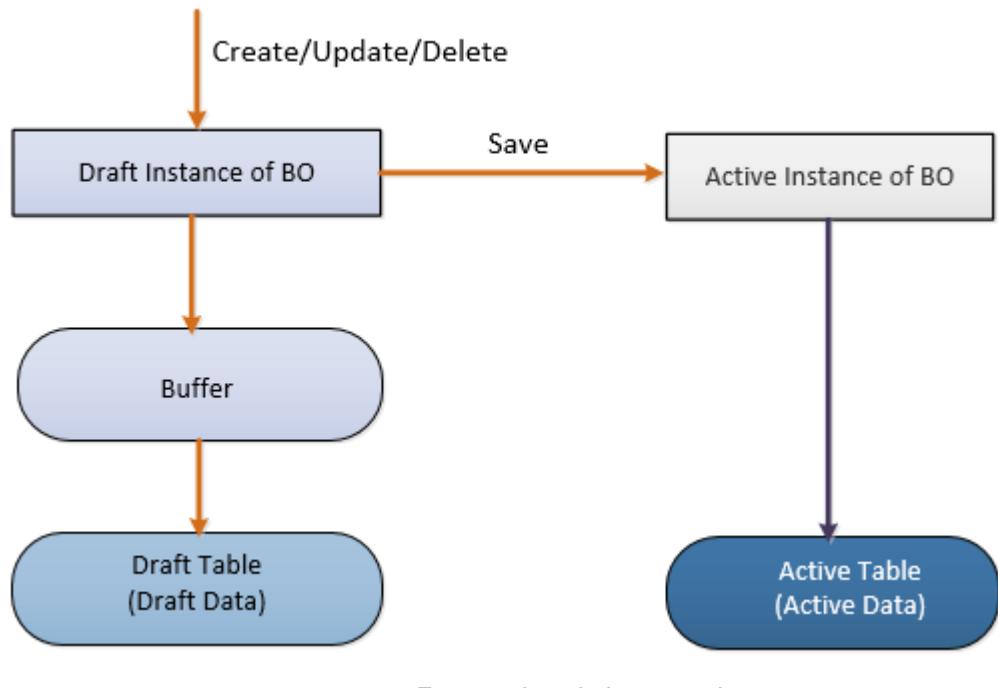
Draft Support

Up to now, SAP's transactional applications have been developed using stateful technologies, such as Floorplan Manager (FPM) for WebDynpro ABAP or the classic Dynpro technique. These stateful applications rely on a server session along with application buffers that can fulfill client requests (user interactions with multiple backend round trips) until the user has saved the data changes and finished his or her work.

Let us assume that you, as the application developer have to develop a completely new transactional app for SAP Fiori UI that also provides the draft data support. As the application developer, you may need to provide the end user with capabilities to store changed data at any time in the backend and proceed at a later point in time or to recover such data, even if the application client has crashed. However, this use case requires certain additional activities to enable the draft concept for your new app.

Such an app must always be able to deal with two versions of data:

- The **active data** that represents the state of the business entity (sales order) that is stored in the active persistence.
- The **draft data** that represents the transient state of a business entity until it is permanently stored in the persistence layer as active data. The draft data is stored in the draft persistence until its transition to the active data. The draft data is stored in the draft persistence until its transition to active data. As depicted in the figure below, a draft version of the business object plays a central role in this transition.



Preview

In the first version of our example app for sales order processing, the end user is able to list all sales orders available in the system and perform all CRUD operations on sales order instances. In addition, the resulting Fiori app provides draft-enabling capabilities.

Sales Order ID	Customer	Status
700000001	HEPA Tec (100000013)	New (N) >
700000004	Brazil Technologies (100000028)	Delivered (D) >
700000005	Indian IT Trading Company (100000039)	New (N) >
700000003 Draft	SAP (100000000)	In Progress (I) >
700000002 Draft	Talpa (100000003)	In Progress (I) >

Resulting Fiori app provides both transactional behavior and draft qualities

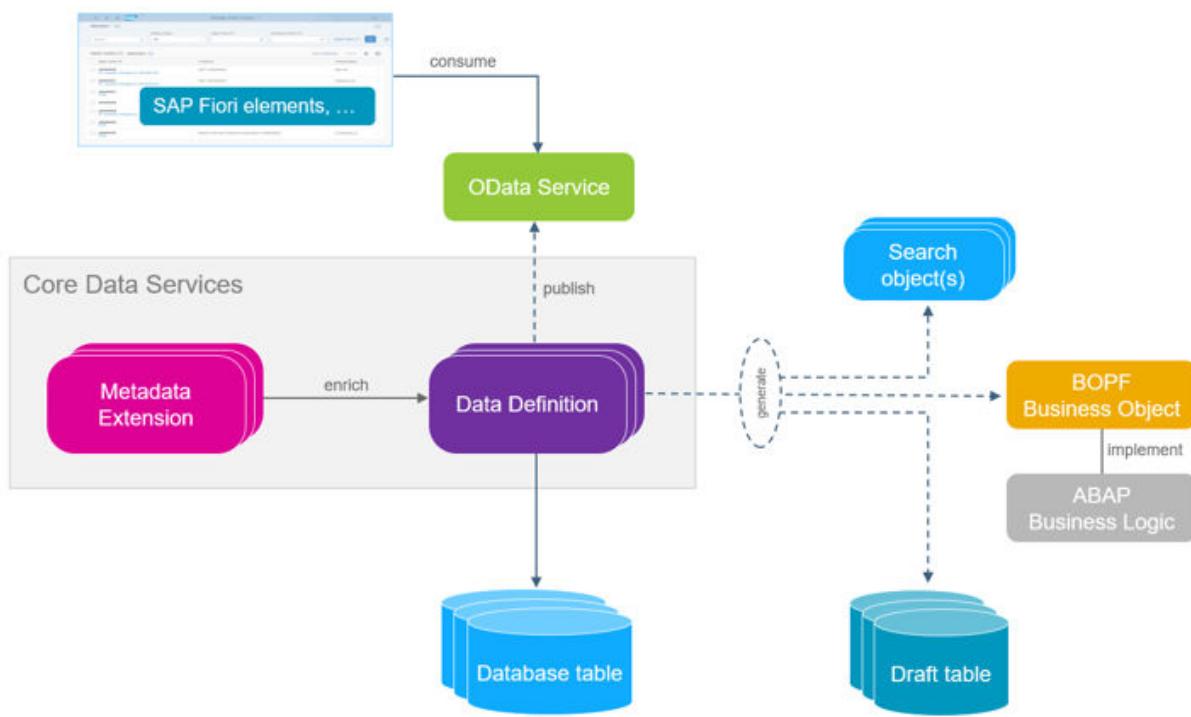
In the extended version of our example app, we will also enhance the simplified data model for processing sales order items. If the end user clicks on the resulting Fiori UI in edit mode, he or she has the option to create new items (including a draft version) and then edit the corresponding draft data.

Position	Product	Quantity	Total Amount
20	HT-1030	7	1,610.00
30	HT-1000	10	9,560.00

The extended Fiori app provides transactional behavior including draft qualities also at the level of sales order items

Involved Technologies and Development Objects

This new generation of applications enhances the general architecture of transactional applications through stateless communication with the ABAP back-end server. The figure below provides an overview of the main development objects and technologies involved when creating a transactional, draft-enabled Fiori apps based on the new ABAP programming model.



Frameworks and development objects involved when implementing transactional apps with draft capability

ABAP Core Data Services (CDS)

ABAP CDS provides a data modeling infrastructure for defining and consuming semantically rich data models in the ABAP platform starting with SAP NetWeaver AS for ABAP 7.4 SP05. CDS data models can be enhanced at the data model level using CDS view extensions and at the metadata level using CDS metadata extensions. ABAP CDS is a core technology within SAP S/4HANA and the ABAP development for SAP HANA in general.

[\[More...\]](#)

Metadata Extensions (MDE)

MDEs are used to define CDS annotations for a CDS view outside of the corresponding data definition. A CDS metadata extension is always assigned to a specific layer such as core, industry, partner or customer. The use of MDEs allows the separation of concerns by separating the data model from domain-specific semantics, such as UI-related information for Fiori elements. [\[More...\]](#) [\[page 185\]](#)

Business Object Processing Framework (BOPF)

BOPF is a framework that provides a set of generic services and functionalities to speed up, standardize, and modularize your development of business applications. BOPF manages the entire life cycle of your business objects and covers all aspects of your business application development. Instead of expending effort for developing an application infrastructure, the developer can focus on the individual business logic. Using BOPF provides you with the entire application infrastructure and integration of various components for free. This allows you to build applications rapidly on a stable and customer-proved infrastructure. [More:](#)

When using BOPF, you have various options to add application-specific business logic:

- BOPF determinations, which are used to calculate side-effects (automatically triggered by changes).
- BOPF validations, which are consistency checks that raise messages (automatically triggered by changes).
- BOPF actions, which are named operations that can be called (usually by a button on the UI).

SAP Fiori Elements

SAP Fiori elements (formerly known as smart templates) allow the automatic generation of SAP Fiori apps based on UI-specific annotations that can, for example, be specified in a CDS view or in locale annotations within the SAP Web IDE. The set of predefined domain-specific CDS annotations has been enhanced to include such UI annotations with ABAP 7.5x. The development of Fiori elements apps takes place in the SAP Web IDE.

[More...]

Activities Relevant to Developers

1. Defining the Draft Business Object for Sales Order [page 109]
2. Providing Additional Business Logic with a BOPF Determination [page 121]
3. Defining the Consumption Layer [page 124]
4. Adding UI Semantics for Consumption in Fiori UI [page 127]
5. Running the Resulting Fiori App [page 131]
6. Extending the Implementation for Adding New Sales Order Items [page 132]
7. Extending the Implementation with Value Validation [page 148]

8. Extending the Implementation for Adding New Sales Order Items [page 132]
9. Extending the Implementation with Value Validation [page 148]
10. Extending the Implementation with an Action [page 152]

5.3.1 Defining the Draft Business Object for Sales Order

As with any other business application, a set of database tables and other ABAP Dictionary objects defines the persistence model of a transaction application. To serve as a data source for a business object to be generated, a database table must be provided for each business object CDS view (also referred to as a business object views). Generally, business object views are normalized CDS views that define the hierarchy of entities by using associations and `@ObjectModel` annotations. A business object view is defined on top of the underlying database table or another CDS view and exposes all elements that are defined in the `SELECT` list, including the key elements that correspond to the primary keys of the underlying database table. In the transactional scenario of the programming model for SAP Fiori, the business object view is required for business object generation based on the BOPF framework.

i Note

In the context of draft-enablement, the concept of the business objects is (re)used to generate a draft entity based on the BOPF business object metadata model.

Activities Relevant to Developers

1. Modeling of Normalized Data Persistence [page 110]
2. Defining the CDS Data Model for the Draft Business Object [page 113]
3. Providing a Data Model for the Sales Order Header [page 116]

5.3.1.1 Modeling of Normalized Data Persistence

Sales order data can be distributed across multiple database tables: a table for the sales order (header data), a table for business partners, or a table for currency and status information. With the provision of normalized persistence, each individual database table is used to arrange data into logical groupings such that each one describes a small part of the whole. In the context of business applications, each table field corresponds to an attribute of the business object that is represented by the entire database table. Each table row represents a unique instance of that business object and must be different in some way from all other rows (that is, no duplicate rows are possible).

For our sample scenario, we will first create a suitable database table (`ZDEMO_SOH`) for sales order header data using ADT ABAP Dictionary tools and then use this as a data source for a new CDS-based data model.

Table ZDEMO_SOH for Sales Order Header (including administrative fields)

Field	Data Element
CLIENT (key) / (initial value)	MANDT
SALESORDERUUID (key) / (initial value)	SNWD_NODE_KEY
SALESORDER	SNWD_SO_ID
BUSINESSPARTNER	SNWD_PARTNER_ID
OVERALLSTATUS	SNWD_SO_OA_STATUS_CODE
CHANGEDAT	TIMESTAMPL
CHANGEDBY	UNAME
CREATEDBY	UNAME
CREATEDAT	TIMESTAMPL

→ Remember

The database table above also provides the basic administrative fields. They are used for administrative data, which usually covers the user that has created or last changed an instance and the corresponding timestamps. In particular, we will use the field `CHANGEDAT` for **ETag check**. In our draft scenario, the ETag check is used to determine whether two representations of a business entity, such as an active instance

and the corresponding draft BO instance, are the same. If the representation of the entity ever changes, a new and different ETag value is assigned.

ETags play a significant role in the lock lifetime when working with draft instances of business objects. For further information, see: [Draft Entities and Durable Locks \[page 46\]](#)

Sales order data can be distributed across multiple database tables: a table for the sales order (header data), a table for business partners, or a table for currency and status information. With the provision of normalized persistence, each individual database table is used to arrange data into logical groupings such that each one describes a small part of the whole. In the context of business applications, each table field corresponds to an attribute of the business object that is represented by the entire database table. Each table row represents a unique instance of that business object and must be different in some way from all other rows (that is, no duplicate rows are possible).

For our sample scenario, we will first create a suitable database table (`ZDEMO_SOH`) for sales order header data using ADT ABAP Dictionary tools and then use this as a data source for a new CDS-based data model.

Table ZDEMO_SOH for Sales Order Header (including administrative fields)

Field	Data Element
CLIENT (key) / (initial value)	MANDT
SALESORDERUUID (key) / (initial value)	SNWD_NODE_KEY
SALESORDER	SNWD_SO_ID
BUSINESSPARTNER	SNWD_PARTNER_ID
OVERALLSTATUS	SNWD_SO_OA_STATUS_CODE
CHANGEDAT	TIMESTAMPL
CHANGEDBY	UNAME
CREATEDBY	UNAME
CREATEDAT	TIMESTAMPL

Sales order data can be distributed across multiple database tables: a table for the sales order (header data), a table for business partners, or a table for currency and status information. With the provision of normalized persistence, each individual database table is used to arrange data into logical groupings such that each one describes a small part of the whole. In the context of business applications, each table field corresponds to an attribute of the business object that is represented by the entire database table. Each table row represents a unique instance of that business object and must be different in some way from all other rows (that is, no duplicate rows are possible).

For our sample scenario, we will first create a suitable database table (`ZDEMO_SOH`) for sales order header data using ADT ABAP Dictionary tools and then use this as a data source for a new CDS-based data model.

Table ZDEMO_SOH for Sales Order Header (including administrative fields)

Field	Data Element
CLIENT (key) / (initial value)	MANDT
SALESORDERUUID (key) / (initial value)	SNWD_NODE_KEY
SALESORDER	SNWD_SO_ID
BUSINESSPARTNER	SNWD_PARTNER_ID
OVERALLSTATUS	SNWD_SO_OA_STATUS_CODE
CHANGEDAT	TIMESTAMPL
CHANGEDBY	UNAME
CREATEDBY	UNAME
CREATEDAT	TIMESTAMPL

→ Remember

The database table above also provides the basic administrative fields. They are used for administrative data, which usually covers the user that has created or last changed an instance and the corresponding timestamps. In particular, we will use the field CHANGEDAT for **ETag check**. In our draft scenario, the ETag check is used to determine whether two representations of a business entity, such as an active instance and the corresponding draft BO instance, are the same. If the representation of the entity ever changes, a new and different ETag value is assigned.

ETags play a significant role in the lock lifetime when working with draft instances of business objects. For further information, see: [Draft Entities and Durable Locks \[page 46\]](#)

Starting Point and Prerequisites

To provide data persistence for our sample sales order scenario, we assume that you...

- Have the standard developer authorization profile to create ABAP development objects with [ABAP Development Tools](#).
- Reuse the data elements from the EPM sample data model (SNWD_*) when creating table fields for sales order (header) table.
- Access related business data from associated views or tables that originate in the EPM sample data model.

Procedure

Since SAP NetWeaver AS for ABAP 7.52 SP00, you can create and work with database tables in ABAP Development Tools (ADT) using the source-based editor.

1. Open the source-based ABAP Dictionary editor in ADT and create the database table `ZDEMO_SOH` with the fields as they are listed below. **For further information, see:**
2. Edit the database table as follows:

```
@EndUserText.label : 'Demo: SO header table'  
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE  
@AbapCatalog.tableCategory : #TRANSPARENT  
@AbapCatalog.deliveryClass : #L  
@AbapCatalog.dataMaintenance : #ALLOWED  
define table zdemo_soh {  
    key client          : mandt not null;  
    key salesorderuuid : snwd_node_key not null;  
    salesorder         : snwd_so_id;  
    businesspartner   : snwd_partner_id;  
    overallstatus     : snwd_so_oa_status_code;  
    changedat         : timestamp;  
    changedby         : uname;  
    createdby         : uname;  
    createdat         : timestamp;  
}
```

3. *Activate* the table.

Results

The table `ZDEMO_SOH` represents the active persistence for the draft sales order business object to be generated. Each table field corresponds to an attribute of the sales order business object represented by the entire table.

5.3.1.2 Defining the CDS Data Model for the Draft Business Object

The CDS Data Model

For didactic reasons, we will keep the data model as simple as possible. We will therefore reduce the number of business-relevant fields to a minimum set of view fields – as listed in the table below.

Buisness Object Entities	View Fields
Sales Order Header	SalesOrderUUID (key) SalesOrder BusinessPartner (association to BusinessPartner) OverallStatus
Business Partner	Business Partner

BO Semantics in the Data Model

Every BO view must include specific `@ObjectModel` annotations that indicate, for example, the draft persistence or the key of the given entity. The BO view of the root entity in particular requires certain additional view annotations in order to trigger automatic generation of a draft BO whose name corresponds to the respective BO view.

These annotations are:

Syntax: CDS annotations specific to transactional processing

```
@ObjectModel: {
    transactionalProcessingEnabled: true,
    compositionRoot: true, -- on root node only
    createEnabled:      true,
    deleteEnabled:     true,
    updateEnabled:     true,
    writeActivePersistence: '<DatabaseTable/View>',
    semanticKey: ['<SemanticKeyName>', ... ]
}
```

Sub Annotation of <code>@ObjectModel</code>	Effect
<code>transactionalProcessingEnabled: true</code>	Enables transactional runtime support
<code>compositionRoot: true</code>	Defines the root of the compositional hierarchy for the (draft) business object to be created
<code>createEnabled: true</code>	Allows you to create new business object instances in the active persistence
<code>deleteEnabled: true</code>	Allows you to delete business object instances in the active persistence

Sub Annotation of @ObjectModel	Effect
updateEnabled: true	Allows you to update existing business object instances in the active persistence
writeActivePersistence: '<database table/view>'	Specifies a database table or a database view for storing active data

Syntax: CDS annotations specific to draft enablement

```
@ObjectModel: {
    draftEnabled: true,
    writeDraftPersistence: '<DraftTable>'
}
```

Sub Annotation of @ObjectModel	Effect
draftEnabled: true	Allows you to create new business object instances
writeDraftPersistence: '<database_table>'	<p>Specifies the name of the database table for storing draft data</p> <p>This table is automatically generated by BOPF after activation of the CDS BO view.</p> <p>The draft table serves as a shadow table for intermediate storage.</p>

Syntax: CDS annotation used for ETag check

```
@ObjectModel: {
    entityChangeStateId: '<ChangedAt>'
}
```

Sub Annotation of @ObjectModel	Effect
entityChangeStateId: '<ChangedAt>'	<p>Refers to a single element that contains the change state of an active instance of a business object.</p> <p>Usually, elements like last changed timestamp, hash values, or version counters are used for ETag check.</p> <p>The ETag check is used to determine whether two representations of a business entity, such as an active instance and the corresponding draft BO instance, are the same. If the representation of the entity ever changes, a new and different ETag value is assigned.</p>

For further information, see: [ObjectModel Annotations \[page 428\]](#)

5.3.1.3 Providing a Data Model for the Sales Order Header

Starting Point and Prerequisites

For our sales order scenario, we assume that...

- The DDL-based data definition, as the corresponding development object, is already created in the ABAP package of your choice. In our case, the data definition and the business object view are named as `ZDEMO_I_SALESORDER_TP_D` and `ZDEMO_I_SalesOrder_TP_D` respectively.
- **For further information, see:** [Create a Data Definition for CDS View \[page 13\]](#)
- The persistence model is already defined. In particular, you have already created and activated the database table `ZDEMO_SOH` that serves as the data source for the business object view.

⚠ Caution

Whenever the table field names and the corresponding element names in the business object view are different (except for camel-case), this will cause problems when activating the view. For further information on how to avoid such an issue, see also: [Mapping CDS View-Elements onto Table Fields \[page 286\]](#)

- The key in the data model corresponds to the UUID-based primary key `SALESORDERUUID` of the underlying database table.

Implementing the Data Model

The following listing provides you with the implementation of our simple sales order data model with the database table `ZDEMO_SOH` defined as the data source for the corresponding CDS view `ZDEMO_I_SalesOrder_TP_D` (camel case notation!). The same table `ZDEMO_SOH` also serves as a storage location for changes to business object data that result from transactional behavior. Therefore, this table is specified in the `@ObjectModel.writeActivePersistence` annotation. For the draft persistence, the draft table `ZDEMO_SOH_D` is referenced in the `@ObjectModel.writeDraftPersistence` annotation.

To be able to access business data from other entities (business partner, overall status), corresponding associations are defined as part of the data model. These associations refer to CDS views that are already provided as part of the EPM data model.

This CDS view also provides an element `changedat` for ETag check. With the `@Semantics` sub-annotation `systemDateTime.lastChangedAt`, the corresponding value contains the date and time of the creation or last change that is recorded by the database.

```
@AbapCatalog.sqlViewName: 'ZDEMO_I_SOH_V'  
@AccessControl.authorizationCheck: #NOT_REQUIRED  
@EndUserText.label: 'Sales order for transactional app'  
@Search.searchable: true
```

```

@ObjectModel: {

    -- Annotations for transactional processing
    semanticKey: 'SalesOrder',
    compositionRoot: true,
    transactionalProcessingEnabled: true,
    createEnabled: true,
    deleteEnabled: true,
    updateEnabled: true,
    writeActivePersistence: 'ZDEMO_SOH',

    -- Additional annotations for draft enablement
    draftEnabled: true,
    writeDraftPersistence: 'ZDEMO_SOH_D',

    -- Additional ETag annotation (time stamp)
    entityChangeStateId: 'ChangedAt'
}

define view ZDEMO_I_SalesOrder_TP_D

    as select from zdemo_soh as SalesOrder -- the sales order table is the data
    source for this view

    /* Cross BO associations      */
    association [0..1] to SEPM_I_BusinessPartner           as _BusinessPartner
    on $projection.BusinessPartner = _BusinessPartner.BusinessPartner

    /* Associations for value help */
    association [0..1] to Sepm_I_SalesOrdOverallStatus    as _OverallStatus
    on $projection.OverallStatus   = _OverallStatus.SalesOrderOverallStatus

    {
        -- UUID-based key is required to enable draft capabilities
        @ObjectModel.readOnly: true
        key SalesOrder.salesorderuuid      as SalesOrderUUID,

        @Search.defaultSearchElement: true
        @ObjectModel.readOnly: true
        SalesOrder.salesorder          as SalesOrder,

        @ObjectModel.foreignKey.association: '_BusinessPartner'
        SalesOrder.businesspartner       as BusinessPartner,

        @Search.defaultSearchElement: true
        @ObjectModel.foreignKey.association: '_OverallStatus'
        SalesOrder.overallstatus        as OverallStatus,

        @Semantics.systemDateTime.lastChangedAt: true
        SalesOrder.changedat           as ChangedAt,

        @Semantics.systemDateTime.createdAt: true
        SalesOrder.createdat           as CreatedAt,
        @Semantics.user.createdBy: true
        SalesOrder.createdby            as CreatedBy,
        @Semantics.user.lastChangedBy: true
        SalesOrder.changedby            as ChangedBy,

        /* Associations */
        _BusinessPartner,
        _OverallStatus
    }
}

```

Activating the Data Definition

Once you have ensured that the syntax of the CDS view is complete and correct, you can activate the data definition (as the corresponding development object).

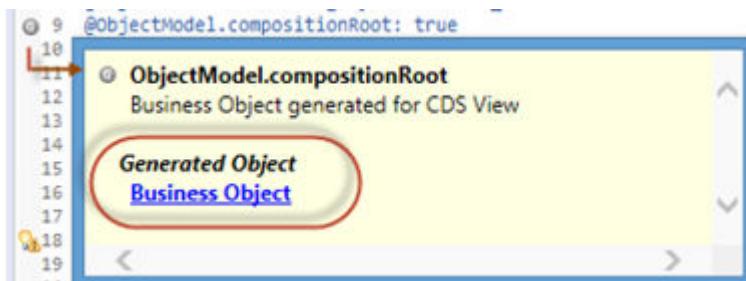
The activation procedure generates, the following new development objects are generated in your package:

- Draft table `ZDEMO_SOH_D` for the sales order header
- Draft BO, including all requisite ABAP Dictionary objects and other artifacts that define the new BO.

For more information about the generated business object, see [Checking the Generated Business Object \[page 118\]](#).

5.3.1.4 Checking the Generated Business Object

As a result of successful activation, the BOPF runtime creates a business object `ZDEMO_I_SALESORDER_TP_D` (upper-case notation!) with draft capabilities that is named after, and is stored in, the same package as the underlying CDS view, in our case: `ZDEMO_I_SalesOrder_TP_D` (camel case notation!). If you move the cursor over the decorator, an info screen informs you that the business object has been generated for the CDS view. In addition, the info screen provides you with a link to the business object editor.



Quick info screen after successful creation of the business object

On the entry page of the BO editor, you can identify the constants interface that has been generated together with the draft business object.

→ Remember

A *constants interface* is an ABAP interface that is dedicated to a specific business object. The interface includes constants for each business object's entity, such as nodes, attributes, and so on. We will use it later when we provide additional business logic using BOPF determinations, validations, or actions.

Business Object Overview

General Information

General Information of Business Object

Name: ZDEMO_I_SALESORDER_TP_D

Source CDS View: [ZDEMO_I_SALESORDER_TP_D](#)

Description:

Category: Draft Object

Constants Interface: * [ZIF_DEMO_I_SALESORDER_TP_D_C](#)



Regenerate

What's Next

Proceed to next steps

[Go to the nodes of this BO](#)

[Go to the ROOT node](#) 

Overview Nodes

Entry page of the BO editor with the generated draft BO for sales orders

If you now choose [Go to the ROOT node](#), you can view additional details at the entity level of the draft business object. You must know these when you proceed to use the BOPF API - for example, when implementing a determination. Here, you can find the names of the ABAP Dictionary objects (including the draft table) that have been generated together with the draft business object.

i Note

The transition of draft data to the permanent active data persistence is implemented using a BO-specific draft class. For our scenario (developing a new transactional app with draft capability), the generated **draft class comes with a default implementation**. As an application developer, you do not therefore need to provide any code to comply with the draft contract.

Node Overview

General Information

General Information of Node

Name: *	ZDEMO_I_SALESORDER_TP_D
Source CDS View:	ZDEMO_I_SALESORDER_TP_D
Description:	
Persistent Structure: *	ZSDEMO_I_SALESORDER_TP_D_D
Transient Structure:	
Combined Structure: *	ZSDEMO_I_SALESORDER_TP_D1
Combined Table Type: *	ZTDEMO_I_SALESORDER_TP_D
Database Table: *	ZDEMO_SOH_D — draft table

Create Enabled
 Update Enabled
 Delete Enabled

Text Node

Implementation Details

Node Implementation Details of Draft Business Object

Draft Class:	/BOBF/CL_LIB_DR — draft class with default implementation
--------------	---

Overview Alternative Keys Properties Associations Actions Determinations

BO editor – Generated ABAP Dictionary objects and the draft class with default implementation

As part of the business object generation, several ABAP Dictionary Objects are generated for each node. These structures are used during runtime.

Persistent Structure

The persistent structure contains fields that indicate the state of the corresponding entities.

salesorder	Elements of the corresponding CDS view.
businesspartner	
overallstatus	
activeuuid	Indicates the UUID of the active instance version.
hasactiveentity	If this field is flagged, the persistent structure relates to a draft which is created based on an active entry, whose UUID can be found in the field activeuuid.
include	Indicates the administration fields.
sdraft_write_draft_admin	

If this field is flagged, the persistent structure relates to the active entity.

For create requests, this field indicates if the values are written to the active database table or the draft database table.

Combined Structure

The combined structure contains structural information about the node and the persistent structure.

include /bobf/ s_frw_key_incl	Specifies the node relationships: key parent_key root_key
node_data	Persistent structure.

Database Table

The database table is the draft table, whose name is specified in the root CDS view. The table is generated on generation of the business object.

key salesordeuuid	Elements of the active database table.
salesorder	
businesspartner	
overallstatus	
activeuuid	Indicates the UUID of the active instance version.
hasactiveentity	Indicates if the draft data records has an active version in the active database table..
include sdraft_write_draft_admin	Indicates the administration fields.

5.3.2 Providing Additional Business Logic with a BOPF Determination

As you already know, the BOPF framework automatically includes the basic transactional operations (CRUD). Depending on the scenario, however, it may be necessary to provide additional application-specific business logic. For example, the following functionality is required for our sales order-processing scenario:

The end user should already have the possibility to create new sales orders based on the generated draft business object. To provide data consistency, we must therefore ensure that the application automatically calculates an ID for the sales order to be created.

We will implement this requirement using a BOPF determination, which allows us to calculate a new sales order ID at the level of the generated BO node. In a first step, we will therefore create a determination called

`CALC_SALESORDER_ID` for the sales order node, configure it appropriately for our use case, and finally, provide the implementation for this determination.

Preview

The end user has the option to create a new sales order and then edit the corresponding draft data. The system then calculates the sales order ID for the new (draft) version.

General Information

Sales Order ID:
700000001 ← Calculated Sales Order ID

Customer:

Status:

Creating a sales order instance

As you already know, the BOPF framework automatically includes the basic transactional operations (CRUD). Depending on the scenario, however, it may be necessary to provide additional application-specific business logic. For example, the following functionality is required for our sales order-processing scenario:

The end user should already have the possibility to create new sales orders based on the generated draft business object. To provide data consistency, we must therefore ensure that the application automatically calculates an ID for the sales order to be created.

We will implement this requirement using a BOPF determination, which allows us to calculate a new sales order ID at the level of the generated BO node. In a first step, we will therefore create a determination called `CALC_SALESORDER_ID` for the sales order node, configure it appropriately for our use case, and finally, provide the implementation for this determination.

Implementing the Determination `CALC_SALESORDER_ID`

EXECUTE Method

The `execute` method performs the determination at runtime.

To implement this method, add the source code from the listing below to the method `/bobf/if_frw_determination~execute`.

i Note

In the example source code listed below, we provide a relatively simple solution for generating a sales order IDs. In a real application scenario, bear in mind that you might retrieve such an ID from a **number range object** defined in our SAP system.

Listing: Implementing Determination to Calculate an ID for a New Sales Order

```
method /BOBF/IF_FRW_DETERMINATION~EXECUTE.

    "Find the highest used sales order number in both active and draft data
    WITH +both AS ( SELECT salesorder FROM zdemo_soh      "active data
                    UNION ALL
                    SELECT salesorder FROM zdemo_soh_d )           "draft data
    SELECT SINGLE
        FROM +both
        FIELDS MAX( salesorder ) AS salesorder
        INTO @DATA(lv_max_salesorder).

    "If there are no entries, set a start value
    IF lv_max_salesorder IS INITIAL.
        lv_max_salesorder = '0700000000'.
    ENDIF.

    "Read data with the given keys
    DATA lt_data TYPE ZTDEMO_I_SALESORDER_TP_D.

    io_read->retrieve(
        EXPORTING
            iv_node          = is_ctx-node_key      " uuid of node name
            it_key           = it_key              " keys given to the
determination
        IMPORTING
            eo_message       = eo_message         " pass message object
            et_data          = lt_data            " itab with node data
            et_failed_key    = et_failed_key     " pass failures
    ).

    "Assign numbers to each newly created line and tell BOPF about the
modification
    LOOP AT lt_data REFERENCE INTO DATA(lr_data).
        IF lr_data->salesorder IS INITIAL.
            ADD 1 TO lv_max_salesorder.
            lr_data->salesorder = lv_max_salesorder.

            " Fill leading zeros for ALPHANUM field on database
            lr_data->salesorder = |{ lr_data->salesorder alpha = IN }|.

        io_modify->update(
            EXPORTING
                iv_node          = is_ctx-node_key      " uuid of node
                iv_key           = lr_data->key        " key of line
                is_data          = lr_data            " ref to modified data
                it_changed_fields = VALUE
            #(( ZIF_DEMO_I_SALESORDER_TP_D_C=>sc_node_attribute-zdemo_i_salesorder_tp_d-
salesorder ) )
        ).
    ENDIF.
    ENDLOOP.

endmethod.
```

Related Information

[Determinations for CDS-Based Business Objects \[page 63\]](#)

5.3.3 Defining the Consumption Layer

A consumption CDS view (also referred to as consumption view) allows a business object view to be consumed in a different manner by different OData services. Each service-specific view is defined on top of a business object view and exposes its fields. A consumption view enhances the data model with metadata, additional associations, or even with transient fields that fit a given consumption use case.

The `@ObjectModel` annotations are also used in the consumption views to enable draft capability and the transactional-related aspects of the business data model.

→ Remember

According to the CDS rule, we have to double-maintain the `@ObjectModel` annotations that have the `VIEW` scope. In the CDS views, only the annotations with the `ELEMENT` and `ASSOCIATION` scope are inherited from the BO view.

Syntax: CDS annotations specific to transactional processing

```
@ObjectModel: {  
    transactionProcessingDelegated: true,  
  
    compositionRoot: true, -- on root composition node only  
  
    createEnabled:      true,  
    deleteEnabled:     true,  
    updateEnabled:     true  
}
```

Sub Annotation of <code>@ObjectModel</code>	Effect
<code>transactionProcessingDelegated : true</code>	Indicates that transactional requests to a consumption view are delegated to the transactional runtime of the underlying BO view (which is annotated with <code>@ObjectModel.transactionProcessingEnabled: true</code>)
<code>compositionRoot: true</code>	Defines the root of the compositional hierarchy
<code>createEnabled: true</code>	Allows a consumer to create new business object instances
<code>deleteEnabled: true</code>	Allows a consumer to delete business object instances

Sub Annotation of @ObjectModel	Effect
updateEnabled: true	Allows a consumer to update existing business object instances

Syntax: CDS annotations specific to draft enablement

```
@ObjectModel: {
    draftEnabled: true, -- on root consumption only
}
```

Sub Annotation of @ObjectModel	Effect
draftEnabled: true	Allows a consumer to create draft data for a business object instance

Activities Relevant to Developers

1. [Providing a Consumption View for the Sales Order \[page 125\]](#)
2. Activating the Data Definition with Consumption View
3. Activating the OData Service

5.3.3.1 Providing a Consumption View for the Sales Order

Based on the sales order BO view, we are now going to implement the data model for consumption with the help of the consumption view.

Prerequisites

For our sales order scenario, we assume that

- The DDL-based data definition, as the corresponding development object (without content), is already created. If you have not yet already done so, follow the steps from the topic . In our case, the DDL data definition and the CDS view are named `ZDEMO_C_SALESORDER_TP_D` and `ZDEMO_C_SalesOrder_TP_D` respectively.
- The business object CDS view that you implemented in the previous step serves as the data source for the consumption view.
- The **UUID-based key** in the data model in the consumption view corresponds to the primary key of the underlying database table.

Implementing the Data Model for Consumption

In this implementation, the business object view is the data source of the consumption view `ZDEMO_C_SalesOrder_TP_D`. The `@ObjectModel.transactionalProcessingDelegated: true` annotation indicates that transaction requests to the consumption view are delegated to the underlying business object view. To permit the use of metadata extensions, the `@Metadata.allowExtensions` annotation with the value `true` is added.

To expose this elementary data model composition and its metadata as an OData service, the consumption view is annotated with `@OData.publish: true` (auto-exposure option).

The SELECT list includes a set of fields that are relevant for consumption in a user interface (UI).

For this scenario, the key must be specified as the (technical) UUID-based key `SalesOrderUUID`.

```
@AbapCatalog.sqlViewName: 'ZDEMO_C_SOH_V'  
@AccessControl.authorizationCheck: #NOT_REQUIRED  
@EndUserText.label: 'Sales Order Header - Consumption View'  
@Search.searchable: true  
@Metadata.allowExtensions: true  
@ObjectModel: {  
    -- Annotations for transactional processing  
    semanticKey: 'SalesOrder',  
    compositionRoot: true,  
    transactionalProcessingDelegated: true,  
    createEnabled: true,  
    deleteEnabled: true,  
    updateEnabled: true,  
  
    -- Additional annotation for draft enablement  
    draftEnabled: true  
}  
@OData.publish: true  
  
define view ZDEMO_C_SalesOrder_TP_D  
    as select from ZDEMO_I_SalesOrder_TP_D as SalesOrder  
{  
    -- UUID-based key required  
    key SalesOrder.SalesOrderUUID,  
  
    @Search.defaultSearchElement: true  
    SalesOrder.SalesOrder,  
  
    SalesOrder.BusinessPartner,  
    @Search.defaultSearchElement: true  
    SalesOrder.OverallStatus,  
  
    /* Exposing associations */  
    SalesOrder._BusinessPartner,  
    SalesOrder._OverallStatus  
}
```

Activating the Data Definition of the Consumption View

After successful activation of the corresponding data definition development object, *ABAP Development Tools* generates several SAP Gateway artifacts that are stored in the back end of the *Application Server ABAP* and are required for OData service activation in the SAP Gateway hub. These artifacts include:

- The actual service artifact with the technical name <CDS_VIEW>_CDS. Available at SAP Gateway Business Suite Enablement - Service object (object type: R3TR IWSV).
- An SAP Gateway model (object type: R3TR IWMO) with the name <CDS_VIEW>_CDS.
- An SAP Gateway annotation model (object type R3TR IWVB) with the name <CDS_VIEW>_VAN
- An ABAP class ZCL_<CDS_VIEW> (in the case of local objects) or <Namespace>CL_<CDS_VIEW> (other objects) that is used to provide model metadata to the SAP Gateway service.

All generated service artifacts are automatically added to the same ABAP package as the underlying consumption CDS view.

Activating the OData Service

To make your data model ready for consumption, you only have to activate the resulting OData service in the SAP Gateway hub.

For further information, see: [Generating OData Service With Auto-Exposure \[page 19\]](#)

Service Catalog	
Type	Technical Service Name
BEP	ZDEMO_C SALESORDER TP D CDS

Activated OData service is listed in the Service Catalog of the SAP Gateway

→ Tip

To verify that the activated OData service works correctly, you can test the service. For further information, see: [Test the Activated OData Service \[page 26\]](#)

Related Information

[Exposing CDS Entities as OData Service \[page 339\]](#)

5.3.4 Adding UI Semantics for Consumption in Fiori UI

To separate the metadata specified in the annotations from the actual data definition in the view, the consumption views are not used to enhance the data model with UI-specific annotations for later consumption

in Fiori UIs. To add UI semantics, we actually use metadata extensions to annotate UI metadata for each consumption view. This enables UI-related metadata to be defined and transported independently of the data definition.

UI metadata annotations use the following syntax:

```
@Metadata.layer: #CORE | CUSTOMER | INDUSTRY | LOCALIZATION | PARTNER

@UI: {
  ...
}
annotate view <CDS_View> with
{
  @UI: {
    ...
  }
  <element_name>;
  ...
}
```

i Note

Before the `annotate view` statement, the `@Metadata.layer` annotation must be specified with an enumeration value in order to define the layer for the metadata extension. This layer determines the priority of the metadata if multiple metadata extensions exist for the same CDS view.

Related Topic: [ABAP CDS - ANNOTATE VIEW \(Syntax Reference\)](#)

Prerequisites

CDS (consumption) views are not extensible by default. To use a metadata extension for a view, you must consider the following condition:

The `@Metadata.allowExtensions` annotation must be added with the value `true` in the definition of the CDS (consumption) view. This annotation explicitly permits the use of metadata extensions.

Activities Relevant to Developers

1. [Creating a Metadata Extension \[page 129\]](#)
2. [Adding UI Metadata \[page 130\]](#)
3. Activating the Metadata Extension

5.3.4.1 Creating a Metadata Extension

Context

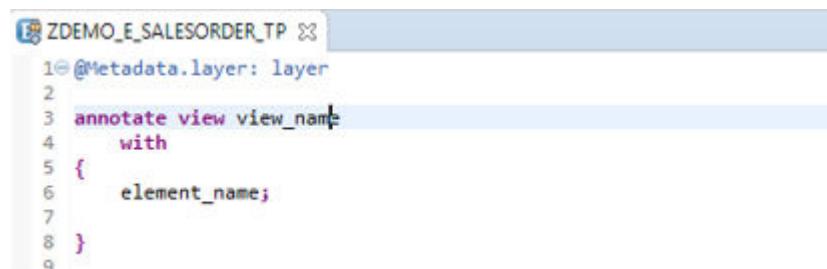
Metadata extensions enable you to write UI annotations for a CDS (consumption) view in a different development object in order to separate them from the CDS view. A metadata extension is a transportable ABAP development object.

Procedure

1. In your ABAP project, select the relevant package node in the *Project Explorer*.
2. Open the context menu and choose ► *New* > *Other ABAP Repository Object* > *Core Data Services* > *Metadata Extension* ▶ to launch the creation wizard.
3. Enter the name **ZDemo_E_SalesOrder_TP_D** and the description for the metadata extension to be created.
4. Choose *Next*.
5. Assign a transport request.
6. Choose *Finish*.

Results

In the package selected, the ABAP back-end system creates an inactive version of a metadata extension. You can now start to define metadata extensions.



The screenshot shows a code editor window with the title bar "ZDEMO_E_SALESORDER_TP". The code is a snippet of ABAP code:

```
1@Metadata.layer: layer
2
3 annotate view view_name
4   with
5 {
6   element_name;
7
8 }
```

Editing the metadata extensions

5.3.4.2 Adding UI Metadata

Annotating the Consumption View

Add the following source code to the metadata extension ZDemo_E_SalesOrder_TP_D:

Listing: Annotating consumption view with UI metadata

```
@Metadata.layer: #CUSTOMER

@UI: {
    headerInfo: {
        typeName: 'Sales Order',
        typeNamePlural: 'Sales Orders',
        title: { type: #STANDARD, value: 'SalesOrder' }
    }
}

annotate view ZDEMO_C_SalesOrder_TP_D with
{
    @UI.hidden: true
    SalesOrderUUID;

    @UI: {
        lineItem: [ { position: 10, label: 'Sales Order ID', importance: #HIGH } ],
        selectionField: [ { position: 10 } ],
        identification: [ { position: 10, label: 'Sales Order ID' } ]
    }
    SalesOrder;

    @UI: {
        lineItem: [ { position: 20, label: 'Customer', importance: #MEDIUM } ],
        identification: [ { position: 20, label: 'Customer' } ]
    }
    BusinessPartner;

    @UI: {
        lineItem: [ { position: 30, label: 'Status', importance: #MEDIUM } ],
        identification: [ { position: 30, label: 'Status' } ],
        selectionField: [ { position: 30 } ]
    }
    OverallStatus;
}
```

In the listing above, the value CUSTOMER is used for the metadata extension layer. This is the highest level. If several metadata extensions are provided for a data definition, the system considers the metadata extension with the highest value.

5.3.5 Running the Resulting Fiori App

To make sure that all parts of the application work correctly for the end user, we are now going to run the resulting application.

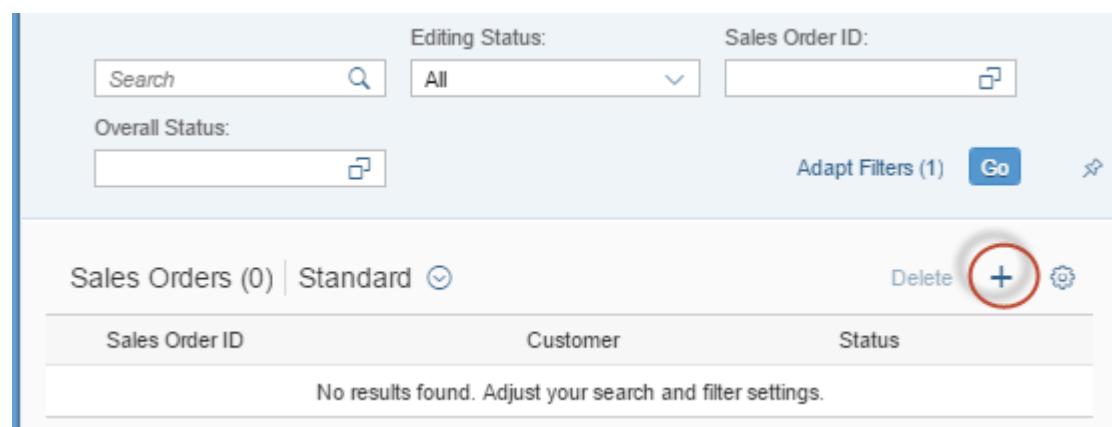
Prerequisites

For the corresponding OData service we assume that

- The consumption view ZDEMO_C_SalesOrder_TP_D is exposed as an OData service using the auto-exposure approach (@OData.publish: true).
For further information, see: [Generating OData Service With Auto-Exposure \[page 19\]](#)
- The resulting OData service has been activated in the SAP Gateway hub.
For further information, see: [Activate OData Service in the SAP Gateway Hub \[page 22\]](#)
- *Fiori elements* are used as building blocks during UI development.
For further information, see: [Consume Business Data Using SAP Fiori Elements \[page 27\]](#)

Procedure

If you run the new app based on *Fiori elements* in the *Fiori Launchpad*, the resulting UI provides you with a list of sales orders, including all fields that you exposed for consumption. In our case however, the initial screen with the list of sales orders is still empty.



Initial screen with an empty sales order list

To create a new sales order instance, choose the + icon and maintain all the fields required to specify the sales order header.

→ Remember

Each field that results from an association's path provides you with predefined value help options. **For further information, see:**

General Information

Sales Order ID:
700000001 ← Calculated SO ID

Customer:
100000013

Status:
N

Draft saved Save Cancel

Creating a sales order instance in the object page

If the changes to the sales order header fields have not been saved, a draft version of the sales order instance is created.

Sales Order ID	Customer	Status
700000001 Draft	HEPA Tec (100000013)	New (N) >

Draft version of the new sales order instance in the list report (after navigating back from the object page)

For the given sales order instance, the draft data is transferred to the active data only upon saving.

Sales Order ID	Customer	Status
700000001	HEPA Tec (100000013)	New (N) >

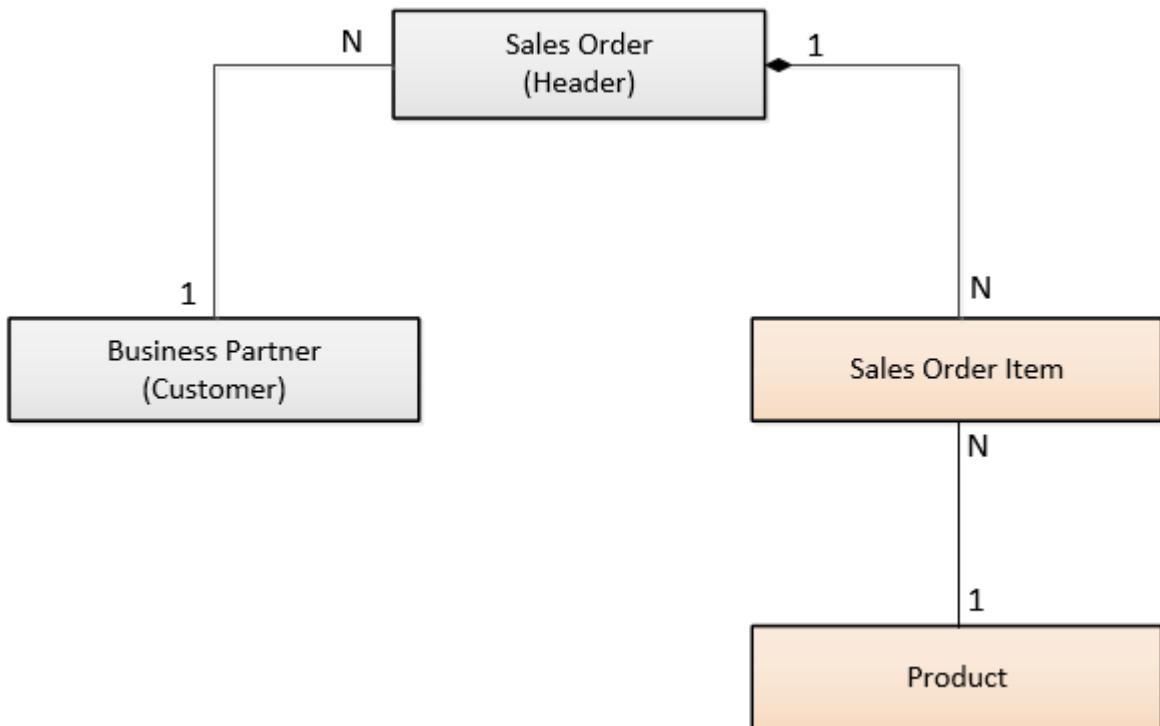
Active version of the new sales order instance

5.3.6 Extending the Implementation for Adding New Sales Order Items

Up to now, we have used a greatly simplified data model for our example scenario. In this chapter, however, we will set out a realistic scenario and use a two-level data model by adding sales order items. As a result, the sales order item will be added as a subnode of the sales order draft business object.

Extended Persistency and Data Model

The figure below shows the extended data model with the compositional hierarchy of the sales order header, the business partner, the sales order item, and the product. In this case, the sales order header represents the composition root of the data model.



Extended data model with associations and compositions

→ Remember

In draft scenarios, there might be two versions of target data for an association: active data and draft data. To ensure that the association not only reaches active data, it must be draft-enabled. Draft-enabled associations retrieve active data if you follow the association from an active instance and it retrieves draft data if you follow it from a draft instance.

As the association to Sales Order Item represents a composition of the business object, the association in the case under consideration is draft-enabled by default.

There are other cases, for example cross-BO associations, for which draft-enablement needs to be implemented manually.

For more information about associations in the draft context and how to draft-enable them, see: [Implementing Draft-Enabled Associations \[page 288\]](#).

Activities Relevant to Developers

1. [Extending the Definition of the Draft BO \[page 134\]](#)
2. [Extending the Implementation for Creating Sales Order Items \[page 138\]](#)
3. [Extending the Consumption Layer \[page 144\]](#)
4. [Adding UI Semantics for Sales Order Items \[page 146\]](#)
5. [Running the Extended Fiori App \[page 147\]](#)

5.3.6.1 Extending the Definition of the Draft BO

Modelling the Normalized Data Persistence for the Sales Order Item

Open the source-based ABAP Dictionary editor in ADT and create the database table ZDEMO_SOI with the fields listed below.

Related Topic: [Modeling of Normalized Data Persistence \[page 110\]](#)

Business-related fields of database table ZDEMO_SOI

Table Field	Data Element
CLIENT (key) / (initial value)	MANDT
SALESORDERITEMUUID (key) / (initial value)	SNWD_NODE_KEY
SALESORDERUUID (key) / (initial value)	SNWD_NODE_KEY
SALESORDERITEM	SNWD_SO_ITEM_POS
PRODUCT	SNWD_PRODUCT_ID
GROSSAMOUNT	SNWD_TTL_GROSS_AMOUNT
CURRENCYCODE	SNWD_CURR_CODE
QUANTITY	INT4

⚠ Caution

For performance reasons, we recommend defining the root UUID field (in our case, the salesorderuuid field) as a key field in the sales order items table.

Database table ZDEMO_SOI in the source-based editor of ADT:

```
@EndUserText.label : 'Sales Order Items table'  
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE  
@AbapCatalog.tableCategory : #TRANSPARENT  
@AbapCatalog.deliveryClass : #L  
@AbapCatalog.dataMaintenance : #ALLOWED  
define table zdemo_soi {  
    key client : mandt not null;  
    key salesorderitemuuid : snwd_node_key not null;  
    key salesorderuuid : snwd_node_key not null;  
    salesorderitem : snwd_so_item_pos;  
    product : snwd_product_id;  
    currencycode : snwd_curr_code;  
    @Semantics.amount.currencyCode : 'zdemo_soi.currencycode'  
    grossamount : snwd_ttl_gross_amount;  
    quantity : int4;
```

```
}
```

Modelling the CDS Data for the Sales Order Item

Create the data definition ZDEMO_I_SALESORDERITEM_TP_D and define a business object CDS view that exposes the fields listed below.

Related Topic: [Defining the CDS Data Model for the Draft Business Object \[page 113\]](#)

CDS view fields of the sales order item

Business Object Entities	View Fields
Sales Order Item	SalesOrderItemUUID (key)
	SalesOrderUUID
	SalesOrderItem
	Product (association to Product)
	GrossAmount
	CurrencyCode
	Qunatity
Product	Product

As you can see in the listing below, a set of associations is defined as part of the data model for a sales order item. Compositional associations define the view compositional hierarchy by providing an association to the different (root/parent) entities along the given composition path.

Listing: Business object view for sales order items

```
@AbapCatalog.sqlViewName: 'ZDEMO_I_SOI_V'
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Sales Order Items - subnode BO view'
@Search.searchable:      true
@ObjectModel: {
    semanticKey: [ 'SalesOrderItem' ],
    writeActivePersistence: 'ZDEMO_SOI',
    createEnabled: true,
    deleteEnabled: true,
    updateEnabled: true,
    writeDraftPersistence: 'ZDEMO_SOID'          -- Draft persistence
}
define view ZDEMO_I_SalesOrderItem_TP_D
    as select from zdemo_soi as SalesOrderItem
/* Compositional associations */
association [1..1] to ZDEMO_I_SalesOrder_TP_D   as _SalesOrder on
$projection.SalesOrderUUID = _SalesOrder.SalesOrderUUID
/* Cross BO associations */
association [0..1] to SEPM_I_Product_E           as _Product     on
$projection.Product = _Product.Product
/* Associations for value help */
association [0..1] to SEPM_I_Currency           as _Currency    on
$projection.CurrencyCode = _Currency.Currency
```

```
{
    @ObjectModel.readOnly: true
    key SalesOrderItem.salesorderitemuuid
    SalesOrderItemUUID,
        @ObjectModel.readOnly: true
        SalesOrderItem.salesorderuuid
        @Search.defaultSearchElement: true
        @ObjectModel.readOnly: true
        SalesOrderItem.salesorderitem
        @ObjectModel.foreignKey.association: '_Product'
        @ObjectModel.mandatory: true
        SalesOrderItem.product
        @ObjectModel.foreignKey.association: '_Currency'
        @Semantics.currencyCode: true
        @ObjectModel.readOnly: true
        SalesOrderItem.currencycode
        @Semantics.amount.currencyCode: 'CurrencyCode'
        @ObjectModel.readOnly: true
        SalesOrderItem.grossamount
        SalesOrderItem.quantity
        /* Exposed associations */
        @ObjectModel.association.type: [#TO_COMPOSITION_PARENT, #TO_COMPOSITION_ROOT]
        _SalesOrder,
        _Product,
        _Currency
}
```

→ Remember

CDS views that do not represent the root node of the hierarchy must have an association to their compositional parent view annotated with `#TO_COMPOSITION_PARENT`. In addition, they must have a `#TO_COMPOSITION_ROOT` association for performance reasons. However, the difference between the two association types is effective only from the third level of data model hierarchy.

Adding the Association from Sales Order to Sales Order Item

In the root BO view `ZDEMO_I_SalesOrder_TP_D`, add the compositional association `_Item` that corresponds to the listing below. The specified association must also be annotated appropriately in the projection by using the `@ObjectModel.association.type` annotation with the appropriate value: `#TO_COMPOSITION_CHILD`. This annotation defines which associated BO views are to be created as subnodes of the draft BO.

Related Topic: [Modeling of Normalized Data Persistence \[page 110\]](#)

Listing: Business object view for sales orders

```
...
define view ZDEMO_I_SalesOrder_TP_D
...
/* Compositional associations */

association [0..*] to ZDEMO_I_SalesOrderItem_TP_D as _Item on
$projection.SalesOrderUUID = _Item.SalesOrderUUID
...
/* Exposed associations */
@ObjectModel.association.type: [#TO_COMPOSITION_CHILD]
```

```
_Item,  
...  
}
```

Activating the Data Definitions for both BO Views

In this activation step, both the draft persistence and the draft BO are (re)generated based on the new data model.

→ Tip

The previously inactive data definition for the SO item view (`ZDEMO_I_SalesOrderItem_TP_D`) is referenced in the SO root view (`ZDEMO_I_SalesOrder_TP_D`). This causes the corresponding code line to issue an error in the root view. However, this error can be fixed by activating both objects in a single step.

Procedure

1. In the *Project Explorer* tree of your ABAP project, use multiple selection to select both relevant data definitions `ZDEMO_I_SALESORDER_TP_D` and `ZDEMO_I_SALESORDERITEM_TP_D`.
2. Choose *Activate* from the context menu or the toolbar.

Results

As a result of this activation procedure, the extended draft BO is generated.

Checking the Regenerated BO

The screenshot shows the SAP Fiori Node Overview screen for a draft Business Object (BO). The 'General Information' section includes fields for Name, Source CDS View, Description, Persistent Structure, Transient Structure, Combined Structure, Combined Table Type, and Database Table. The 'Database Table:' field is highlighted in yellow and contains 'ZDEMO_SOID' with a note '← draft table for SO items'. Other fields like 'Name:' and 'Source CDS View:' also have notes indicating they are part of the draft table. The 'Implementation Details' section shows the Draft Class field. A navigation bar at the bottom includes tabs for Overview, Alternative Keys, Properties, Associations, Actions, Determinations, Validations, and Authorization.

Added item subnode in the draft BO

5.3.6.2 Extending the Implementation for Creating Sales Order Items

In this topic, we want to extend the metadata model of the generated draft BO to provide additional application logic. Now, the end user should be able to add new items to a sales order header.

Preview

If the end user clicks on the Fiori UI in the edit mode, he or she has option to create a new item (as a draft) and then edit the corresponding draft data. The user should also be able to select and then delete individual items.

Adding sales order items in change mode

Not all fields of the sales order draft item are initially populated with data. In our case, only the product and the quantity fields are to be specified by the user.

When the user chooses the **Add** button, the data for the new item is generated as a draft version.

Editable fields for a new sales order item

The position of the new item is calculated using the existing item positions, whereas the amount data are determined on the basis of the table records.

700000005

General Information		Second Facet		Determination CALC_AMOUNT is used to calculate total amount		
Position	Product	Quantity		Total Amount		
<input type="radio"/> 20	HT-1030	<input type="button" value=""/>	7	1,610.00	EU	<input type="button" value=""/>

Draft saved

Draft version of the new item

For the given sales order instance, the draft data of the new item is transferred to the active data only upon saving.

Position	Product	Total Amount	Quantity	
10	Smart Office (HT-1100)	320.94 USD	3.000 EA	<input type="button" value=""/>
20	Smart Design (HT-1101)	285.24 EUR	3.000 EA	<input type="button" value=""/>
30	Smart Network (HT-1102)	82.11 USD	1.000 EA	<input type="button" value=""/>
40	Smart Multimedia (HT-1103)	183.26 EUR	2.000 EA	<input type="button" value=""/>
50	Notebook Basic 15 (HT-1000)	7,963.48 EUR	7.000 EA	<input type="button" value=""/>

Active version of the new item

Development Steps

To implement the missing calculations for the item position and the total amount of an item, you must first create corresponding determinations (`GET_ITEM_POSITION` and `CALC_AMOUNT`) at the level of the sales order item subnode, configure them by means of appropriate trigger conditions and finally implement the determination class.

Step 1: Create and Configure the Determination `GET_ITEM_POSITION` for the Sales Order Item Subnode

Related Topic: [Providing Additional Business Logic with a BOPF Determination \[page 121\]](#)

General Information

General Information of Determination

Name:*	GET_ITEM_POSITION
Description:	Calculates position for new SO item
Implementation Class:*	ZCL_DEMO_D_GET_ITEM_POSITION
Category:	React after modification

Triggers

Information on Triggers

New	Delete				
type filter text					
Node	A.	Create	Update	Del...	Lo...
ZDEMO_I_SALESORDERITEM_TP_D		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Configuring the determination GET_ITEM_POSITION

Step 2: Implement the Determination GET_ITEM_POSITION

Listing: EXECUTE method of the determination implements the calculation of item positions

```

CLASS ZCL_DEMO_D_GET_ITEM_POSITION IMPLEMENTATION.
  method /BOBF/IF_FRW_DETERMINATION~EXECUTE.
    " Find the highest used item number in both active and draft data (draft
    table)
    WITH +both AS (  SELECT salesorderitem FROM ZDEMO_SOI
      UNION ALL
      SELECT salesorderitem FROM ZDEMO_SOID )
      SELECT SINGLE
      FROM +both
      FIELDS MAX( salesorderitem ) AS salesorderitem
    INTO @DATA(lv_max_salesorderitem).
    " If there are no entries, set a start value
    IF lv_max_salesorderitem IS INITIAL.
      lv_max_salesorderitem = '0000000000'.
    ENDIF.
    "Read data with the given keys (typed with combined type table)
    DATA lt_data TYPE ztdemo_i_salesorderitem_tp_d.
    io_read->retrieve(
      EXPORTING
        iv_node                  = is_ctx-node_key    " uuid of node name
        it_key                   = it_key            " keys given to the
determination
      IMPORTING
        eo_message               = eo_message        " pass message object
        et_data                  = lt_data          " itab with node data
        et_failed_key            = et_failed_key   " pass failures
    ).
  
```

```

" Assign numbers to each newly created item and trigger the modification in
BOPF
LOOP AT lt_data REFERENCE INTO DATA(lr_data).
  IF lr_data->salesorderitem IS INITIAL.
    ADD 10 TO lv_max_salesorderitem.
    lr_data->salesorderitem = lv_max_salesorderitem.
    io_modify->update(
      EXPORTING
        iv_node          = is_ctx-node_key      " uuid of node
        iv_key           = lr_data->key        " key of line
        is_data          = lr_data            " ref to modified data
        it_changed_fields = VALUE
    #( ( zif_demo_i_salesorder_tp_d_c=>sc_node_attribute-zdemo_i_salesorderitem_tp_d-
salesorderitem ) )
    ).
  ENDIF.
ENDLOOP.
endmethod.
ENDCLASS.

```

Step 3: Create and Configure the Determination CALC_AMOUNT for the Sales Order Item Subnode

Related Topic: [Providing Additional Business Logic with a BOPF Determination \[page 121\]](#)

The screenshot shows the SAP Fiori Determination Overview page for the determination `ZDEMO_I_SALESORDERITEM_TP_D`. The page is divided into sections: General Information, Triggers, and a table of nodes.

General Information:

- Name: `CALC_AMOUNT`
- Description: `Calculates amount for a SO item`
- Implementation Class: `ZCL_DEMO_D_CALC_AMOUNT`
- Category: `React after modification` (highlighted in yellow)

Triggers:

- New
- Delete

Table of Nodes:

Node	A..	Create	Update	Del...	Load	Det...
<code>ZDEMO_I_SALESORDERITEM_TP_D</code>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Configuring the determination CALC_AMOUNT

Step 4: Implement the Determination CALC_AMOUNT

Listing: EXECUTE method of the determination is used to retrieve and calculate the total amount for an item

```
CLASS ZCL_DEMO_D_CALC_AMOUNT IMPLEMENTATION.
method /BOBF/IF_FRW_DETERMINATION~EXECUTE.
  " Read data with the given keys (typed with combined type table)
  DATA lt_item TYPE ztdemo_i_salesorderitem_tp_d.
  io_read->retrieve(
    EXPORTING
      iv_node          = is_ctx-node_key
      it_key           = it_key
    IMPORTING
      eo_message       = eo_message
      et_data          = lt_item
      et_failed_key   = et_failed_key
  ).
  " Read price-relevant product data from EPM product table SNWD_PD
  SELECT FROM snwd_pd FIELDS
    product_id AS product,
    price,
    currency_code
    FOR ALL ENTRIES IN @lt_item
    WHERE product_id = @lt_item-product
    INTO TABLE @DATA(lt_prices).
  SORT lt_prices BY product.
  " Calculate amount and update if needed.
  LOOP AT lt_item REFERENCE INTO DATA(lr_item).
    IF lr_item->product IS NOT INITIAL AND lr_item->quantity > 0.
      READ TABLE lt_prices WITH KEY product = lr_item->product ASSIGNING FIELD-
      SYMBOL(<ls_price>) BINARY SEARCH.
      IF sy-subrc = 0.
        DATA(lv_new_amount) = lr_item->quantity * <ls_price>-price.
        IF lv_new_amount <> lr_item->grossamount.
          lr_item->grossamount = lv_new_amount.
          lr_item->currencycode = <ls_price>-currency_code.
          io_modify->update(
            EXPORTING
              iv_node          = is_ctx-node_key
              iv_key           = lr_item->key
              is_data          = lr_item
              it_changed_fields = VALUE
#( ( zif_demo_i_salesorder_tp_d_c=>sc_node_attribute-zdemo_i_salesorderitem_tp_d-
      grossamount )
      ( zif_demo_i_salesorder_tp_d_c=>sc_node_attribute-
      zdemo_i_salesorderitem_tp_d-currencycode ) )
          .
        ENDIF.
      ENDIF.
    ENDIF.
  ENDLOOP.
endmethod.
ENDCLASS.
```

5.3.6.3 Extending the Consumption Layer

Based on the BO view for the sales order items, we are now going to implement the data model for consumption with the help of the CDS consumption view.

Define the Consumption View for the Sales Order Items

Create a DDL-based data definition with the name `ZDEMO_C_SALESORDERITEM_TP_D` as the required development object and define the CDS view `ZDEMO_C_SalesOrder_TP_D` in accordance with the listing below.

Related Topic: [Defining the Consumption Layer \[page 124\]](#)

Listing: Consumption view for sales order items

```
@AbapCatalog.sqlViewName: 'ZDEMO_C_SOI_V'
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Sales Order Item - Consumption View'
@Metadata.allowExtensions: true
@Search.searchable: true
@ObjectModel: {
    semanticKey: ['SalesOrderItem'],
    createEnabled: true,
    deleteEnabled: true,
    updateEnabled: true
}
define view ZDEMO_C_SalesOrderItem_TP_D
    as select from ZDEMO_I_SalesOrderItem_TP_D as SalesOrderItem
    /* Composition and cross BO associations */
    association [1..1] to ZDEMO_C_SalesOrder_TP_D as _SalesOrder on
        $projection.SalesOrderUUID = _SalesOrder.SalesOrderUUID
    association [0..1] to SEPM_I_Product_E as _Product on
        $projection.Product = _Product.Product
    /* Associations for value help */
    association [0..1] to SEPM_I_Currency as _Currency on
        $projection.CurrencyCode = _Currency.Currency
{
    @ObjectModel.readOnly: true
    key SalesOrderItem.SalesOrderItemUUID,
    @ObjectModel.readOnly: true
    SalesOrderItem.SalesOrderUUID,
    @Search.defaultSearchElement: true
    _SalesOrder.SalesOrder,
    SalesOrderItem.SalesOrderItem,
    SalesOrderItem.Product,
```

```

        SalesOrderItem.CurrencyCode,
        SalesOrderItem.GrossAmount,
        SalesOrderItem.Quantity,
        /* Exposed associations */
        @ObjectModel.association.type: [ #TO_COMPOSITION_ROOT,
#TO_COMPOSITION_PARENT ]
        _SalesOrder,
        _Product,
        _Currency
    }
}

```

Add the Association from Sales Order to Sales Order Item

Related Topic: [Providing a Consumption View for the Sales Order \[page 125\]](#)

Listing: Consumption view for sales orders

```

...
define view ZDEMO_C_SalesOrder_TP_D

...
/* Composition and cross BO associations */
    association [0..*] to ZDEMO_C_SalesOrderItem_TP_D as _Item on
    _Item.SalesOrderUUID = SalesOrder.SalesOrderUUID

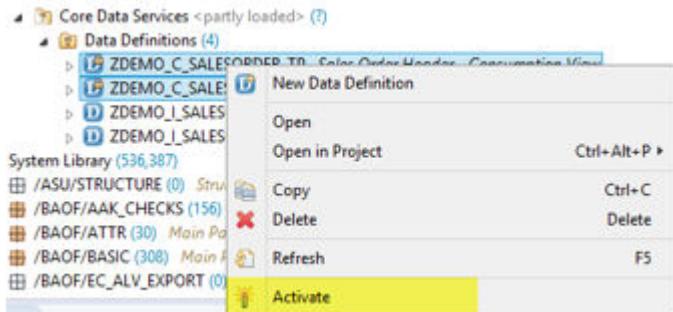
{
    ...
    ...
    /* Exposing value via associations */
    @ObjectModel.association.type: [ #TO_COMPOSITION_CHILD ]
    _Item,
    ...
}

```

Activate the Data Definitions for both Consumption Views

Procedure

1. In the *Project Explorer* tree of your ABAP project, use multiple selection to select both relevant data definitions *ZDEMO_C_SALESORDER_TP_D* and *ZDEMO_C_SALESORDERITEM_TP_D*.
2. Choose [Activate](#) from the context menu or the toolbar.



Activating multiple data definitions

5.3.6.4 Adding UI Semantics for Sales Order Items

In this step, we make use of a metadata extension to annotate UI metadata of the consumption view for the sales order item.

Prerequisites

CDS (consumption) views are not extensible by default. To use a metadata extension for a (consumption) view, you must consider the following condition:

The `@Metadata.allowExtensions` annotation must be added with the value `true` in the definition of the CDS (consumption) view. This annotation explicitly permits the use of metadata extensions.

Related Topic: [Adding UI Semantics for Consumption in Fiori UI \[page 127\]](#)

Creating an Additional Metadata Extension

Create a metadata extension with the name `ZDEMO_E_SALESORDERITEM_TP_D` as the required transportable ABAP development object and add the UI metadata from the listing below.

Related Topic: [Creating a Metadata Extension \[page 129\]](#)

Adding UI Metadata

Listing: Annotating the Consumption View for Sales Order Items

```

@Metadata.layer: #CUSTOMER

@UI: {
    headerInfo: {
        typeName: 'Sales Order Item',

```

```

        typeNamePlural: 'Sales Order Items',
        title: { type: #STANDARD, value: 'SalesOrderItem' }
    }
}

annotate view ZDEMO_C_SalesOrderItem_TP_D with
{
    @UI.hidden: true
    SalesOrderItemUUID;

    @UI.hidden: true
    SalesOrderUUID;

    @UI.hidden: true
    SalesOrder;

    @UI: {
        lineItem: [ { position: 10, label: 'Position', importance: #HIGH } ],
        identification:[ { position: 10, label: 'Position' } ]
    }
    SalesOrderItem;

    @UI: {
        lineItem: [ { position: 20, label: 'Product', importance: #MEDIUM } ],
        identification:[ { position: 20, label: 'Product' } ]
    }
    Product;

    @UI: {
        lineItem: [ { position: 30, importance: #MEDIUM, label: 'Quantity' } ],
        identification:[ { position: 30, label: 'Quantity' } ]
    }
    Quantity;

    @UI.hidden: true
    CurrencyCode;

    @UI: {
        lineItem: [ { position: 40, importance: #MEDIUM } ],
        identification:[ { position: 40 } ]
    }
    GrossAmount;
}

```

Activating the Metadata Extension

To make the UI metadata effective for the OData service at runtime, you must activate the corresponding development object (metadata extension [ZDEMO_E_SALESORDERITEM_TP_D](#)).

5.3.6.5 Running the Extended Fiori App

To make sure that the extended parts of the application also work correctly for the end user, we are now going to run the resulting app based on Fiori elements as building blocks.

For this purpose, we must create a new [List Report Application](#) project in the Web IDE.

For further information, see: : [Create a Project for a Fiori App in the Web IDE \[page 28\]](#)

New List Report Application

Template Customization

Data Binding

OData Collection* ZDEMO_C_SalesOrder_TP_D

OData Navigation to_Item

Defining the navigation path from SO header to SO item when creating a WEB IDE project

5.3.7 Extending the Implementation with Value Validation

In this topic, we extend the current implementation of our sample scenario to include a consistency validation for the draft data when editing sales order items. This consistency validation checks specifically if the quantity data of a product is valid. If it is not, the consistency validation prevents the draft data from being saved as active data version. This ensures that only consistent sales order item instances can be saved.

Preview

If the end user clicks on the Fiori UI in edit mode, he or she has the option to create a new item (as a draft) and then edit the corresponding draft data. The value entered in the quantity field must be checked for validity. After the end user presses the **ENTER** key or clicks into another input field, the system recognizes negative values as faulty and marks the quantity field accordingly in the Fiori UI. In addition, a corresponding error message is displayed to the end user.

The screenshot shows a form titled "General Information". It has fields for "Position" (30), "Product" (HT-1001), and "Quantity" (-7). A red dashed border highlights the quantity input field, and a tooltip says "The number is negative: 7-". Below the quantity field is a price field with value "0.00". At the bottom, there's a message bar with "Draft saved" and buttons for "Add" and "Cancel".

Value validation of the quantity field triggered by ENTER

Implementation Steps – Overview

We will create a BOPF validation at the level of the sales order item subnode, configure it by means of an appropriate trigger condition and then implement the consistency validation interface to check whether the entered quantity values are positive.

→ Remember

In BOPF, a validation is an entity of a business object node that is triggered in certain situations to check various aspects of a given set of node instances. In particular, consistency validations check if a node instance is consistent under consideration of the consistency criteria imposed by the business requirements. Such validations are called at pre-defined points within the BOPF BO transaction cycle to ensure that BO nodes are persisted in a consistent state. These points can be configured for consistency validations. Each such validation configuration contains a trigger condition that is checked by BOPF at several time points during the transaction. If the trigger condition is fulfilled, the consistency validation is executed. Otherwise, if there are inconsistent node instances, a consistency validation sends messages to the consumer and prevents the transaction from being saved until the inconsistency is corrected.

Creating and Configuring the Validation for the Sales Order Item Subnode

Procedure

- To create a BOPF validation for the sales order item subnode, open the BO editor with the associated item node (in our case: `ZDEMO_I_SALESORDERITEM_TP_D`) and choose the *Validations* tab.
- Choose *New...* to launch the *New Validation* wizard.
- Specify the details:
 - Name:* `POSITIVE_QUANTITY_VAL`
 - Description:* Enter a short description that describes the purpose of the validation.

- *Implementation Class*: Accept the name of the determination class that is suggested by the wizard.
 - *Validation Category*: Select *Consistency Check*.
4. Choose *Finish* to complete the creation procedure.
5. To check the triggers for the validation, select the *POSITIVE_QUANTITY_VAL* and choose **CTRL** + **click** to open the *Validation Overview* screen.

Validation Overview

General Information

General Information of Validation

Name:*	POSITIVE_QUANTITY_VAL
Description:	Check if quantity value >=0
Implementation Class:*	ZCL_DEMO_V_POSITIVE_QUANTITY_V
Category:	Consistency Check

Triggers

Information on Triggers

New Delete

Node	Assoc...	Create	Update	Delete
ZDEMO_I_SALESORDERITEM_TP_D		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

The validation POSITIVE_QUANTITY_VAL

6. Leave the predefined trigger condition unchanged and save the validation entries.
7. *Activate* the draft business object.

Results

The BOPF framework assigns the new validation to the business object subnode for sales order items and creates a validation class that implements the BOPF validation interface /BOBF/IF_FRW_VALIDATION.

ZDEMO_I_SALESORDER_TP_D ▶ ZDEMO_I_SALESORDER_TP_D ▶ ZDEMO_I_SALESORDERITEM_TP_D ▶

Validations

New... Delete

Name	Implementation Class	Category	Triggers
POSITIVE_QUANTITY_VAL	ZCL_DEMO_V_POSITIVE_QUANTITY_V	Consistency Check	Triggers configured

Ctrl+click to open

Added consistency check at the item level

Implementing the Validation POSITIVE_QUANTITY_VAL

The execute method performs the validation at runtime.

To implement this method, add the source code from the listing below to the method /bobf/if_frw_validation~execute. This method implements the main validation logic. It does not provide any modifying access to the node's instance data but returns the messages and the keys of failed node instances (instances for which the save action may not be executed). For each sales order item (remember that we make use of "mass-enabled" implementation), we check if the value <s_sales_order_item>-quantity is negative.

Listing: EXECUTE method for validation

```
METHOD /BOBF/IF_FRW_VALIDATION~EXECUTE.

  " Typed with node's combined table type
  DATA lt_sales_order_item TYPE ztdemo_i_salesorderitem_tp_d.

  " Retrieve the data of the requested node instance
  io_read->retrieve(
    EXPORTING
      iv_node      = is_ctx-node_key
      it_key       = it_key
    IMPORTING
      et_data      = lt_sales_order_item
      eo_message   = eo_message
      et_failed_key = et_failed_key
  ).

  LOOP AT lt_sales_order_item ASSIGNING FIELD-SYMBOL(<s_sales_order_item>).
    IF <s_sales_order_item>-quantity < 0.
      IF <s_sales_order_item>-isactiveentity = abap_false.

        DATA(lv_lifetime) = /bobf/cm_frw=>co_lifetime_state. "draft
        ELSE.
          lv_lifetime = /bobf/cm_frw=>co_lifetime_transition. "active
        ENDIF.
        eo_message = /bobf/cl_frw_factory=>get_message( ).
        eo_message->add_message(
          EXPORTING is_msg = VALUE #( msgid = 'CM_SEPMRA_SALESORDER' " example
            msgno = 5
            msgv1 = 'The number is negative: ' ##no_text
            msgv2 = <s_sales_order_item>-quantity
            msgty = /bobf/cm_frw=>CO_SEVERITY_ERROR
          )
          iv_node = is_ctx-node_key
          iv_key = <s_sales_order_item>-key
          iv_attribute = zif_demo_i_salesorder_tp_d_c=>sc_node_attribute-
zdemo_i_salesorderitem_tp_d-quantity
          iv_lifetime = lv_lifetime
        ).

        APPEND VALUE #( key = <s_sales_order_item>-key ) TO et_failed_key.

      ENDIF.
    ENDLOOP.

  ENDMETHOD.
```

Related Information

[Validations for CDS-Based Business Objects \[page 58\]](#)

5.3.8 Extending the Implementation with an Action

You may wish to extend the current scenario to include a further step. For example, you may be interested in extending the business logic so that status changes to individual instances of the sales order can be executed without the end user having to switch to edit mode. In this case, the BOPF actions come in to play.

We have already demonstrated the corresponding development steps in another context. (**For further information, see:** [Extending Apps with Quick Actions \[page 96\]](#))

i Note

The general steps listed below remain valid also in the draft context and can easily be applied to our example scenario. Be aware that the example code must be modified at some points there to match the data model of this scenario.

Activities Relevant to Developers

1. [Adding a New BOPF Action \[page 96\]](#)
2. [Implementing the Action \[page 98\]](#)
3. [Enabling Actions for OData Consumption \[page 101\]](#)

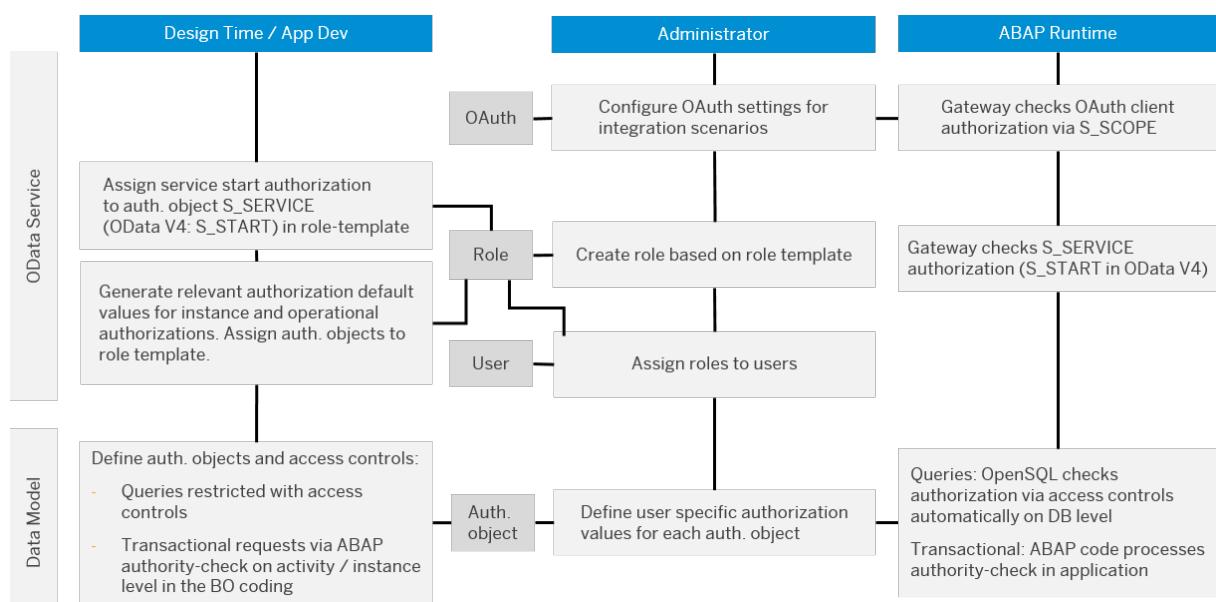
→ Tip

If you have arrived at this point, you might consider extending the present example scenario to include some further interesting features of the programming model. For example, you could enable or disable the action SET_TO_PAID depending on the status of the sales order. To provide such dynamic action control to the data model, you must implement the property determination class of the Sales Order BO.

For more detailed information on how you can provide such dynamic control, refer to topic: [Dynamic Action Control \[page 201\]](#)

6 Add Authorizations

Overview Graphic



Development Tasks

[Authorization Checks in Transactional Development Scenarios \[page 154\]](#)

[Implementing Authorizations for Read-Only Applications \[page 156\]](#)

[Implementing Authorizations for Generated Business Objects \[page 160\]](#)

- [Understanding the BOPF Authorization API \[page 592\]](#)

Administrative Tasks

[Defining Authorizations for External Service Consumption \[page 168\]](#)

- [Assigning Authorization Defaults to OData Services \[page 167\]](#)
- [Assigning Authorizations to Roles \[page 169\]](#)

Related Information

[Authorization Concept in BOPF \[page 165\]](#)

6.1 Authorization Checks in Transactional Development Scenarios

Business applications require an authorization concept for their data and for the operations on their data. Display and CRUD operations, as well as specific business-related activities, are therefore allowed for authorized users only.

In a transactional development scenario, you can add authorization checks to various components of an application. In this case, different mechanisms are used to implement the authorization concept.

Read Access Control with DCL

ABAP CDS has its own authorization concept based on a data control language (DCL). The authorization and role concept of ABAP CDS uses conditions defined in CDS access control objects to check the authorizations of users for read access to the data model and data in question. In other words, access control allows you to limit the results returned by a CDS entity to those results you authorize a user to see.

More on this: [Implementing Authorizations for Read-Only Applications \[page 156\]](#) and

Authorizations for Generated Business Using BOPF API

In BOPF, an authorization check is introduced as an entity of a business object node used to manage authorizations on the node level. Using this authorization and role concept allows you to check whether a user is allowed to perform a certain activity (action) at the node instances or to manipulate certain node data (create, delete, and update operations).

More on this: [Implementing Authorizations for Generated Business Objects \[page 160\]](#)

Authorizations for OData Services Consumption

SAP Gateway provides predefined roles as templates for developers, administrators, end users of the content scenarios, and support colleagues. You, as the SAP customer, will configure the roles based on these templates and assign users to the roles.

More on this: [Defining Authorizations for External Service Consumption \[page 168\]](#)

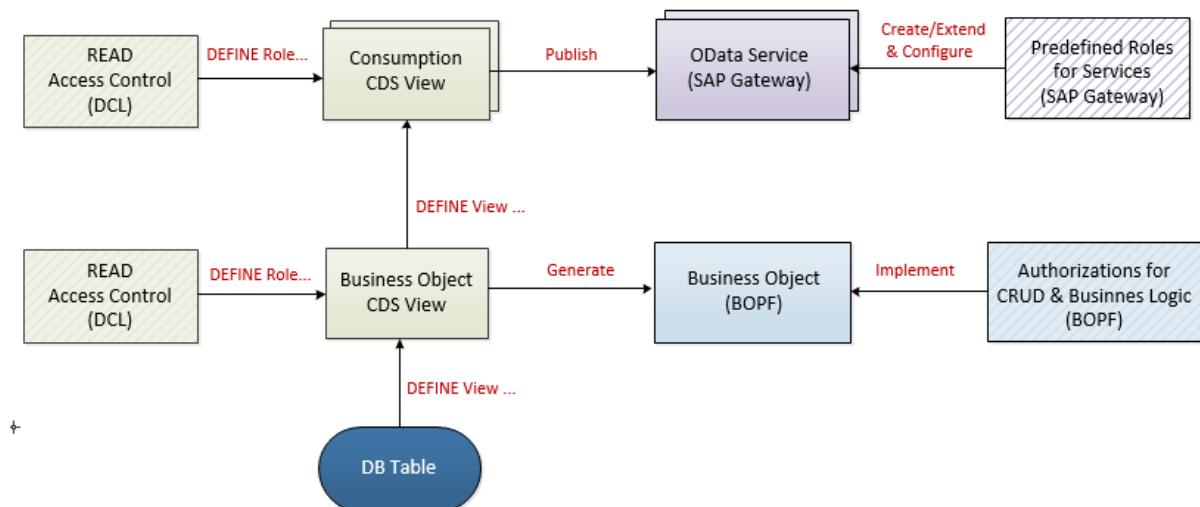
Summary (at a Glance)

The figure below provides you with an overview of the main design time components in a transactional development scenario, including the artifacts required for enabling authorization checks at all levels of the application.

Normalized views (or database tables) serve as the data source for modeling the data associated with the business object layer. To check the authorizations of users for read access to CDS-based data model and data, a corresponding CDS role is defined using the data control language (DCL). A CDS role specifies access rules. Each access rule defines access to the CDS view that the role is assigned to. Different access controls are created for access control at business object data model level (in figure: *Business Object CDS View*) and at the consumption level (in figure: *Consumption CDS View*).

The root BO CDS view is also used to generate the corresponding BOPF BO. The generated BO provides appropriate code exits for implementing the CRUD authorization contract based on well-defined authorization BOPF APIs.

For developers at SAP, no further steps (concerning authorizations) are required for the resulting OData service to be consumed in the customer's landscape. SAP gateway already provides predefined roles as templates for accessing the OData services and SAP Fiori apps.



Development Process

1. Adding read-only access control to business object data model
More on this: [Implementing Authorizations for Read-Only Applications \[page 156\]](#) and [ABAP CDS - Access Control \(ABAP Keyword Documentation\)](#)
2. Adding read-only access control to consumption model
3. Implementing authorization checks for generated business objects
More on this: [Implementing Authorizations for Generated Business Objects \[page 160\]](#)
4. Create, extend and configure roles in the SAP Gateway landscape for OData service consumption (relevant for SAP Customers only)
More on this: [Roles in the SAP Gateway Landscape](#)

6.2 Implementing Authorizations for Read-Only Applications

Access controls with DCLs enables you to add an authorization filter directly to a query of a CDS entity. The filter then limits the data the user can see.

We have created the SalesOrderItem CDS entity (see [Define a Data Model Based on CDS Views \[page 13\]](#) above). Now we want to add user authorizations to restrict the query results. Here we describe use cases with examples of access controls for this CDS entity.

Creating Access Controls

Access controls enable you to filter access to data in the database based on fixed values or conditions on PFCG, among others. We use data control language (DCL) to write access controls. If no access control is created and deployed for the CDS entity, a user who can access the CDS entity can view all the data returned.

For more information about how to start the access control editor, see .

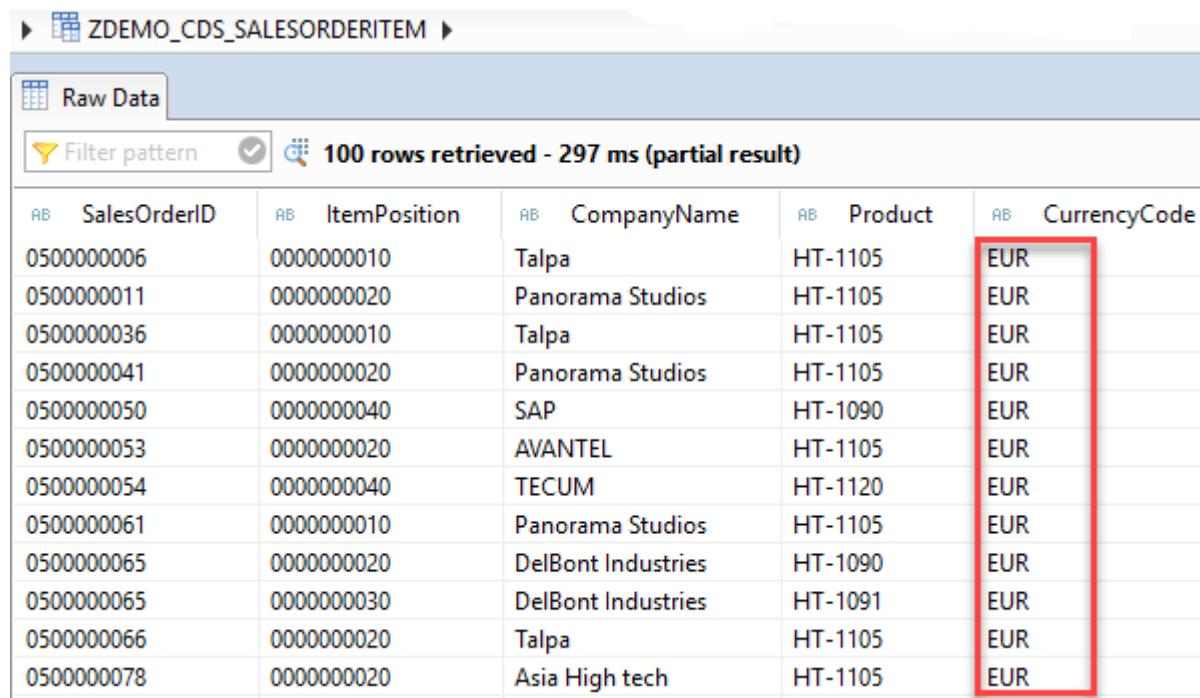
Use Case 1: Access Control with a Fixed Value

In this example, we restrict access to the items with currency code *EUR*. This example is a very simple one, because you could achieve the same thing much easier by adding a *WHERE* condition to the CDS entity like `WHERE CurrencyCode = 'EUR'`. In the next example we will add user relevant values.

Code Syntax

```
@EndUserText.label: 'Access Control for Sales Order Item'  
@MappingRole: true  
define role ZDEMO_CDS_SalesOrderItem {  
    grant select on  
        ZDEMO_CDS_SalesOrderItem  
        where  
            CurrencyCode = 'EUR';  
}
```

In the data preview, you can see that only the items with currency code Euro are displayed:



AB SalesOrderID	AB ItemPosition	AB CompanyName	AB Product	AB CurrencyCode
0500000006	0000000010	Talpa	HT-1105	EUR
0500000011	0000000020	Panorama Studios	HT-1105	EUR
0500000036	0000000010	Talpa	HT-1105	EUR
0500000041	0000000020	Panorama Studios	HT-1105	EUR
0500000050	0000000040	SAP	HT-1090	EUR
0500000053	0000000020	AVANTEL	HT-1105	EUR
0500000054	0000000040	TECUM	HT-1120	EUR
0500000061	0000000010	Panorama Studios	HT-1105	EUR
0500000065	0000000020	DelBont Industries	HT-1090	EUR
0500000065	0000000030	DelBont Industries	HT-1091	EUR
0500000066	0000000020	Talpa	HT-1105	EUR
0500000078	0000000020	Asia High tech	HT-1105	EUR

Data Preview of Sales Order Item with Access Limited to Currency Code Euro

Use Case 2: Access Control Based on a PFCG Authorization

If you use the PFCG_AUTH aspect in the access control, user-dependent PFCG authorizations are used when accessing the CDS view. To implement this, there must be a PFCG object in the ABAP system. If you want to see the data, your user must be assigned a role that includes this authorization object with the matching value in the relevant field.

In this example, we have the same CDS entity *SalesOrderItem* again, and we want to limit the results by the value of the authorization object *S_ACMDemo*, which we map to fields of a sales order item entity.

Step 1: In your ABAP system, use transaction SU01 to assign your user the role that includes the relevant authorization object and field (in our example, the role *SAP_INTNW_EPM_DEMO* with the authorization object *S_ACMDemo*).

Change Role: Authorizations			
Selection criteria Manually Organizational levels... Trace Information Versions			
Role	SAP_INTNW_EPM_DEMO	Maint.	0 unmaint. org. levels, 0 open fields
Status	generated	Values	
Group/Object/Authorization/Field	Maintenanc...	A...	Value
> OOB Object Class AAAB	Manual		Cross-application Authorization Objects
< OOB Object Class AAAT	Manual		Cross-application, non-productive auth o
< OOB Authorization Object S_ACMD_DEMO	Manual	Display	Authorization object for ACM demo
< OOB Authorization	Manual	All values	Authorization object for ACM demo
ACTVT	Manual	Display	Activity
SACMTSOCS	Manual	All values	ACMTST: SalesOrder Lifecycle Status
SACMTSOID	Manual	0500000001-0500000003	ACMTST: Sales Order ID
SACMTCOUNTRY	Manual	*	Country Key
SACMORGUID	Manual	*	ACMTST: Org-Unit ID

Role with PFCG Authorization Object

Step 2: In the access control editor, map the field SACMTSOID of S_ACMD_DEMO to the field SalesOrderID of the CDS entity SalesOrderItem and add display authorizations (ACTVT = 03).

The code snippet below shows how you would specify this in the access control editor:

Code Syntax

```

@EndUserText.label: 'Access Control for Sales Order Item'
@MappingRole: true
define role ZDEMO_CDS_SalesOrderItem {
    grant select on
        ZDEMO_CDS_SalesOrderItem
        where
            ( SalesOrderID ) =
                aspect pfcg_auth( S_ACMD_DEMO, SACMTSOID, ACTVT = '03' );
}

```

In the data preview, you can see that only the items with SalesOrderIDs from 50...01 to 50...03 are displayed.

ZDEMO_DDL_SALES_ORDER_ITEM

ZDEMO_CDS_SALEORDERITEM

Raw Data

Filter pattern 23 rows retrieved - 341 ms

SalesOrderID	ItemPosition	CompanyName	Product	CurrencyCode
0500000001	0000000010	DelBont Industries	HT-1030	EUR
0500000001	0000000040	DelBont Industries		USD
0500000001	0000000050	DelBont Industries	HT-1036	CAD
0500000002	0000000010	TECUM	HT-1040	EUR
0500000003	0000000010	Asia High tech	HT-1073	EUR
0500000001	0000000030	DelBont Industries		EUR
0500000002	0000000030	TECUM	HT-1042	EUR
0500000002	0000000060	TECUM	HT-1052	EUR
0500000002	0000000070	TECUM		EUR
0500000001	0000000070	DelBont Industries	HT-1020	EUR
0500000001	0000000100	DelBont Industries	HT-1023	USD
0500000002	0000000050	TECUM		USD
0500000001	0000000090	DelBont Industries	HT-1022	ARS
0500000002	0000000020	TECUM	HT-1041	ARS
0500000002	0000000080	TECUM		JPY
0500000003	0000000040	Asia High tech	HT-1082	ARS

Data Preview of Sales Order Item with Access Control Based on PFCG Authorization

Use Case 3: Access Control with Combination of Fixed Value and PFCG Authorization

You can use the AND and OR operators to join conditions. In this example, we combine use cases 1 and 2 to only retrieve sales order items with SalesOrderIDs from 50...1 to 50...3 and currency code Euro:

Code Syntax

```
@EndUserText.label: 'Access Control for Sales Order Item'
@MappingRole: true
define role ZDEMO_CDS_SaleOrderItem {
    grant select on
        ZDEMO_CDS_SaleOrderItem
        where
            CurrencyCode = 'EUR' and
            ( SalesOrderID ) =
                aspect pfccg_auth( S_ACN_DEMO, SACMTSOID, ACTVT = '03' );
}
```

In the data preview, you can see that only the items with SalesOrderID from 50...01 to 50...03 and currency code Euro are displayed.

SalesOrderID	ItemPosition	CompanyName	Product	CurrencyCode
0500000001	0000000010	DelBont Industries	HT-1030	EUR
0500000002	0000000010	TECUM	HT-1040	EUR
0500000003	0000000010	Asia High tech	HT-1073	EUR
0500000001	0000000030	DelBont Industries		EUR
0500000002	0000000030	TECUM	HT-1042	EUR
0500000002	0000000060	TECUM	HT-1052	EUR
0500000002	0000000070	TECUM		EUR
0500000001	0000000070	DelBont Industries	HT-1020	EUR

Data Preview of Sales Order Item with Access Control with Combination of Fixed Value and PFCG Authorization

6.3 Implementing Authorizations for Generated Business Objects

If you are going to implement application-specific authorizations, you must consider the state of the business entity (active or draft data version).

→ Remember

- Active data represents the state of the business entity that is stored in the active persistence.
- Draft data represents the transient state of a business entity until it is permanently stored in the persistence layer as active data. The draft data is stored in the draft persistence pending transition to the active data.

Authorization Checks for Active BO Instances

The authorization checks for active data require an application-specific authorization exit class to be implemented at root node level of the business object.

Implementation Steps – Overview

Create an authorization class for the relevant BO

To create an authorization exit class for the relevant BO, open the editor for the root node and choose the *Authorization* tab. Here, you can check first whether an authorization class is already assigned to the relevant BO root node. If this is not the case, you can create a new one or assign an existing one as the authorization class.

The screenshot shows the SAP BO editor interface. The title bar indicates the path: ZDEMO_I_SALESORDER_TP > ZDEMO_I_SALESORDER_TP > Authorization. The main area is titled "General Information" and contains a section for "General Information of Authorization". Under "Authorization Class", there is a checkbox "Define Own Authorization Class" which is checked, and the field "Authorization Class:" contains the value "ZCL_AU_DEMO_I_SALESORDER_TP". This field is circled in red. Below the input field are two buttons: "New..." and "Browse...". At the bottom of the screen, there is a navigation bar with tabs: Overview, Alternative Keys, Properties, Associations, Actions, Determinations, Validations, Authorization, and another Authorization tab. The "Authorization" tab is currently selected.

Application-specific authorization class in the BO editor

The generated skeleton of the authorization class looks as follows:

```

CLASS zdemo_i_salesorder_auth DEFINITION ABSTRACT
  PUBLIC
  INHERITING FROM /bobf/cl_lib_auth_draft_active
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
  METHODS:
    /bobf/if_lib_auth_draft_active~check_static_authority REDEFINITION,
    /bobf/if_lib_auth_draft_active~check_instance_authority REDEFINITION.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS zdemo_i_salesorder_auth IMPLEMENTATION.

METHOD /bobf/if_lib_auth_draft_active~check_static_authority.
ENDMETHOD.

METHOD /bobf/if_lib_auth_draft_active~check_instance_authority.
ENDMETHOD.

ENDCLASS.

```

Implement static checks in the CHECK_STATIC_AUTHORITY method

In the implementation of this method, you can check whether the user has the permission to access a functionality at all. The result of this static check does not depend on the specific node instance (data value).

i Note

This is usually called before execution of a core service in order to check whether the user has authorization to execute the functionality at all.

More on this: Method **CHECK_STATIC_AUTHORITY** [page 594]

Implement instance-based checks in the CHECK_INSTANCE_AUTHORITY method

In this method, you can evaluate the node instance data and then check whether the user has permission to display or change data where an authorization-relevant attribute has a specific value.

i Note

This method is usually called once the static authorization check has been passed.

More on this: [Method CHECK_INSTANCE_AUTHORITY \[page 596\]](#)

Authorizations at Subnode Level

Consider that authorizations at root level nodes of business objects implicitly have an impact on their composite subnodes. Within the BO hierarchy tree, at runtime the static and instance-based checks will be propagated along the composition tree. For example, authorization checks that are defined for the ROOT node will be propagated to ITEM node.

In case of executing an action that is assigned to a subnode, for example, or when modifying a subnode (CRUD operations), the DISPLAY authorization (activity: *DISPLAY*) and the CHANGE authorization (activity: *CHANGE*) are requested from the root node where the authorizations are calculated in the corresponding authorization exit class.

Authorization Checks for Draft BO Instances

Unlike active BO instances, an authorization check is explicitly required for draft versions of BO instances only in rare situations. This is because the creator of draft is always allowed to perform delete and update operations as well as execute actions. In those cases, the authorization exit is not invoked. However, while creating or activating the draft version, the following use cases should be considered:

Use Case 1: Creating new draft from scratch (NEW)

For initially created instances of business objects, only the static check is executed, without any instance-based checks. This is because we assume that the fields in the initially created draft instance do not normally contain any data to be checked (and even if they would, they could be changed at any time before the actual activation of the draft data).

Step 1: Create an authorization class for the root node of the BO

Step 2: Implement the static check in the CHECK_STATIC_AUTHORITY method

In the source code of this method check if the creation of a new draft instance is granted for the respective users:

```
method /bobf/if_lib_auth_draft_active~check_static_authority.  
  ...  
  case is_ctx-activity.  
    when /bobf/cl_frw_authority_check=>sc_activity-create.  
      " Check the static CREATE authorization here...  
  ...  
endmethod.
```

More on this: [Method CHECK_STATIC_AUTHORITY \[page 594\]](#)

i Note

If you need an instance-based check in very specific cases, you can wrap the creation of the instance into an action and implement the instance-based authorization check there too. Creating of a new draft version of a BO instance without a relationship with an active version makes it necessary to check for CREATE on the active node instance.

Use Case 2: Creating a draft for an active instance (EDIT)

Creating a draft for an existing active instance version makes it necessary to check for UPDATE on the active node instance.

Note that the instance-based part of authorization check is not implemented in the authorization exit class but in the draft class to be executed immediately before copying the active data to the draft instance.

Step 1: Implement the static check in the CHECK_STATIC_AUTHORITY method

For implementing a static check the authorization exit class is used. In this case, the check is calculated in the `check_static_authority` method with `activity = EXECUTE` and `action_name = EDIT`.

```
method /bobf/if_lib_auth_draft_active~check_static_authority.  
...  
    case is_ctx-activity.  
        when /bobf/cl_frw_authority_check=>sc_activity-create.  
            " Check the static authorization here...  
        ...  
    endmethod.
```

Step 2: Access the draft class of the draft BO

Node Overview

General Information

General Information of Node

Name: *	ZDEMO_I_SALESORDER_TP		New...	Browse...
Source CDS View:	ZDEMO_I_SALESORDER_TP		New...	Browse...
Description:				
Persistent Structure: *	ZSDEMO_I_SALESORDER_TP_D		New...	Browse...
Transient Structure:			New...	Browse...
Combined Structure: *	ZSDEMO_I_SALESORDER_TP			
Combined Table Type: *	ZTDEMO_I_SALESORDER_TP			
Database Table: *	ZDEMO_SO_TP			
<input checked="" type="checkbox"/> Create Enabled	<input checked="" type="checkbox"/> Update Enabled	<input checked="" type="checkbox"/> Delete Enabled		
<input type="checkbox"/> Text Node				

Implementation Details

Node Implementation Details of Draft Business Object

Draft Class: ZCL_DR_DEMO_I_SALESORDER_TP

< >

Accessing draft class of the draft BO

Step 3: Redefine the method /BOBF/IF_FRW_DRAFT~CREATE_DRAFT_FOR_ACTIVE_ENTITY of the draft class and add the authorization checks there:

```
method /bobf/if_frw_draft~create_draft_for_active_entity.  
  ...  
  " Check the instance-based UPDATE authorization here...  
  ...  
endmethod.
```

Use Case 3: Activating a draft instance (COPY draft to active)

Activating a draft for an existing active instance version makes it necessary to check for UPDATE on the active node instance.

Note that the authorization check is not implemented in the authorization exit class but in the draft class to be executed at runtime immediately before copying the draft data to the active instance.

Step 1: Access the draft class of the draft BO

Step 2: Redefine the method /BOBF/IF_FRW_DRAFT~COPY_DRAFT_TO_ACTIVE_ENTITY of the draft class and add the authorization checks there:

```
method /bobf/if_frw_draft~copy_draft_to_active_entity.  
  ...  
  " Check both, the static and instance-based UPDATE authorization here  
  ...  
endmethod.
```

6.3.1 Authorization Concept in BOPF

Business applications require an authorization concept for their data and for operations on their data. Display, create, delete and update activities are therefore allowed for authorized users only.

→ Remember

In BOPF, an authorization check is introduced as an entity of a business object root node that is used for managing authorizations. Using this authorization and role concept allows you to check whether a user can perform a certain activity on the node instances or manipulate certain node data.

Examples

Let us assume in a sales order application that sales orders with `customer_region AMERICAS` must be visible solely for authorized users. In addition, only authorized users should be able to change the status of individual sales orders to `DELIVERED` by executing the `SET_TO_DELIVERED` action.

→ Remember

Note that authorization checks implemented for the BO layer are not suitable to limit read access when using OData services. To limit read access for this use case, add the DCL-based access control to CDS views (BO CDS views and consumption CDS views). **More on this:**

Static and Instance-Based Authorization Checks

The generic authorization concept in BOPF differentiates between static and instance-based authorization checks:

- **Static checks:** Check whether the user has permission to perform a specific activity or an operation on data (for example: `CREATE`). The result of a static check does not depend on specific data values.
- **Instance-based checks:** Evaluate the node data and check whether the user has permission to display or change data, or perform a specific activity where an authorization-relevant attribute has a specific value. The result of this check depends on the node data.

The first sales order example where `customer_region AMERICAS` must be visible solely for authorized persons is therefore handled by an instance-based check, whereas the second example (sales orders can be changed to status `DELIVERED` solely by authorized persons) is handled by a static check.

The static check utilizes the first field `ACTVT` and the second field `BOACT_NAME` of an authorization object depicted in the figure below. The instance-based check utilizes all authorization fields, especially application-specific ones like `B_REGION`.

Display Authorization Object	
Object	ZSO_AUTH
Text	Authorization object for protecting sales order data access
Class	AAAT Cross-application, non-productive auth objects for test
Author	[REDACTED]
Authorization fields	
Authorization Field	Short Description...
ACTVT	Activity
EPM_BP_ID	EPM: Business Partner ID
BOACT_NAME	Name of the BO action
B_REGION	CI buyer region

Authorization object for sales order app (example)

Privileged Mode in BOPF

BOPF services always run in privileged mode. Privileged mode is the processing mode that allows the implementation of a business object to have direct access to further resources and entities during a round trip. This means, authorization checks are executed when a BOPF service is accessed. If the implementation of the business object accesses further services during the roundtrip, for example, when executing a determination or an action, or even when accessing data of other business object, then the authorization is not checked any more.

Related Information

[Implementing Authorizations for Generated Business Objects \[page 160\]](#)

[Understanding the BOPF Authorization API \[page 592\]](#)

6.3.2 Assigning Authorization Defaults to OData Services

In the backend system, you add start authorizations for the OData services.

Prerequisites

- You have a user in the back-end system with the authorizations required for [Maintain Authorization Default Values](#) (transaction SU24).
- You have generated the required service artifacts.
For more information, see [Generate Service Artifacts From a CDS View \[page 21\]](#).

Procedure

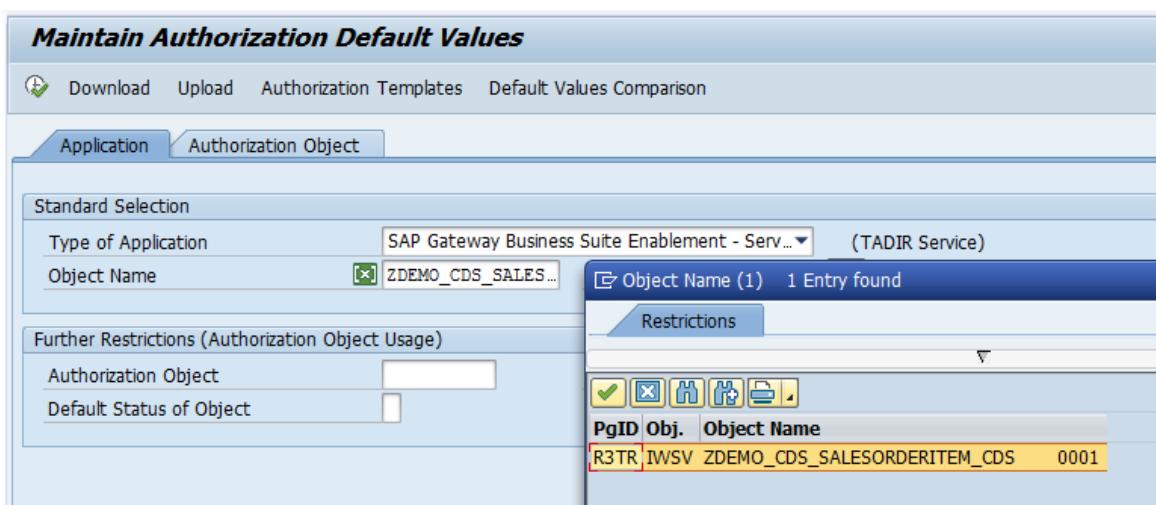
1. Open the SAP GUI for the relevant ABAP project by starting the SAP GUI Launcher *ABAP Development Tools* (icon  in the toolbar). Within the embedded SAP GUI, you are able to access the complete functionality of the classic ABAP Workbench.

i Note

Alternatively, you log in to the corresponding development system using the SAP GUI mode.

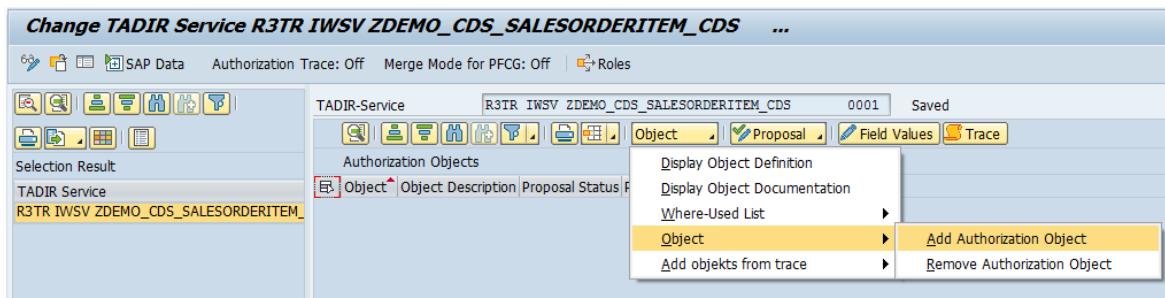
2. In the command field, enter the transaction code **SU24**.
3. As *Type of Application*, select *SAP Gateway Business Suite Enablement - Service*.
4. As *Object Name*, enter the name of your R3TR IWSV object (in our case: `ZDEMO_CDS_SALESORDERITEM_CDS*`).

Use the * (asterisk) at the end of the name and then select the correct name with version (in our case: 0001).

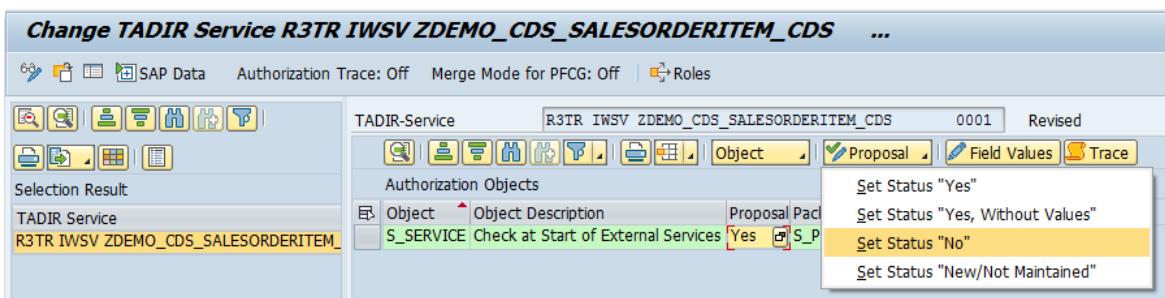


PgID	Obj.	Object Name	Version
R3TR	IWSV	ZDEMO_CDS_SALESORDERITEM_CDS	0001

5. Execute the program by pressing F8.
6. On the *Change TADIR Service* screen, change into the *Edit* mode.
7. Choose **Object > Object > Add Authorization Object** and enter `s_SERVICE` as the authorization object.



8. Set the proposal status to `No`.



9. Add the authorization objects used in the DCL of the CDS view with Proposal `Yes` and specify values which should be fixed or with Proposal `Yes, Without Values`
10. Save and transport your changes.

6.4 Defining Authorizations for External Service Consumption

→ Remember

For you as service developers (at SAP), there are no further steps required for the service to be consumed externally within the customer's landscape. In particular, you don't need to provide any authorization default values of the authorization objects and specific role templates required for execution of your service. SAP Gateway already provides predefined roles as templates for accessing SAP Fiori apps.

Process Steps (on Customer's Side):

Using the profile generator (transaction PFCG), the authorization administrators on the customer's side create or extend roles based on the delivered SAP role templates.

Authorization administrators configure the roles based on the provided template and assign users to the roles.

The role templates specify the authorizations for content that can be accessed by users of the specific SAP Fiori app within the customer's namespace.

For detailed information, take a look at:

- [Assigning Authorization Defaults to OData Services \[page 167\]](#)
- [Assigning Authorizations to Roles \[page 169\]](#)

Related Information

[Roles in the SAP Gateway Landscape](#)

6.4.1 Assigning Authorizations to Roles

You need to assign the start authorization to users. You can do that by extending the authorizations of an existing backend role.

Prerequisites

- You have a user in the back-end system with the authorizations required for [Role Maintenance](#) (transaction PFCG).
- Either the role already exists in the back-end system or you must create a new role and assign it to your test user. In this procedure, we assume that you already have a role assigned to your test user.

Procedure

1. Open the SAP GUI for the relevant ABAP project by starting the SAP GUI Launcher *ABAP Development Tools* (icon  in the toolbar). Within the embedded SAP GUI, you are able to access the complete functionality of the classic ABAP Workbench.

i Note

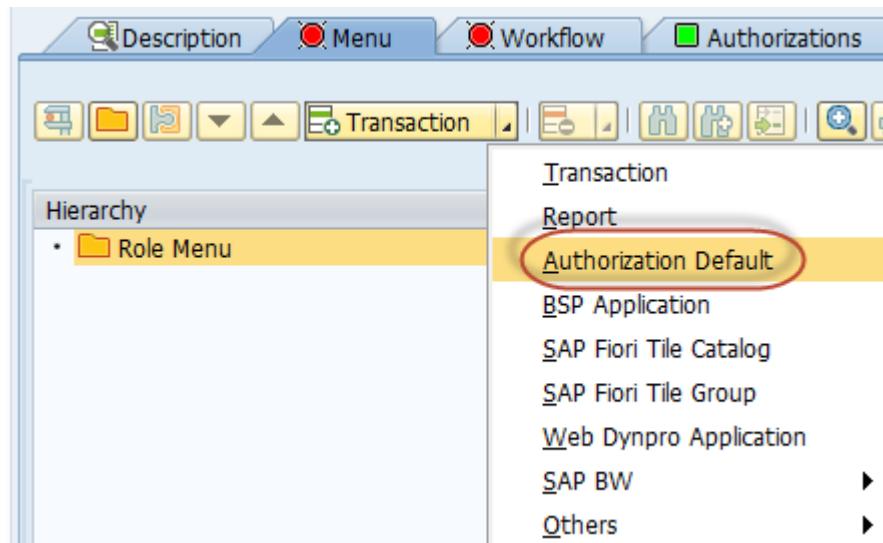
Alternatively, you log in to the corresponding development system using the SAP GUI mode.

2. In the command field, enter the transaction code **PFCG** to start the Role Maintenance.
3. Enter the name of the backend role to be extended (in our case: **ROLE-APW-68**).

Role Maintenance

The screenshot shows the SAP Role Maintenance interface. At the top, there are icons for creating, deleting, and changing roles, followed by a 'Transactions' button. Below this, a table has 'Role' and 'Short Description' columns. The 'Role' column contains the value 'ROLE-APW-68', which is circled in red. To the right of the table are three small icons: a pencil for change, a question mark for help, and a magnifying glass.

4. Choose *Change*.
5. On the *Menu* tab, choose **Transaction > Authorization Default**



6. As *Authorization Default*, select *TADIR Service*.
7. As *Object Type*, select *IWSV SAP Gateway Business Suite Enablement - Service*.
8. Enter the name of the generated backend enablement service (in our case: ZDEMO_CDS_SALESORDERITEM_CDS*).

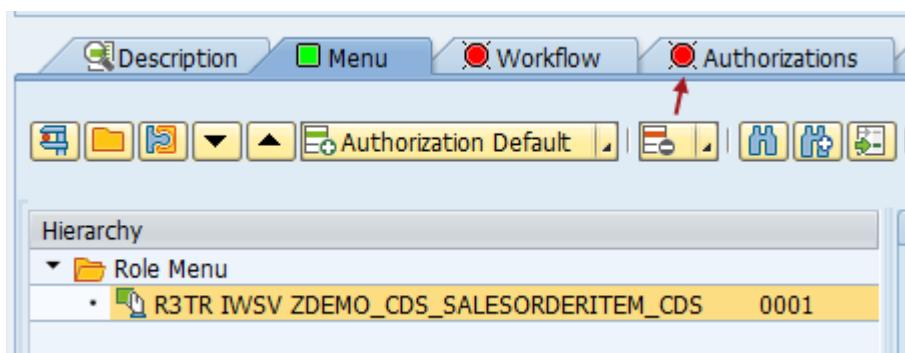
Use the * (asterisk) at the end of the name and then select the correct name with version (in our case: 0001) from the value help (press F4).

Service

The screenshot shows the SAP Service configuration screen. It has fields for 'Authorization Default' (set to 'TADIR Service'), 'Program ID' (set to 'R3TR'), and 'Obj. Type' (set to 'IWSV SAP Gateway Business Suite Enablement -'). Below these, a table lists services. The first row shows 'TADIR Service' and 'Text'. The second row shows 'ZDEMO_CDS_SALESORDERITEM_CDS' and '0001'. This second row is circled in red. To the right of the table, there are four columns: 'R3TR', 'IWSV', 'ZDEMO_CDS_SALESORDERITEM_CDS', and '0001'.

9. Choose *Copy*.
10. On the *Change Roles* screen, choose *Save*.

The status icon on the *Authorizations* tab turns red.



11. On the *Authorizations* tab, choose *Change Authorization Data*.

Now the start authorization and the authorization defaults for the service are merged into the role's authorizations.

Role ROLE-APW-68 Maint. 0 unmaint. org. levels, 4 open fields Status: Changed			
<input type="button" value="Status"/> <input type="button" value="Edit"/> <input type="button" value="Values"/> <input type="button" value="Search"/> <input type="button" value="Action"/>			
Group/Object/Authorization/Field	Maintain...	Update ...	Action
Object class AAAB	Standard	New	
Authorization Object S_SERVICE	Standard	New	
Authorization T-H502000100	Standard	New	
SRV_NAME	Standard	<input type="button" value=""/>	B
SRV_TYPE	Standard	<input type="button" value=""/>	H
Object class AAAT	Manually	Old	

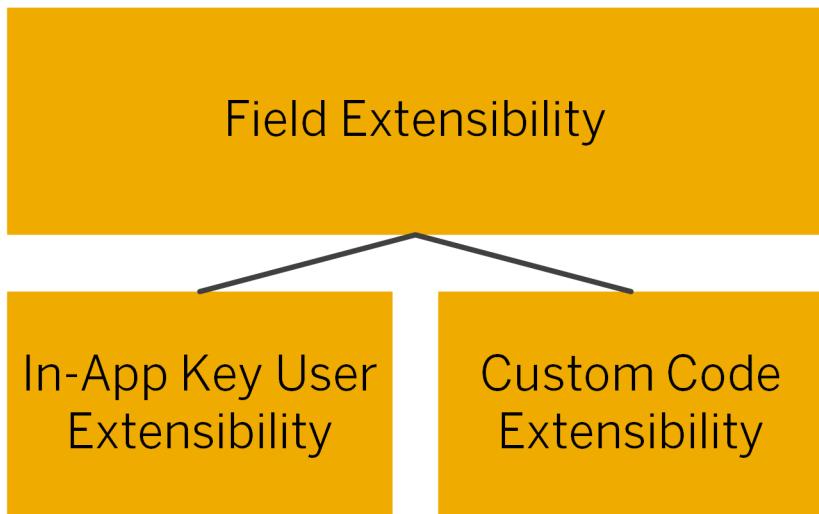
12. Choose *Save*.
13. Maintain the open fields for the authorization objects from the DCL.
14. Choose *Generate* to update the role.

7 Extend

Context

There are various approaches to extend an application with additional features. You can extend your application with additional fields (either as a key user in the app or by extending the delivered code in the back end), with additional nodes or functionality for your business object, or with an additional artifact to separate annotations from your CDS data model.

Click the boxes to get more information about the different extensibility options.



- [#unique_9/unique_9_Connect_42_subsection-im1 \[page 173\]](#)
- [#unique_9/unique_9_Connect_42_subsection-im2 \[page 174\]](#)
- [#unique_9/unique_9_Connect_42_subsection-im3 \[page 174\]](#)
- [#unique_9/unique_9_Connect_42_subsection-im4 \[page 174\]](#)
- [#unique_9/unique_9_Connect_42_subsection-im5 \[page 174\]](#)

Click the element for more information about each extensibility option.

Field Extensibility

Field extensibility means that you would like to provide additional fields in a given business application context that is represented by the UI actually used. These are fields that were not foreseen by the application when delivered by SAP. In such a case, you have the option to use the extensibility infrastructure of the ABAP platform in order to add custom fields to the original application.

You can use this infrastructure in different scenarios:

- **Key user extensibility** – You are solely interested in configurable extensions as they are used in the context of business power user (key user) adaption.
- **Custom code extensibility** – You want to maximize leverage of this infrastructure for field extensibility with custom code by implementing your own extensions.

→ Recommendation

If an SAP application is enabled for extensibility we recommend that you use the [key user](#) tool to create custom fields, even as a developer, because the key user tool hides the technical details of the SAP application from you.

On the other side, we recommend to create custom fields directly using extend view only if...

- The SAP application is not yet enabled for extensibility
- The functionality of the key user tool is not sufficient. For example, you want to do a calculation in a custom field.

Be aware that custom code extensibility requires the entire development cycle to implement extensions by means of code additions. This, in turn, requires an extension for the data model that was originally delivered in the form of CDS views and was published as an OData service for further consumption.

In-App Key User Extensibility

Key user extensibility allows key user to create custom fields directly in the app, if the SAP application is enabled for extensibility. The key user has a business perspective and does not need to take care of technical details. So, in the end, the key user is able to create and change custom fields and their properties (name, label, type, or the position on the screen) without adding or modifying any delivered code.

More on this: [SAP Fiori: Extensibility](#)

Use this option whenever possible and sufficient.

Custom Code Extensibility

More on this: [Extending Apps with Custom Code \[page 175\]](#).

Functional Extensibility (BOPF Enhancement Concept)

Standard BOPF BOs and their functionality can be extended with the extensibility options by [S/4 HANA](#) if the application is extensible (BOPF Enhancement concept).

More on this:

i Note

The BOPF enhancement concept is not applicable for business objects that are generated from CDS and used within the ABAP Programming Model.

Extensibility for Annotations

A CDS entity can be extended with metadata extension, a separate development object in which annotations can be maintained. This follows the guideline of [separation of concerns](#). This implies that the pure data model and its metadata are separated from each other.

More on this: [Metadata Extensions \[page 185\]](#)

7.1 Extending Apps with Custom Code

As an SAP customer you may want - in some cases - to extend the user interfaces (UI) of the On-Premise applications delivered by SAP or SAP partners with additional fields.

Using a concrete example, this documentation section demonstrates how you, as an SAP customer, can implement field extensions with your own custom code in the context of the ABAP CDS extension model. In this example, we assume that the fields you want to use for your extension are already persisted on the database.

i Note

If you want to extend an application with fields that are not persisted on the database, you need to define an *append structure* first. You define all the fields to be extended in this structure. This is only possible if the database table is enabled for extensibility (annotation `@AbapCatalog.enhancementCategory : #EXTENSIBLE_ANY`).

Field Extensibility

The extensibility infrastructure of the ABAP platform enables the extension of existing application with your own custom code. That means, when you want to include new fields into your application you do not need to modify delivered code, but you simply attach new code to the relevant development artifacts. The infrastructure ensures that the new elements are added to all involved layer from database to UI (ABAP Dictionary, CDS; SAP Gateway, UI).

For each CDS view to be extended, the developer needs to create a data definition as development object and implement the corresponding CDS view extension using `EXTEND VIEW` syntax:

```
AbapCatalog.sqlViewAppendName: 'CDS_APPEND_VIEW'  
[@extension_annot1]  
[@extension_annot2]  
...  
EXTEND VIEW cds_view_original WITH cds_view_extension  
    [association1 association2 ...]  
    { select_list_extension } [;]
```

Using the syntax above, you can extend a delivered CDS view `cds_view_original` using a CDS view extension `cds_view_extension`. The CDS view extension adds the following elements to the `SELECT` list of the available view without making modifications:

- The elements of the specified extension list `select_list_extension`, as known from view fields
- Optional associations `association1, association2, ...` for the `SELECT` statement of the extended CDS view
- Further annotations `extension_annot1, extension_annot2...` can also be specified.

i Note

A CDS view can be extended using multiple CDS view extensions.

Typical Use Cases for Custom Code Adaption

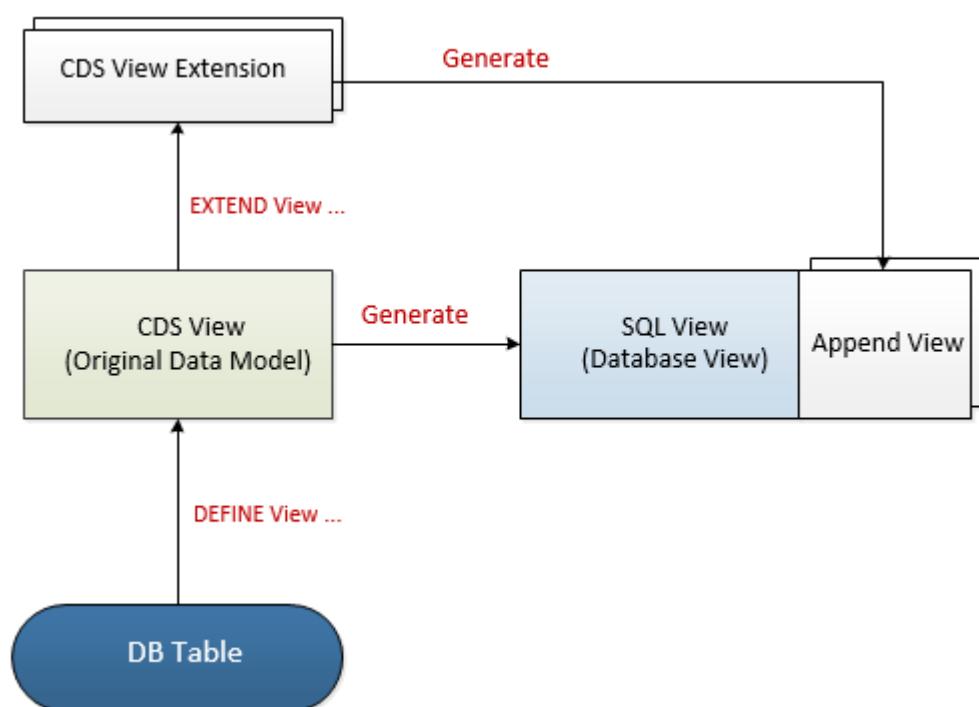
- Adding calculated fields
- Defining new associations in the extension code
- Adding fields resulting from new associations

CDS Extension Model and Append Views

The CDS extension is based on the proven append technique that you probably already know from the ABAP Dictionary. Special views, known as append views, are the type of views that you as an SAP customer can use to add new fields to existing database views. They are intended for enhancements of database views of the SAP standard.

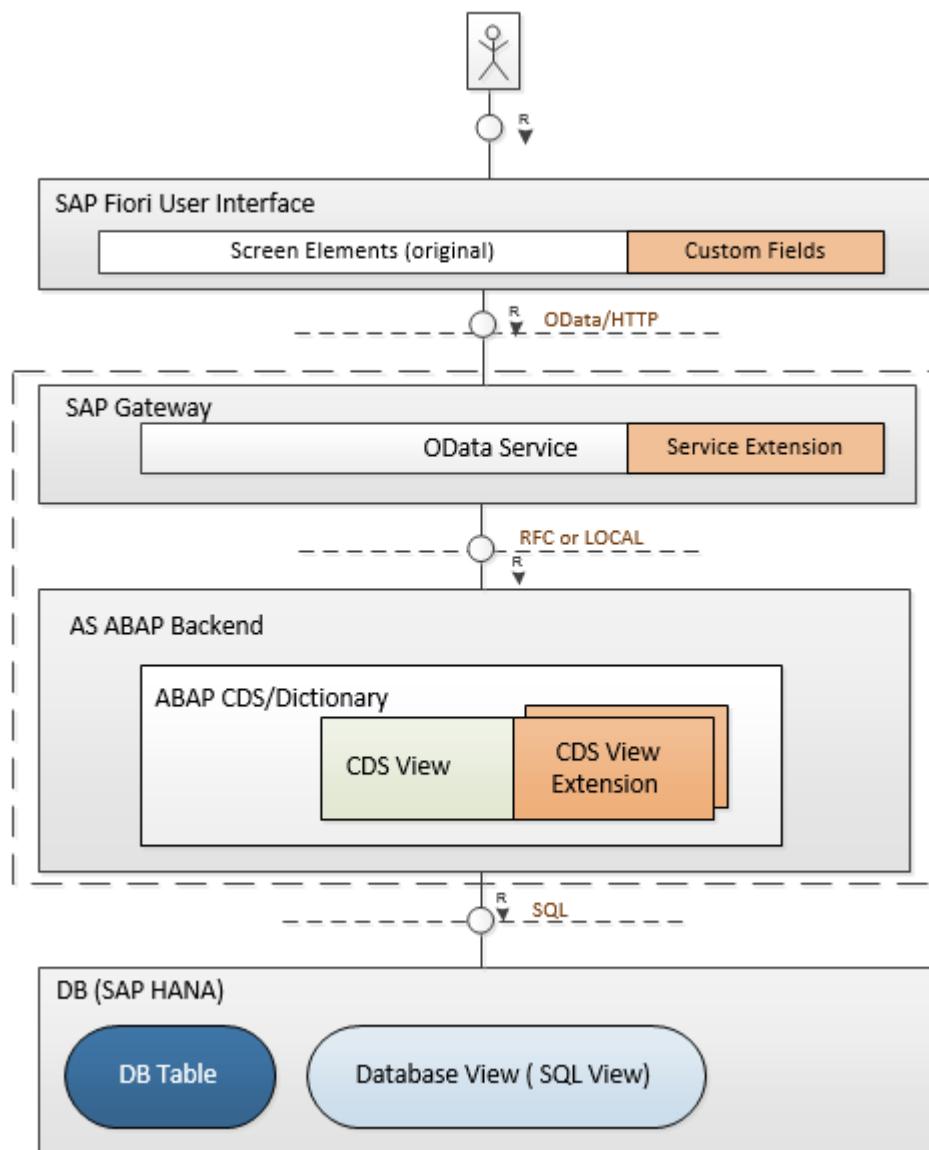
→ Remember

With an append view, fields of the original tables can be included in the view without modifications. An append view is assigned to one database view. However, you can create more than one append view for a database view. For each active database view, the system looks for all the append views that are assigned and their fields are then appended to the database view. When you create or change an append view, the assigned database view is automatically adjusted to this change when you activate the append view.



CDS extension model for field extensibility – design time view

Architecture Overview



Architecture with CDS extension model – runtime view

Related Information

[Syntax of ABAP CDS - EXTEND VIEW](#)

[Extensibility apps for key users](#)

[SAP Fiori: Extensibility](#)

7.1.1 Creating an Appropriate CDS View Extension

Prerequisites

- You need the standard developer authorization profile to create ABAP development objects with [ABAP Development Tools](#).
- You identified the data definition that implements the original CDS-based data model you are going to extend.

Context

Starting Point

Let us assume you - as an SAP customer – are accessing our own Fiori Launchpad where you open a list reporting app displaying all parties that are relevant for your business activities.

However, in your business context you may require some further details that were not foreseen by the application's UI delivered by SAP. For example, you would like to display the role (supplier, customer, partner, and so on) or the contact details for each organization displayed in the list. In other words: You would like to provide additional fields in the given business application context that is represented by the UI actually used.

Org ID	Organization	Location
100003380	Tessile Casa Di Roma	Rome >
100003380	Tessile Casa Di Roma	Rome >
600005660	Alessio Galasso	Rome >
600005660	Alessio Galasso	Rome >

Actually used UI – an example

Let us also assume you have identified the development object (data definition) that implements the original data model for [Party Address Data](#) - as shown in the listing below.

```
@AbapCatalog.sqlViewName: 'SQL_PARTY_ORIG'  
@AbapCatalog.compiler.compareFilter: true
```

```

@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Party Address Data'

@Search.searchable: true

@OData.publish: true

define view DEMO_PARTY_ORIG
    as select from SEPM_I_Party_E as Organization
{
    @UI.lineItem: { importance: #HIGH, label: 'Org ID', position: 10 }
    key Organization.Party as OrgID,
    @Search.defaultSearchElement: true
    @UI.lineItem: { importance: #HIGH, label: 'Organization', position: 20 }
    Organization.PartyName as OrgName,
    @UI.selectionField.position: 10
    @Search.defaultSearchElement: true
    @Search.fuzzinessThreshold: 0.5
    @UI.lineItem: { label: 'Location', position: 30 }
    Organization.CityName as City
}

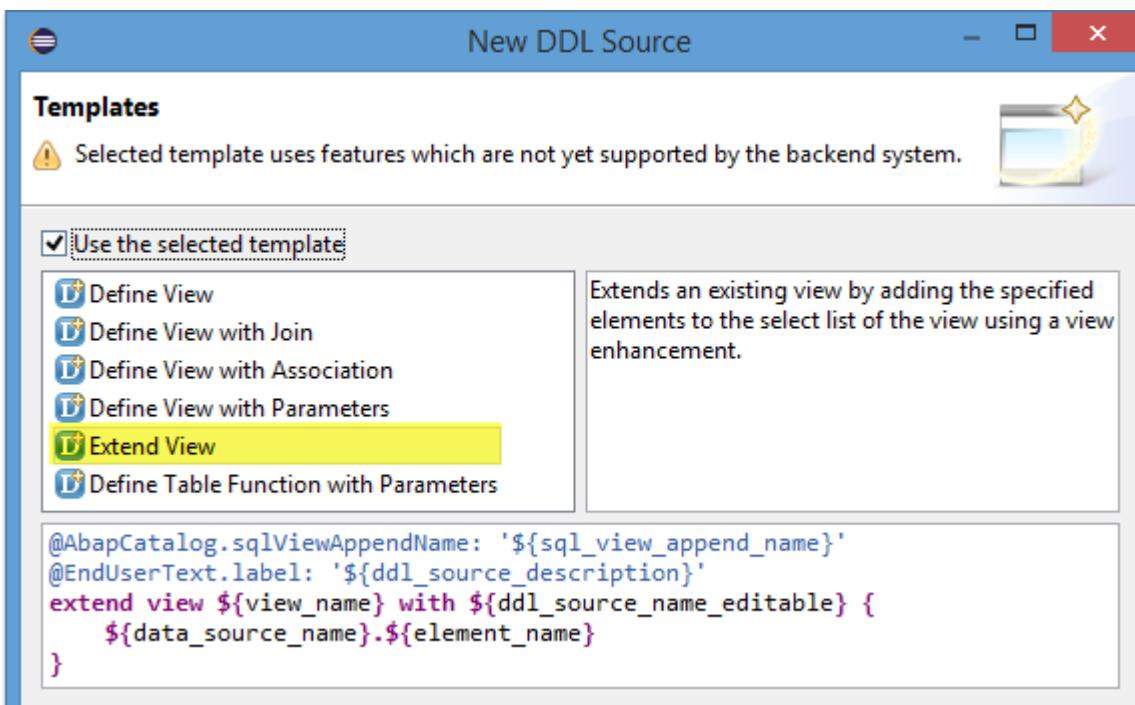
```

The original source code defines quite a simple data model based on the CDS view called `DEMO_PARTY_ORIG`. This view is implemented by means of a query for performing a `SELECT` statement, where the predefined CDS view (that originates from the EPM demo application) `SEPM_I_Party_E` is used as the data source. The select list includes a rather small set of fields for the ID, name, and location of the parties. In addition to the original CDS view, the SQL view (database view) `SQL_PARTY_ORIG` has also been created in the ABAP Dictionary.

Procedure

To create a data definition for extending a CDS view, proceed as follows:

1. Launch the *ABAP Development Tools*.
2. In your ABAP project, select the relevant package node in the *Project Explorer*.
3. Open the context menu and choose to launch the creation wizard.
4. In addition to the *Project* and *Package*, enter the *Name* (with due regard to your namespace) and the *Description* for the extension to be implemented. Choose *Next*.
5. Assign a transport request and choose *Next*.
6. Select the *Extend View* template to speed up the extension definition.



Selecting a suited template for extending a view

7. Choose *Finish*.

Results

In the selected package, the ABAP back-end system creates an inactive version of a data definition for the extension and stores it in the ABAP Repository. The generated source code is automatically displayed in the DDL editor and already provides you with the necessary view annotations and adds placeholders for names of the append view, the original CDS view, and for the actual view extension.

The code is annotated with arrows pointing to specific parts:

- 'original CDS view' points to the 'view_name' placeholder in the 'extend view' statement.
- 'CDS view extension' points to the 'Zdemo_Party_Ext' placeholder in the 'extend view' statement.
- 'Append view' points to the 'sql_view_append_name' placeholder in the first line of the code.

The generated template code in the DDL editor

Next Steps

If you have not yet already done so, open the newly created data definition and specify the names of the...

- Append view to be generated in the ABAP Dictionary, for example: ZSQL_PARTY_EXT

- Original CDS view: DEMO_PARTY_ORIG (in our example)
- Actual view extension, for example: ZDEMO_PARTY_EXT.

7.1.2 Adding Custom Fields to Extension View

This topic demonstrates how you can extend the syntax of the original CDS view with a view extension to provide some additional custom fields.

Adding predefined fields from original data source

In our example, by far not all fields from the original data source have been adopted into the data model. For example, details on the exact address of the organizations are missing. As a substitute for further address data, we will adopt the field `CountryName` into the list of line items:

```
...
    -- Adding predefined field
    @UI.lineItem: { label: 'Country', position: 35 }
        Organization.CountryName
...

```

Adding a calculated field

Fields that were not foreseen and result only after a calculation based on other defined fields are a very good example of the use of view extensions. In the following listing, a new field named `party_role` has been introduced. The values of this field are determined first within the case statement. In this way, the specification of the role `Customer` or `Supplier` can be output for each individual organization.

```
...
    -- Adding calculated field
    @UI.lineItem: { label: 'Role', position: 25}
        case Organization.PartyRole
            when '01' then 'Customer'
            when '02' then 'Supplier'
            else 'Unspecified'
        end    as party_role
...

```

Results

The resulting source code for the CDS view extension is the following:

```
@AbapCatalog.sqlViewAppendName: 'ZSQL_PARTY_EXT'
@EndUserText.label: 'Party Adress Data Extended'
```

```

extend view DEMO_PARTY_ORIG with ZDEMO_PARTY_EXT
{
    -- Adding predefined field
    @UI.lineItem: { label: 'Country', position: 35 }
    @Search.defaultSearchElement: true
    Organization.CountryName,
    -- Adding a calculated field
    @UI.lineItem: { label: 'Role', position: 25 }
    @Search.defaultSearchElement: true
    case Organization.PartyRole
        when '01' then 'Customer'
        when '02' then 'Supplier'
        else 'Unspecified'
    end as party_role
}

```

After successful activation of the data definition, two ABAP Dictionary objects are created for the CDS view extension and are automatically added to the same ABAP package (and therefore the same transport request) as the underlying DDL source:

- The actual CDS view extension that is specified after the `EXTEND VIEW` keyword
- The CDS append view for a classic append view that is specified in quotation marks after the `@AbapCatalog.sqlViewAppendName` annotation. The new append view extends the CDS SQL database view of the extended CDS view.

⚠ Caution

The name given to the append view can no longer be changed after the CDS view has been transported into a follow-on system.

7.1.3 Adding Fields from Association

This topic demonstrates how you can extend the syntax of the CDS view with view extension to provide some additional custom fields that result from new associations defined in the extension code.

Creating a Second Extension

Create a data definition second as the second development object for extending the original CDS view `DEMO_PARTY_ORIG` and specify the names of the append view as `ZSQL_PARTY_EXT2` and `ZDEMO_PARTY_EXT2` for the new view extension. This association `_PartyContact` associates the original CDS view with the target data source `SEPM_I_ContactPerson_E`. The target is a predefined CDS view that originates from the EPM demo application you can use to provide contact data for the business partner. For representation of this contact data, we adopt only the field `EmailAddress` into the list of `LineItems`.

```

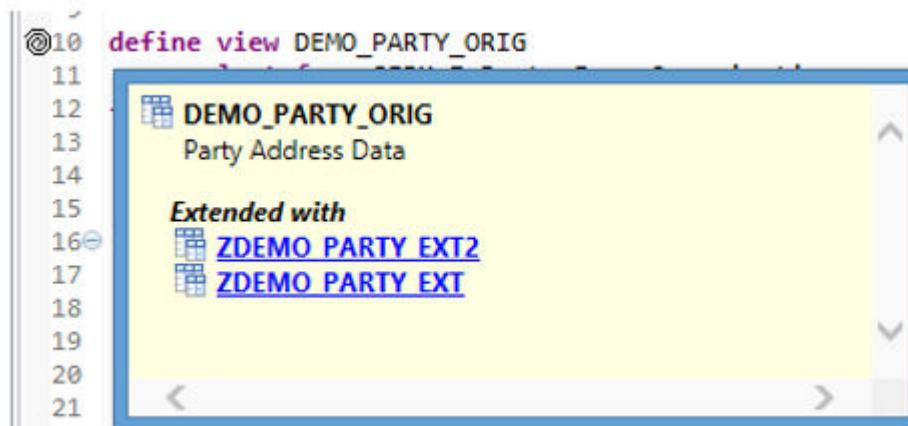
@AbapCatalog.sqlViewAppendName: 'ZSQL_PARTY_EXT2'
@EndUserText.label: 'Party Address Data Extended with Contact'
extend view DEMO_PARTY_ORIG with ZDEMO_PARTY_EXT2
    association [1..1] to SEPM_I_ContactPerson_E as _PartyContact
    on Organization.businesspartner = _PartyContact.businesspartner
{
    @UI.lineItem: { label: 'Contact - Email', position: 50 }
}

```

```
    }  
    _PartyContact._BusinessPartner.EmailAddress
```

Results

If you open the DDL editor with the original data definition, you will detect that (after a refresh) a new decorator  indicates that extensions are available in the current system. If you move the cursor over the decorator, an info screen provides you with links to all existing view extensions. Prerequisites: both extension data definitions has been activated successfully in ABAP Development Tools.



```
@10 define view DEMO_PARTY_ORIG  
11  
12     DEMO_PARTY_ORIG  
13     Party Address Data  
14  
15     Extended with  
16     ZDEMO PARTY EXT2  
17     ZDEMO PARTY EXT  
18  
19  
20  
21
```

Info screen after a successful activation of both extensions

If you now open the original database view in the ABAP Dictionary (in our example: SQL_PARTY_ORIG), you will see that all appends are included in the database view

Dictionary: Display View

View field	Table	Field	Key	Data elem.
MANDT	SEPM_IPARTYE	MANDT	<input checked="" type="checkbox"/>	MANDT
ORGID	SEPM_IPARTYE	PARTY	<input checked="" type="checkbox"/>	SNWD PARTY ID
ORGNAME	SEPM_IPARTYE	PARTYNAME	<input checked="" type="checkbox"/>	SNWD PARTY NAME
CITY	SEPM_IPARTYE	CITYNAME	<input checked="" type="checkbox"/>	SNWD CITY
.APPEND	ZSQL_PARTY_EXT		<input checked="" type="checkbox"/>	
CITYNAME	SEPM_IPARTYE	CITYNAME	<input checked="" type="checkbox"/>	SNWD CITY
COUNTRYNAME	SEPM_IPARTYE	COUNTRYNAME	<input checked="" type="checkbox"/>	LANDX50
PARTY ROLE	DDDLCHARTYPES	CHAR*000011*000...	<input checked="" type="checkbox"/>	
.APPEND	ZSQL_PARTY_EXT2		<input checked="" type="checkbox"/>	
EMAILADDRESS	SEPM_IBUPAE	EMAILADDRESS	<input checked="" type="checkbox"/>	SNWD EMAIL ADDRESS

Resulting database view includes all appends from extension

→ Tip

To verify the result set of the extended CDS view, you can utilize the [Data Preview](#). More on this: [Verify the Result Set in the Data Preview Tool \[page 18\]](#)

7.1.4 Running the Resulting Fiori App

To make sure that the added custom fields work correctly for the end user, we are now going to run the resulting application.

Prerequisites

For the corresponding OData service we assume that...

- The original CDS view is exposed as OData service using the so-called auto exposure approach (`@OData.publish: true`).
More on this: [Generating OData Service With Auto-Exposure \[page 19\]](#)

- *Smart Templates* as Fiori building blocks are used in the course of UI development.
More on this: [Consume Business Data Using SAP Fiori Elements \[page 27\]](#)

Procedure

If you finally run the app in the SAP Fiori Launchpad, the list reporting app provides you with extended functionality - without any further (configuration) steps (no service activation required). The resulting UI screen displays all added fields – as shown in the figure below.

Org ID	Organization	Role	Location	Country	Contact - Email
100003380	Tessile Casa Di Roma	Customer	Rome	Italy	alessio.galasso@tcdr.it
100003380	Tessile Casa Di Roma	Customer	Rome	Italy	alessio.galasso@tcdr.it
600005660	Alessio Galasso	Supplier	Rome	Italy	alessio.galasso@tcdr.it
600005660	Alessio Galasso	Supplier	Rome	Italy	alessio.galasso@tcdr.it

The resulting UI screen provides additional fields

7.2 Metadata Extensions

Metadata extensions enable you to add customer-specific annotations to SAP's CDS entities. Note that these changes do **not** result in modifications.

Definition

A metadata extension is a development object that provides CDS annotations in order to extend the CDS annotations used in a CDS entity. The standard ABAP Workbench functions (transport, syntax check, activation, and so on) are supported.

Use

Metadata extensions enable you to write the annotations for a CDS entity in a different document to separate them from the CDS entity.

 This functionality is supported as of SAP NetWeaver AS for ABAP 7.51 innovation package.

Overview

The metadata of CDS entities are not extensible by default.

To use a metadata extension for a CDS entity, you have to consider the following conditions:

1. In the definition of the CDS entity, the `@Metadata.allowExtensions` annotation with the value `true` is added. This annotation explicitly allows the use of metadata extensions.
2. In the metadata extension, you have to define the name of the CDS entity to be annotated in the `annotate view` statement.
3. In the Switch Framework, metadata extensions are switchable.

Advantages

You can benefit from the following advantages using metadata extensions:

1. **Separation of Concerns:** Separating the metadata specified in the annotations from the implementation of the entity:
 - Improves the readability of the source code
 - Simplifies the development and maintenance of the CDS entityIn addition, the metadata can be developed and updated independently of the data definition.
2. **ABAP Dictionary-independent activation:** When activating a CDS entity, the metadata extensions will be ignored. This results in the following advantages:
 - It reduces the number of ABAP Dictionary (mass) activations required to develop and maintain the CDS entity.
 - It speeds up the overall development process.
 - It facilitates changing the metadata of a CDS entity in a running system, thereby reducing downtime.
3. **Modification-free enhancements:** Customers, partners, and industries can customize the metadata without modifying the CDS entity.
In addition, metadata extensions are switchable. This means, the metadata can be specifically enabled or disabled depending on the use case.

Activation

In general, in a metadata extension only those annotations are permitted that do not affect the ABAP Dictionary activation/generation or the activation/generation of secondary objects (for example, OData services). For example, the ABAP annotation `@EndUserText` and the component-specific annotations `@UI` can be specified in metadata extensions. A syntax error occurs if annotations that are not permitted are specified.

Obtaining Merged Annotations

The metadata contained in metadata extensions is only available to consumption clients that access the CDS entity metadata using the ABAP API `CL_DD_DDL_ANNOTATION_SERVICE`. In this case, metadata in the metadata extensions is merged together with the metadata in the CDS entity and also with the metadata that is inherited from underlying entities (and metadata extensions) in the entity hierarchy.

Related Information

 [ABAP CDS Metadata Enhancements \(ABAP Keyword Documentation\)](#)

8 Common Tasks

----- Development Tasks Related to... -----

Object Model

[Adding Field and Action Control \[page 189\]](#)

[Providing Value Help \[page 275\]](#)

[Mapping CDS View-Elements onto Table Fields \[page 286\]](#)

[Implementing Draft-Enabled Associations \[page 288\]](#)

Data Provisioning

[Using Virtual Elements in CDS \[page 206\]](#)

[Consuming Temporal Data \[page 236\]](#)

[Deriving Values from Foreign Entities \[page 247\]](#)

[Enabling Text and Fuzzy Searches in SAP Fiori Apps \[page 259\]](#)

[Using Aggregate Data in SAP Fiori Apps \[page 263\]](#)

UI Semantics

[Defining Text Elements \[page 271\]](#)

[Adding Field Labels and Descriptions \[page 284\]](#)

[Defining CDS Annotations for Metadata-Driven UIs \[page 296\]](#)

----- Administrative Tasks -----

[Defining Authorizations for External Service Consumption \[page 168\]](#)

[Exposing CDS Entities as OData Service \[page 339\]](#)

8.1 Adding Field and Action Control

This topic treats the concept of field and action control in the context of SAP Fiori application development for **transactional** use cases.

Field and Action Control in a Nutshell

Field and action control is a means that you can provide as information to the service on how data has to be displayed for consumption in the SAP Fiori UI.

In general, the following characteristics are relevant when providing field control:

- *Read-only*: - Will the user be allowed to change the value of a field (or a node instance)?
- *Mandatory*: - Must the user provide a value?
- *Hidden*: - Should a field be available for consumption in the UI?

i Note

We do not cover this characteristics in this topic since the relevant information is provided by...

- The consumption annotation `@Consumption.hidden: true`. In such a case, this information is part of the consumption view, not part of the service.
More on this: [Consumption Annotations \[page 390\]](#)
- The UI annotation `@UI.hidden:true`. The information in this case is in fact part of the service, but not exposed to the end-user.
More on this: [UI Annotations \[page 465\]](#)

The field and action control can relate to individual properties (node attributes in BOPF) or a whole entity (business object node instance in BOPF).

The field (and action) control information is either static or dynamic:

- **Static** - The field control information is valid for all instances of a business object node, regardless of their state. For example, consider a field PAID in an invoice scenario that is always read-only since it can only be changed by an action, not directly by the consumer after an update procedure.
More on this: [Static Field Control \[page 190\]](#)
- **Dynamic** - The field (and action) control information depends on the state of the node instances. For example, the field COMMENTS should be read-only if the invoice's PAID attribute is set to true.
More on this:
 - [Dynamic Field and Entity Control \[page 194\]](#)
 - [Dynamic Action Control \[page 201\]](#)

⚠ Caution

If you use the **Referenced Data Source (RDS)** option for OData service creation, it is important that you always regenerate your gateway project in transaction **SEGW** after you add, change, or remove field or action control in the consumer views contained in the project.

8.1.1 Static Field Control

Here we explain how you can implement static field control in CDS views.

Overview

In a typical transactional scenario, you have to implement so-called business object CDS views when modeling business data. Such views are required for business object generation on the basis of the BOPF framework. To provide static field control for your application, you have to add the appropriate CDS annotations in the corresponding business object view - in fact, for both the `VIEW` and the `ELEMENT` scope. Using annotations with the `VIEW` scope, you define the field control that is related to the whole entity. You thus specify at the business object node level whether each node instance is enabled for creation, update, or deletion. When you add annotations with the `ELEMENT` scope, you have the option to define the field control for each individual view field and specify the field control at the attribute level of the corresponding business object node. If no such field control annotations are explicitly added to the view, the default value (`@ObjectModel.readOnly: false`) is assumed.

Field control annotations can be restricted in the consumption view. This means, you can overwrite `read-only` value and change it from `false` (as defined in the BO view) to `true` (in the consumption view).

Defining Static Field Control in Business Object Views

Entity Control at Business Object's Node Instance Level

The following annotations are relevant at the CDS `view level`:

Annotation and Value	Effect
<code>@ObjectModel.createEnabled: true/false</code>	Specifies whether new instances of a business object node that correspond to the underlying CDS view can be created If this annotation has the value <code>true</code> , creation of new instances of business object nodes is allowed.
<code>@ObjectModel.updateEnabled: true/false</code>	Specifies whether existing instances of a business object node that corresponds to the underlying CDS view can be updated If this annotation has the value <code>true</code> , updating existing instances is allowed.
<code>@ObjectModel.deleteEnabled: true/false</code>	Specifies whether existing instances of a business object node that corresponds to the underlying CDS view can be deleted If this annotation has the value <code>true</code> , updating existing instances is allowed.

Example

The annotations mentioned above are used in a business object CDS view (in our example: ZDEMO_I_SalesOrder_TX) to define that instances of a business object node can be created and updated, but not deleted.

```
...
@ObjectModel.transactionalProcessingEnabled: true
@ObjectModel.writeActivePersistence: 'ZTAB_SO'

@ObjectModel.createEnabled: true
@ObjectModel.deleteEnabled: false
@ObjectModel.updateEnabled: true

define view ZDEMO_I_SalesOrder_TX
    as select from ztab_so as SalesOrder
    association...
{
    ...
}
```

Effect on the Generated Business Object (in case of BOPF BOs):

The business object CDS view from the sample code above is used for generating the corresponding sales order business object. As you can see in the figure below, the field control that is related to the whole entity is defined accordingly at the business object's node level by the properties Create/Update/Delete Enabled.

The screenshot shows the SAP Fiori Node Overview editor. At the top, there is a header bar with a magnifying glass icon and the text "Node Overview". Below the header, there is a section titled "General Information" with the sub-section "General Information of Node". The form contains the following fields:

Name: *	ZDEMO_I_SALESORDER_TX
Source CDS View:	ZDEMO_I_SALESORDER_TX
Description:	#GENERATED#
Persistent Structure: *	ZSDEMO_I_SALESORDER_TX_D
Transient Structure:	
Combined Structure: *	ZSDEMO_I_SALESORDER_TX
Combined Table Type: *	ZTDEMO_I_SALESORDER_TX
Database Table: *	ZTAB_SO

At the bottom of the editor, there are three checkboxes with a red oval around them:

- Create Enabled
- Update Enabled
- Delete Enabled

BO editor with properties of the business object node corresponding to the CDS view

Static Field Control at the Business Object's Attribute Level

The annotations (required) at **element level** are:

Annotation and Values	Effect
<code>@ObjectModel.readOnly: true/false</code>	If this annotation has the value <code>true</code> , the annotated element must not be updated by the consumer. The BOPF runtime rejects modification requests when updating fields that provide a value <code>true</code> .
<code>@ObjectModel.mandatory: true</code>	Defines that the element is mandatory If this annotation has value <code>true</code> , the field must be filled by the consumer when executing a modification.

Example

For the business object CDS view from the sales order example above, some field control annotations are used to restrict properties of particular fields. Here, the element `BusinessPartner` cannot be modified, whereas the value of the elements `CurrencyCode`, `GrossAmount`, `NetAmount`, and `BillingStatus` can be created or updated. The key element `SalesOrder` is mandatory, that is, the property must contain a value.

```
...
@ObjectModel.updateEnabled: true

define view ZDEMO_I_SalesOrder_TX
  as select from ztab_so as SalesOrder
  association...
{
  @ObjectModel.mandatory: true
  key SalesOrder.salesorder
    as SalesOrder,
  @ObjectModel.readOnly: true
  SalesOrder.businesspartner
    as BusinessPartner,
  @ObjectModel.readOnly: false
  SalesOrder.currencycode
    as CurrencyCode,
  @ObjectModel.readOnly: false
  SalesOrder.grossamount
    as GrossAmount,
  @ObjectModel.readOnly: false
  SalesOrder.netamount
    as NetAmount,
  @ObjectModel.readOnly: true
  SalesOrder.billingstatus
    as BillingStatus,
  ...
}
```

Effect to the Generated Business Object (in case of BOPF BOs):

The field control annotations restrict the attributes of the corresponding business object node. In the BO editor, these attributes are listed in the [Properties](#) tab.

Element Name	Enabled	ReadOnly	Mandatory
KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
PARENT_KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ROOT_KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SALESORDER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
BUSINESSPARTNER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
CURRENCYCODE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GROSSAMOUNT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
NETAMOUNT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BILLINGSTATUS	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Overview | Alternative Keys | **Properties** | Associations | Actions | Determinations

BO editor with individual attributes (Properties tab) of the business object node

Restricting Field Control at the Consumption View Level

You can use field control annotations in the consumption view to restrict the values for static field control that have been specified at the BO view level. In other words, you can overwrite the read-only value and change it from `false` (in the BO view) to `true` (in the consumption view).

→ Remember

CDS Rule: Remember to double-maintain the annotations that have the `VIEW` scope. In CDS views, only the annotations with `ELEMENT` scope are inherited from the business object view.

Example: Adding Annotations for Non-Editable Fields

For the sales order scenario above, the consumption view `ZDEMO_C_SalesOrder_TX` adds the annotation `@ObjectModel.readOnly: true` to mark the fields `GrossAmount` and `NetAmount` as non-editable for the consumer – as not designated by the BO view.

```

...
-- Repeated annotations from BO view - VIEW scope
@ObjectModel.createEnabled: true
@ObjectModel.deleteEnabled: false
@ObjectModel.updateEnabled: true

-- Consumption view
define view ZDEMO_C_SalesOrder_TX

    as select from ZDEMO_I_SalesOrder_TX as Document
    ...
{
    @UI...
    key Document.SalesOrder,           -- inherits field control value

```

```

@UI...
Document.BusinessPartner,          -- inherits field control value

@UI...
Document.CurrencyCode,           -- inherits field control value

@UI...
@ObjectModel.readOnly: true      -- restricts to non-editable field
Document.GrossAmount,

@UI...
@ObjectModel.readOnly: true      -- restricts to non-editable field
Document.NetAmount,

@UI...
Document.BillingStatus
}

```

Related Information

[Developing New Transactional Apps \[page 78\]](#)

[ObjectModel Annotations \[page 428\]](#)

8.1.2 Dynamic Field and Entity Control

In this chapter, we explain how you can define dynamic fields and entities (such as business object nodes) in CDS views using ABAP CDS annotations and then implement dynamic field and entity control using a BOPF determination.

Overview

Dynamic fields and entities allow you to control the visibility and changeability of fields or the entire business object nodes depending on the value selected from another field. In the context of the ABAP programming model for SAP Fiori, the field and entity control serve as a way to provide information to the OData service in a transactional scenario about how data has to be displayed for consumption on the SAP Fiori UI. In particular, when using BOPF business objects, the field and entity control information is dependent on the state of the node instances. In this context, the field and entity control can in this context relate to the individual properties (node attributes of the business object) or an entire entity (business object node).

Example

The following app manages invoices. Each invoice has three properties which are the invoice ID, the payment status, and the comments provided. In our example, the fields *Invoice ID* and *Comments* should be read-only if the invoice's *Paid* attribute is set to `true`.

General Information

*Invoice ID:
1,000,066

Paid:

Comments:
SOME COMMENTS

Save **Cancel**

The comments field is editable for invoices that are not paid

General Information

Invoice ID:
1,000,077

Paid:

Comments:
EDIT SOME COMMENTS HERE! read-only

Save **Cancel**

The ID and the comments fields are read-only for a paid invoice

Activities Relevant to Developers

In a typical transactional scenario, you must implement business object (BO) views when modeling business data. The following steps are then required to provide dynamic field (or entity) control to the data model:

1. Adding annotations for dynamic field (or entity) control in the BO view
2. Activating the data definition object that includes the CDS BO view
3. Implementing the (automatically) generated property determination `ACTION_AND_FIELD_CONTROL`.

1. Adding Annotations for Dynamic Field (and Entity) Control in a CDS Business Object View

To enable dynamic field control for your application, you must add the appropriate CDS annotations with the value 'EXTERNAL_CALCULATION' in the relevant business object view - either for the `VIEW` or the `ELEMENT` scope. Using annotations with the `VIEW` scope, you define the entity control that is related to the whole entity (business object node instance). You thus specify at the business object node level whether each node instance is enabled for update, or deletion. When you add annotations with the `ELEMENT` scope, you have the option to define the field control for each individual view field and specify the dynamic fields at the attribute level of the corresponding business object node.

Dynamic Entity Control at the Business Object Node Level (VIEW Scope)

The following CDS annotations (specific to transactional processing) are relevant at the CDS view level when defining dynamic entity control:

```
@ObjectModel: {
    transactionalProcessingEnabled: true,
    ...
    createEnabled:      true | false    -- no dynamic entity control
support for creation operations
    deleteEnabled:     true | false | 'EXTERNAL_CALCULATION'
    updateEnabled:    true | false | 'EXTERNAL_CALCULATION'
    ...
}
define view <ViewName>
    as select from <data_source>
{
...
}
```

Dynamic Field Control at the Business Object Attribute Level (ELEMENT Scope)

The following CDS annotations at the element level of the CDS data model are relevant when defining dynamic field control:

```
define view <ViewName>
    as select from <data_source>
{
    ...
    @ObjectModel.readOnly: true | false | 'EXTERNAL_CALCULATION'
    <view.element>

    ...
    @ObjectModel.mandatory: true | false | 'EXTERNAL_CALCULATION'
    <view.another_element>
    ...
}
```

→ Remember

Within a CDS view (BO view), you have the option to define both the dynamic field control (with the `ELEMENT` scope) and the dynamic entity control (with the `VIEW` scope). You can also add the dynamic field annotation with the value '`EXTERNAL_CALCULATION`' to multiple view elements.

2. Activating the Data Definition Object that Includes the Relevant CDS BO View

After you activate the data definition that defines the relevant CDS BO view, the BOPF framework automatically generates a property determination with the name ACTION_AND_FIELD_CONTROL for the business object in question. This determination is used to implement the field (or entity) control for each element (or each node property) that is annotated with the value 'EXTERNAL_CALCULATION'.

i Note

If the CDS BO View is included in a SEGW project with RDS, you have to regenerate the GW project to enable the feature control. Only then are the auxiliary fields delete_mc, invoiced_mc etc. available in the MPC class.

3. Implementing the Property Determination ACTION_AND_FIELD_CONTROL

The generated determination class that is assigned to the business object node serves as a code exit for implementing the BOPF determination contract. This contract requires a redefinition of the method /bobf/if_frw_determination~execute.

Listing: Generated skeleton of the BO-specific determination class

```
class ZCL_D_DEMO_I_INVOICE_TP_ACTION definition
  public
    inheriting from /BOBF/CL_LIB_D_SUPERCL_SIMPLE
    final
    create public .
  public section.
    methods /BOBF/IF_FRW_DETERMINATION~EXECUTE
      redefinition .
  protected section.
  private section.
ENDCLASS.
CLASS ZCL_D_DEMO_I_INVOICE_TP_ACTION IMPLEMENTATION.
  method /BOBF/IF_FRW_DETERMINATION~EXECUTE.
    endmethod.
ENDCLASS.
```

Implementation Steps – Overview

Implement the following steps in the source code within this execute method:

1. Retrieve data that is required for field or entity control calculation
The method call `io_read->retrieve()` is used to retrieve the data of the root node, and the method `io_read->retrieve_by_association()` is called to retrieve the data stored in an associated subnode.
2. Create an instance of the property class
The property class `/BOBF/CL_LIB_H_SET_PROPERTY` is used to create a property helper (object):

```
data(lo_property_helper) = new /bobf/cl_lib_h_set_property( io_modify =
  io_modify
```

```
is_context = is_ctx ).
```

3. Set the field or entity control information

This property class provides multiple methods for setting properties of BO artifacts such as actions, attributes, associations, or nodes.

The relevant methods for setting properties of BO attributes (fields) are:

Method	Description
SET_ATTRIBUTE_READ_ONLY	Sets the <i>read-only</i> property of an attribute
SET_ATTRIBUTE_ENABLED	Sets the <i>enabled</i> property of an attribute
SET_ATTRIBUTE_MANDATORY	Sets the <i>mandatory</i> property of an attribute

The relevant methods for setting properties of entire entities (BO nodes) are:

Method	Description
SET_NODE_UPDATE_ENABLED	Sets the <i>Update Enabled</i> property of a node
SET_NODE_DELETE_ENABLED	Sets the <i>Delete Enabled</i> property of a node
SET_NODESUBTREE_CHANGE_ENABLED	Sets the <i>Create/Update/Delete Enabled</i> property of a subnode
SET_NODESUBTREE_UPDATE_ENABLED	Sets the <i>Update Enabled</i> property of a subnode
SET_NODESUBTREE_CREATE_ENABLED	Sets the <i>Create Enabled</i> property of a subnode
SET_NODESUBTREE_DELETE_ENABLED	Sets the <i>Delete Enabled</i> property of a subnode

Example

1. Annotating the Dynamic Fields in the CDS Business Object View

The following example implements a simple CDS BO view for invoices. The semantics of the business object is specified by means of CDS annotations used for transactional processing. The view element `Paid` is annotated as a static field whereas the element `Comments` specifies a dynamic field.

Listing: BO view example

```
@AbapCatalog.sqlViewName: 'ZDEMO_I_INV_V'
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Invoices'
@ObjectModel: {
    -- Annotations for transactional processing
    semanticKey: 'Invoice',
    compositionRoot: true,
    transactionalProcessingEnabled: true,
    createEnabled: true,
```

```

        deleteEnabled: true,
        updateEnabled: true,
        writeActivePersistence: 'ZTAB_INV'
    }
define view ZDEMO_I_Invoice_TP
    as select from ztab_inv as Invoice
{
    key Invoice.invoiceid as InvoiceId,
    @ObjectModel.readOnly: false           -- for a static field
    Invoice.paid                          as Paid,
    @ObjectModel.readOnly: 'EXTERNAL_CALCULATION' -- for a dynamic field
    Invoice.comments                     as Comments
}

```

2. Activating the CDS Business Object View

The business object CDS view from the sample code above is used to generate an invoice business object. As you can see in the figure below, the static properties of each generated field are defined for the business object. Since the static properties can only be restricted at runtime, the *Read-Only* flag is not set for the related fields (*INVOICEID*, *PAID*, *COMMENTS*) in our example.

You can choose the *Properties* tab to view the properties of each element (attribute) related to the business object created.

Element Name	Enabled	ReadOnly	Mandatory	
KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
PARENT_KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
ROOT_KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
INVOICEID	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
PAID	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
COMMENTS	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Generated BO properties

The BOPF framework automatically adds a property determination *ACTION_AND_FIELD_CONTROL* to the invoice business object.

Determinations

Name	Implementation C...	Category
ACTION_AND_FIELD_CONTROL	ZCL_D_DEMO_I_INVC	Calculate prop...

Overview Alternative Keys Properties Associations Actions Determinations

Generated property determination ACTION_AND_FIELD_CONTROL

3. Implementing the Property Determination ACTION_AND_FIELD_CONTROL

```

method /BOBF/IF_FRW_DETERMINATION~EXECUTE.
  " The invoice's data is typed with BO node's combined table type
  data lt_invoice_data type ztdemo_i_invoice_tp.
  " (1) Retrieve the data of the invoice's node instance
  io_read->retrieve(
    EXPORTING
      iv_node  = is_ctx-node_key          " uid of the node instance
      it_key   = it_key                  " keys given to the determination
    IMPORTING
      et_data = lt_invoice_data).       " itab with invoice's node data
  " (2) Create a property helper object
  data(lo_property_helper) = new /bobf/cl_lib_h_set_property( io_modify =
  io_modify
                                is_context = is_ctx ).

  " (3) Set the attribute "comments" to read-only for all paid invoices
  loop at lt_invoice_data into data(ls_invoice_data)
    where paid = abap_true.

    lo_property_helper->set_attribute_read_only(
      iv_attribute_name = zif_demo_i_invoice_tp_c=>sc_node_attribute-
zdemo_i_invoice_tp-comments
      iv_key            = ls_invoice_data-key ).
  endloop.
endmethod.

```

Related Information

[Dynamic Action Control \[page 201\]](#)

8.1.3 Dynamic Action Control

In this chapter, we explain how you can provide dynamic action control by using BOPF property determinations.

Overview

From the BOPF perspective, field properties and action properties are very similar: they are both to be implemented in the same property determination `ACTION_AND_FIELD_CONTROL`. In both cases, the property helper class `/BOBF/CL_LIB_H_SET_PROPERTY` is used to set the relevant field or action control information. When using business objects, both the field and the action control information depends on the state of the node instances. Therefore, the action control can relate to an entire business object node or individual node attributes.

Example

The following example app is used to manage invoices. Users can change the payment status for individual invoices without having to switch to edit mode. The action *Set to PAID* is disabled for all paid invoices, since the action control information is related to the attribute *Paid*.

Invoices (3) Standard		
	Invoice ID	Paid
<input checked="" type="checkbox"/>	1,000,066	No
<input type="radio"/>	1,000,077	Yes
<input type="radio"/>	1,000,088	Yes

Action enabled

action enabled

Invoices (3) Standard		
	Invoice ID	Paid
<input type="radio"/>	1,000,066	No
<input checked="" type="checkbox"/>	1,000,077	Yes
<input type="radio"/>	1,000,088	Yes

Action disabled

action disabled

Activities Relevant to Developers

In a typical transactional scenario, you must implement business object (BO) views when modeling business data. Based on this, the following steps are then required to provide dynamic action control to the data model:

1. Creating, configuring, and implementing the BO-specific action - if you have not yet already done so.
2. Reactivating the business object - if necessary.
3. Implementing the determination class for the property determination `ACTION_AND_FIELD_CONTROL`.

1. Creating, Configuring, and Implementing the Action

Since this step is already described in detail in other documentation, we will confine ourselves to referring to the relevant topics.

Related Information

- [Adding a New BOPF Action \[page 96\]](#)
- [Enabling Actions for OData Consumption \[page 101\]](#)
-

2. (Re)Activating the Business Object

To accept the most recent changes to the business object, you must reactivate it. You can then use the generated property determination `ACTION_AND_FIELD_CONTROL` to implement the dynamic action control.

3. Implementing the Property Determination Class

The created determination class that belongs to the business object node serves as a code exit for implementing the BOPF determination contract. This contract requires a re-definition of the method `/BOBF/IF_FRW_DETERMINATION~EXECUTE`. In other words: Each determination in BOPF must implement the `execute` method of the determination interface `/BOBF/IF_FRW_DETERMINATION`.

Listing: Generated skeleton of the determination class

```
class ZCL_D_DEMO_I_INVOICE_TP_ACTION definition
  public
    inheriting from /BOBF/CL_LIB_D_SUPERCL_SIMPLE
    final
    create public .
  public section.
    methods /BOBF/IF_FRW_DETERMINATION~EXECUTE
      redefinition .
  protected section.
  private section.
ENDCLASS.
```

```

CLASS ZCL_D_DEMO_I_INVOICE_TP_ACTION IMPLEMENTATION.
  method /BOBF/IF_FRW_DETERMINATION~EXECUTE.
    endmethod.
  endclass.

```

Implementation Steps – Overview

Implement the following steps within this `execute` method:

1. Retrieve data that is required for action control calculation

The method call `io_read->retrieve()` is used to retrieve the data of the root node, whereas the method `io_read->retrieve_by_association()` is called to retrieve the data stored in associated subnode.

2. Create an instance of the property class

The property class `/BOBF/CL_LIB_H_SET_PROPERTY` is used to create a property helper (object):

```

data(lo_property_helper) = new /bobf/cl_lib_h_set_property( io_modify =
  io_modify
                                is_context = is_ctx ).

```

3. Set the action control information

This property class provides a suitable method for setting properties of the BO actions:

Method SET_ACTION_ENABLED

This method sets the enabled property of the action

Signature

<i>importing</i>	<code>IV_ACTION_KEY</code>	<i>type</i>	<code>/BOBF/CONF_KEY</code>
	<code>IV_KEY</code>	<i>type</i>	<code>/BOBF/CONF_KEY</code>
	<code>IV_VALUE</code>	<i>type</i>	<code>ABAP_BOOL default ABAP_TRUE</code>

Parameters

Parameter	Description
<code>IV_ACTION_KEY</code>	Key of the BOPF action for which the property is to be set
<code>IV_KEY</code>	Key of the business object node instance for which the property is to be set
<code>IV_VALUE</code>	New value of this property The value <code>ABAP_TRUE</code> / <code>(ABAP_FALSE)</code> causes the action to be enabled/(disabled).

Example

The following example implements a simple app for managing invoices.

1. Creating, Configuring, and Implementing the Action

To define the behavior of the invoice business object at runtime, we need to add a corresponding action to the BO node. For our sample scenario, we are going to create an action SET_PAID to change the status of the invoices to PAID after a buyer pays for a product.

To make the action available for consumption in an OData service, it is necessary to configure the exporting type. **For further information, see:** [Enabling Actions for OData Consumption \[page 101\]](#)

Action Overview

General Information

Name: * SET_PAID

Description: set to paid action

Instance Multiplicity: Multiple Node Instances

Implementation Details

Implementation Class: * ZCL_DEMO_A_SET_PAID

Parameter Structure:

Exporting Type: Node

Exporting Parameter BO: ZDEMO_I_INVOICE_TP

Exporting Parameter Node: ZDEMO_I_INVOICE_TP

Exporting Multiplicity: 1

Action SET_PAID with the exporting type configuration

Each action in BOPF must implement the execute method of the action interface /BOBF/IF_FRW_ACTION. The listing below is used to implement the SET_PAID action.

Listing: Implemented SET_TO_PAID action

```
method /BOBF/IF_FRW_ACTION~EXECUTE.  
  " The invoice's data is typed with BO node's combined table type  
  data lt_invoice_data type ztdemo_i_invoice_tp.  
  " READING BO data -----  
  " Retrieve the data of the requested node instance  
  io_read->retrieve(  
    EXPORTING  
      iv_node      = is_ctx-node_key  
      it_key       = it_key  
    IMPORTING  
      et_data      = lt_invoice_data ).  
  " WRITING BO data -----  
  LOOP AT lt_invoice_data ASSIGNING FIELD-SYMBOL(<s_invoice>).  
  " Set the attribute paid to new value  
  <s_invoice>-paid = abap_true. " PAID  
  " Update the changed data (payment status) of the relevant node instance
```

```

io_modify->update(
    iv_node           = is_ctx-node_key
    iv_key            = <s_invoice>-key
    iv_root_key       = <s_invoice>-root_key
    is_data           = REF #( <s_invoice>-node_data )
    it_changed_fields = VALUE #(
        ( zif_demo_i_invoice_tp_c=>sc_node_attribute-zdemo_i_invoice_tp-paid )
    )
).
ENDLOOP.
endmethod.

```

2. Activating the Business Object

To accept the most recent changes to your invoice business object, you must activate it.

3. Implementing the Property Determination Class

Name	Implementation C...	Category
ACTION_AND_FIELD_CONTROL	ZCL_D_DEMO_I_INVC	Calculate prop...

Determination ACTION_AND_FIELD_CONTROL with the Calculate properties category

This property determination class calls the `set_action_enabled` method for setting the new property of the action `SET_PAID`.

Listing: Implemented property determination

```

CLASS ZCL_D_DEMO_I_INVOICE_TP_ACTION IMPLEMENTATION.
method /BOBF/IF_FRW_DETERMINATION~EXECUTE.
" The invoice's data is typed with BO node's combined table type
data lt_invoice_data type ztdemo_i_invoice_tp.
data lv_action_enabled type abap_bool.
" (1) Retrieve the data of the invoice's node instance
io_read->retrieve(
    EXPORTING
        iv_node   = is_ctx-node_key
        it_key    = it_key
    IMPORTING
        et_data = lt_invoice_data .
" (2) Create a property helper object
data lo_property_helper type ref to /bobf/cl_lib_h_set_property.
create object lo_property_helper
    EXPORTING
        io_modify = io_modify
        is_context = is_ctx.
data(lo_property_helper) = NEW /bobf/cl_lib_h_set_property( io_modify =
    io_modify
                                is_context =
                                is_ctx ).
" (3) Action not enabled when invoice is set to PAID
loop at lt_invoice_data ASSIGNING FIELD-SYMBOL(<s_invoice>).

```

```

if <s_invoice>-paid = abap_true.
  lv_action_enabled = abap_false.
else.
  lv_action_enabled = abap_true.
endif.
lo_property_helper->set_action_enabled(
  iv_action_key = zif_demo_i_invoice_tp_c=>sc_action-
zdemo_i_invoice_tp-set_paid
  iv_key         = <s_invoice>-key
  iv_value       = lv_action_enabled ).
endloop.
endmethod.
ENDCLASS.
```

Related Information

[Dynamic Field and Entity Control \[page 194\]](#)

[Determinations for CDS-Based Business Objects \[page 63\]](#)

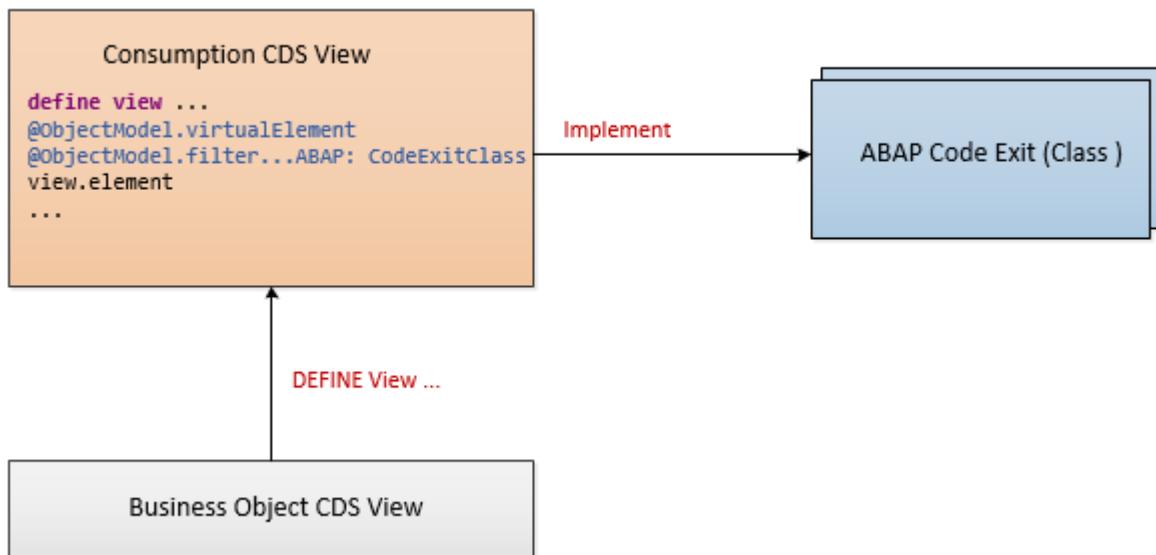
8.2 Using Virtual Elements in CDS

This topic addresses the use of virtual elements in CDS in the context of SAP Fiori application development for transactional use cases.

What Are Virtual Elements, and Why Do I Need Them?

For some business use cases, it may be necessary to add new fields to the data model of an application and to calculate their values using ABAP resources. Virtual elements come into play if these fields are not provided as part of the original persistence data model or cannot be easily integrated in CDS so that their values can be calculated directly on SAP HANA.

Virtual elements represent transient fields in business applications. They are defined at the level of CDS consumption views as additional elements within the `SELECT` list using specific `@DataModel` annotations. However, the calculation or further processing of their values is carried out by means of ABAP classes that implement the specific code exit interfaces provided for this purpose.



Definition of virtual elements in CDS and implementation of ABAP code exit

Use Cases for Virtual Elements

You can use the concept of virtual elements to implement the following use cases:

- Calculating values of fields that are not part of a persistence model
- Filtering of calculated field values
- Sorting of calculated field values

Example

The UI of the following sales order app provides an additional field `GrossAmountWithDiscount` for each sales order item, which is not part of the persistence data model. The values of this field are calculated using the ABAP code exit API for handling virtual elements. In our example, the values of this virtual field provide a reduced gross amount of each sales order item. The discount depends on the gross amount volume: If the gross amount is greater than EUR 1,000.00, the discount is calculated, for example, by the formula:

$$(\text{ConvertedGrossAmount} - 1,000.00) * 0.1$$
. This leaves a remaining amount to pay of $1,000.00 + (\text{ConvertedGrossAmount} - 1,000.00) * 0.9$.

Converted Gross Amount	Gross Amount with Discount	Item Position	Product ID	Sales Order ID	
547.40 EUR	547.40 EUR	10	HT-1030	500000007	>
Total Gross Amount: 547.40 EUR					
1,615.00 EUR	1,553.50 EUR	20	HT-1031	500000007	>
Total Gross Amount: 1,017.45 GBP					
410.55 EUR	410.55 EUR	30	HT-1032	500000007	>
Total Gross Amount: 410.55 EUR					
1,010.23 EUR	1,009.21 EUR	40	HT-1035	500000007	>
Total Gross Amount: 949.62 USD					
725.82 EUR	725.82 EUR	50	HT-1036	500000007	>

List with sales order items that provides a calculated field for the reduced gross amount

Activities Relevant to Developers

The following steps are relevant for providing virtual elements to the CDS data model:

1. Adding the virtual element and corresponding annotations in the relevant consumption CDS view.
More on this: [Adding Annotations for Virtual Elements \[page 208\]](#)
2. Creating and implementing ABAP classes that serve as a code exit for implementing the virtual element contract.
More on this: [Implementing ABAP Code Exits for Virtual Elements \[page 212\]](#)
In case of implementing filters for virtual elements, you need to create conditions for the substitution of filters on virtual elements.
More on this: [Creating Conditions with the Simple Condition Factory \[page 218\]](#)
3. Creating exception classes for handling application-specific exceptions.
More on this: [Creating Application-Specific Exception Classes \[page 231\]](#)

8.2.1 Adding Annotations for Virtual Elements

When you define code exits, the following CDS annotations are relevant at the ELEMENT level of the CDS data model:

CDS annotations specific to the calculation of field values (use case 1)

For the calculation of the field values in CDS views, the following @ObjectModel annotations are required:

i Note

When defining a virtual element, the data type of the element must be specified.

```
define view <CdsConsumptionView>
```

```

        as select from <data_source>
{
    ...
    @ObjectModel.readOnly: true
    @ObjectModel.virtualElement
    @ObjectModel.virtualElementCalculatedBy: 'ABAP:<code_exit_class>'
    cast( '' as <dtype> preserving type) as <view.element>
    ...
}

```

CDS annotations specific to field filtering (use case 2)

For the filtering of field values, the relevant view element must use the `@ObjectModel.filter` annotation:

```

define view <CdsConsumptionView>
    as select from <data_source>
{
    ...
    @ObjectModel.filter.transformedBy: 'ABAP:<code_exit_class>'
    cast( '' as <dtype> preserving type) as <view.element>
    ...
}

```

CDS annotations specific to field sorting (use case 3)

For the sorting of field values, the relevant view element must use the `@ObjectModel.sort` annotation:

```

define view <CdsConsumptionView>
    as select from <data_source>
{
    ...
    @ObjectModel.sort.transformedBy: 'ABAP:<code_exit_class>'
    cast( '' as <dtype> preserving type) as <view.element>
    ...
}

```

Sub Annotation of <code>@ObjectModel</code>	Meaning
<code>virtualElement</code>	<p><i>Semantics:</i></p> <p>Defines a virtual element within the CDS view definition</p> <p>Its value is calculated by the ABAP class specified by the annotation <code>@ObjectModel.virtualElementCalculatedBy</code>.</p> <p>i Note</p> <p>Virtual elements may only be defined within the ELEMENT scope at CDS consumption view level.</p>

Sub Annotation of @ObjectModel	Meaning
<code>virtualElementCalculatedBy : 'ABAP:<code_exit_class>'</code>	<p><i>Semantics:</i></p> <p>Defines the ABAP class that calculates the value of the virtual element</p> <p>This annotation references the technical name of the class <code_exit_class> to be used for implementing the code exit interface IF_SADI_EXIT_CALC_ELEMENT_READ.</p> <p>This annotation requires that the view element in question is also annotated with @ObjectModel.virtualElement: true.</p> <p><i>Engine Behavior:</i></p> <p>ABAP runtime invokes the specified function (ABAP code exit class) for calculating the value of the virtual element.</p>
<code>filter.transformedBy: 'ABAP:<code_exit_class>'</code>	<p><i>Semantics:</i></p> <p>Defines the ABAP class that transforms the filter criteria specified for the annotated element to the filter criteria of other elements</p> <p>This transformation class is assigned by the technical name of the ABAP class <code_exit_class> that implements the code exit interface IF_SADI_EXIT_FILTER_TRANSFORM.</p> <p>Virtual fields are exposed as <i>filterable</i> through OData if they have specified such a transformation class.</p> <p>Note</p> <p>This annotation may be used only for virtual elements and only at CDS consumption view level.</p> <p><i>Engine Behavior:</i></p> <p>SADL framework exposes the annotated element as <i>filterable</i> and invokes the given transformation class at runtime.</p>

Sub Annotation of @ObjectModel	Meaning
<pre>sort.transformedBy: 'ABAP:<code_exit_class>'</pre>	<p><i>Semantics:</i></p> <p>Defines the ABAP class that transforms the sort criteria specified for the annotated view element to sort criteria of other view elements</p> <p>This transformation class is assigned by the technical name of the ABAP class <code_exit_class> that implements the code exit interface IF_SADL_EXIT_SORT_TRANSFORM.</p> <p>Virtual fields are exposed as <i>sortable</i> through OData if they have specified such a transformation function.</p> <p>i Note</p> <p>This annotation may be used only for virtual fields and only at CDS consumption view level.</p> <p><i>Engine Behavior:</i></p> <p>SADL framework exposes the annotated field as <i>sortable</i> and invokes the given transformation class at runtime.</p>

Example

The source code below defines a simple CDS view ZDEMO_C_SOrderItem_VF for processing sales order items. This view is implemented as a consumption view with the predefined view SEPM_I_SalesOrderItem_E (that originates from the EPM demo application) that serves as the data source. The select list includes a virtual element GrossAmountWithDiscount, which is not foreseen as a part of the persistence data model, for each sales order item. The values of the corresponding field are calculated using the code exit ABAP class ZCL_CALCULATION_DISCOUNT. Additional classes (ZCL_FILTER_DISCOUNT and ZCL_SORT_DISCOUNT) are used to implement the filtering and sorting of the calculated field. In our example, the values of this calculated field provide a reduced gross amount of each sales order item. In addition, we add an additional field ConvertedGrossAmount to provide the gross amount in EUR as the target currency. For this purpose, the currency conversion procedure is included in the source code of the CDS view.

Listing: CDS consumption that includes virtual elements

```
@AbapCatalog.sqlViewName: 'ZDEMO_SOI_VF'  
@AccessControl.authorizationCheck: #NOT_REQUIRED  
@EndUserText.label: 'Sales Order Item - Virtual Fields'  
  
@Metadata.allowExtensions: true  
  
@Search.searchable : true  
  
@OData.publish: true  
  
define view ZDEMO_C_SOrderItem_VF  
    as select from SEPM_I_SalesOrderItem_E as Item  
{  
    @Search.defaultSearchElement: true  
    key Item.SalesOrder  
        as SalesOrder,  
    key Item.SalesOrderItem  
        as ItemPosition,
```

```

Item.Product                               as Product,
@Semantics.currencyCode: true
Item.TransactionCurrency                  as TransactionCurrency,
@Semantics.amount.currencyCode: 'TransactionCurrency'
Item.GrossAmountInTransacCurrency        as GrossAmount,
-- Virtual element
@ObjectModel.readOnly: true
@ObjectModel.virtualElement:true
@ObjectModel.virtualElementCalculatedBy:      'ABAP:ZCL_CALCULATION_DISCOUNT'
@ObjectModel.filter.transformedBy:            'ABAP:ZCL_FILTER_DISCOUNT'
@ObjectModel.sort.transformedBy:              'ABAP:ZCL_SORT_DISCOUNT'
cast(0 as abap.curr( 15, 2 ) )           as GrossAmountWithDiscount,
-- Currency conversion begin
@Semantics.currencyCode: true
cast( 'EUR' as abap.cuky )                as TargetCurrency,
-- Gross Amount in EUR
@Semantics.amount.currencyCode: 'TargetCurrency'

CURRENCY_CONVERSION(
amount          => Item.GrossAmountInTransacCurrency,
source_currency => Item.TransactionCurrency,
target_currency => cast( 'EUR' as abap.cuky ),
exchange_rate_date => cast( '20170101' as abap.dats ),
error_handling    => 'SET_TO_NULL' )   as ConvertedGrossAmount
-- Currency conversion end
}

```

8.2.2 Implementing ABAP Code Exits for Virtual Elements

Use Case 1: Calculating Values for Virtual Elements

Prerequisites

The calculation of field values requires that:

- The following virtual element annotations are added to the corresponding element at the level of the CDS consumption view:

```

@ObjectModel.readOnly: true
@ObjectModel.virtualElement: true
@ObjectModel.virtualElementCalculatedBy: 'ABAP:<code_exit_class>'
```

i Note

The annotated view element must not represent a key field.

- An ABAP code exit class is created for implementing the code exit interface

```
IF_SADL_EXIT_CALC_ELEMENT_READ.
```

Implementation Steps

The code exit contract for calculation of the virtual value requires the implementation of both the methods GET_CALCULATION_INFO and CALCULATE of the code exit interface IF_SADL_EXIT_CALC_ELEMENT_READ.

For further information, see: [Interface IF_SADL_EXIT_CALC_ELEMENT_READ \[page 232\]](#).

• Example

The method GET_CALCULATION_INFO provides the list of fields required for data calculation:

- The field CONVERTEDGROSSAMOUNT saves the values of the gross amount for sales order items calculated in EUR (target currency).
- The field GROSSAMOUNTWITHDISCOUNT is used to save the calculated gross amount values under consideration of specific discount rules.

As shown in the listing below, the method GET_CALCULATION_INFO raises a predefined exception entity_not_known that is passed to the parameter TEXTID of the instance constructor of the application-specific exception class ZCX_CALC_EXIT.

See also: [Creating Application-Specific Exception Classes \[page 231\]](#)

The actual calculation of the reduced gross amount is implemented within the method calculate. The discount depends on the gross amount volume and starts for a gross amount greater than 1,000.00 EUR. The customer receives a 10% discount on the delta between the gross amount and 1,000.00 EUR. The gross amount with the discount is calculated with the formula: 1,000.00 + (ConvertedGrossAmount - 1,000.00) * 0.9.

Listing: Calculation class ZCL_CALCULATION_DISCOUNT

```
CLASS zcl_calculation_discount DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
  INTERFACES:
    if_sadl_exit_calc_element_read.

  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS zcl_calculation_discount IMPLEMENTATION.

METHOD if_sadl_exit_calc_element_read~get_calculation_info.
  IF iv_entity <> 'ZDEMO_C_SORDERITEM_VF'.
    RAISE EXCEPTION TYPE zcx_calc_exit EXPORTING textid =
      zcx_calc_exit=>entity_not_known.
  ENDIF.
  IF line_exists( it_requested_calc_elements[ table_line =
    'GROSSAMOUNTWITHDISCOUNT' ] ).
    APPEND 'CONVERTEDGROSSAMOUNT' TO et_requested_orig_elements.
  ENDIF.
ENDMETHOD.

METHOD if_sadl_exit_calc_element_read~calculate.
  CHECK NOT it_original_data IS INITIAL.
```

```

DATA lt_calculated_data TYPE STANDARD TABLE OF zdemo_c_sorderitem_vf WITH
DEFAULT KEY.
MOVE-CORRESPONDING it_original_data TO lt_calculated_data.

LOOP AT lt_calculated_data ASSIGNING FIELD-SYMBOL(<ls_calculated_data>).
  <ls_calculated_data>-grossamountwithdiscount =
    COND #( WHEN <ls_calculated_data>-convertedgrossamount > 1000
    THEN 1000 + ( <ls_calculated_data>-convertedgrossamount - 1000 ) *
'0.9'
    ELSE <ls_calculated_data>-convertedgrossamount ).
ENDLOOP.
MOVE-CORRESPONDING lt_calculated_data TO ct_calculated_data.

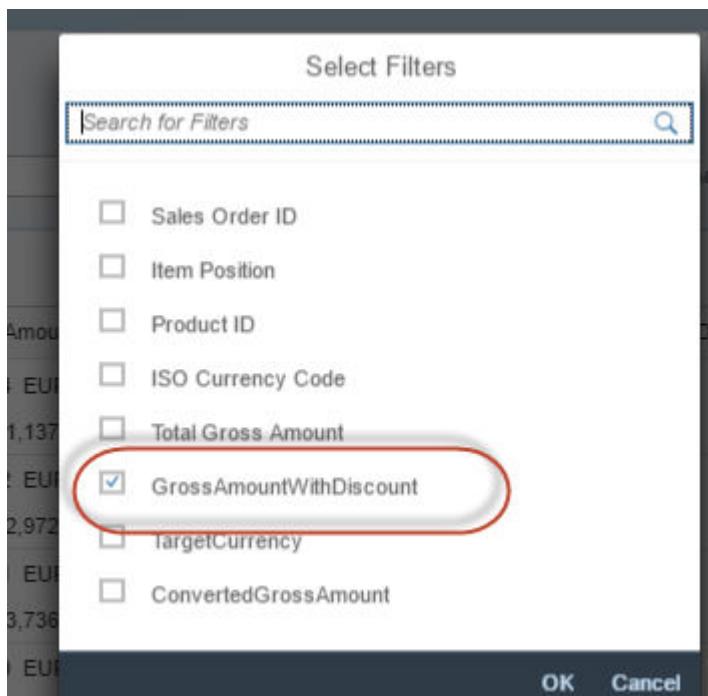
ENDMETHOD.

ENDCLASS.

```

Use Case 2: Implementing Filtering for Virtual Elements

Preview



Adding a filter for calculated field values

Converted Gross Amount	Gross Amount with Discount	Item Position	Product ID	Sales Order ID
1,137.64 EUR	1,123.88 EUR	10	HT-1000	500000000

Total Gross Amount: 1,137.64 EUR

Added filter for calculated field values on the Fiori UI

Prerequisites

The filter transformation requires that:

- The following filter annotation is added to the corresponding element at the level of the CDS consumption view:
@ObjectModel.filter.transformedBy: 'ABAP:<code_exit_class>'
- Remember**

The annotated view element must not represent a key field.
- An ABAP class is created for implementing the code exit interface `IF_SADL_EXIT_FILTER_TRANSFORM`.

Implementation Steps

The code exit contract for filter transformation requires the implementation of the method `MAP_ATOM` of the code exit interface `IF_SADL_EXIT_FILTER_TRANSFORM`.

i Note

The substitution of the condition with the method `MAP_ATOM` must not include virtual elements.

For further information, see: [Interface IF_SADL_EXIT_FILTER_TRANSFORM \[page 234\]](#).

• Example

The implementation of the method `MAP_ATOM` in the listing below demonstrates how you can transform filter conditions specified for the annotated (virtual) view element `GROSSAMOUNTWITHDISCOUNT` to filter criteria of other view elements. In other words, we have to replace the filter condition on the calculated field `GROSSAMOUNTWITHDISCOUNT` with a condition for the original persistent field `CONVERTEDGROSSAMOUNT`.

For further information about building complex conditions, see [Creating Conditions with the Simple Condition Factory \[page 218\]](#).

Listing: Filter transformation class ZCL_FILTER_DISCOUNT

```
CLASS zcl_filter_discount DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .
  PUBLIC SECTION.
    INTERFACES:
      if_sadl_exit_filter_transform.
  PROTECTED SECTION.
```

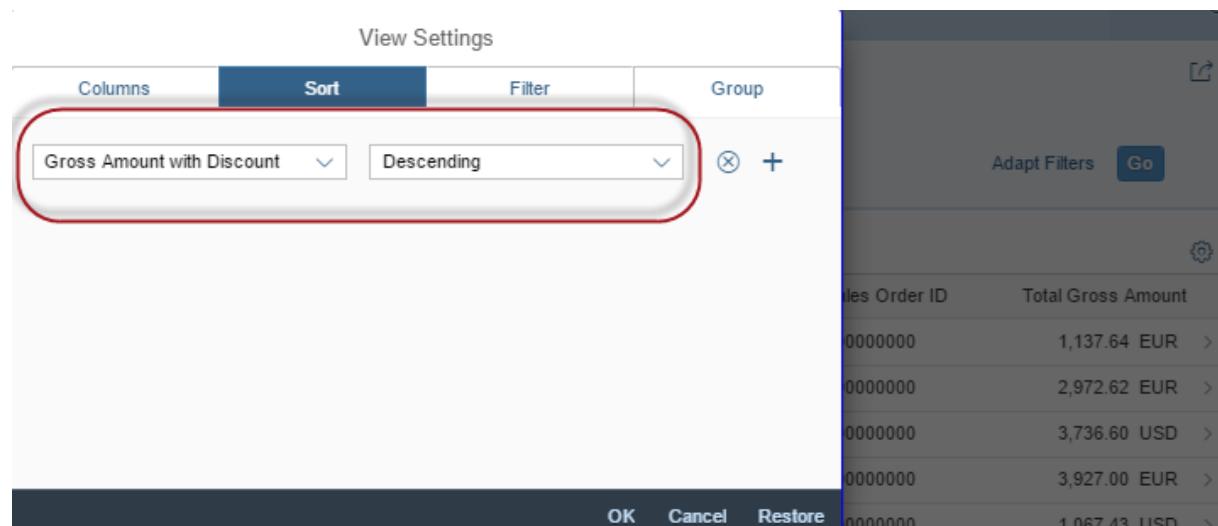
```

PRIVATE SECTION.
ENDCLASS.
CLASS zcl_filter_discount IMPLEMENTATION.
METHOD if_sadl_exit_filter_transform~map_atom.
  IF iv_element <> 'GROSSAMOUNTWITHDISCOUNT'.
    RAISE EXCEPTION TYPE zcx_filter_exit EXPORTING textid =
  zcx_filter_exit=>element_not_expected.
  ENDIF.
  DATA(lo_cfac) =
  cl_sadl_cond_prov_factory_pub=>create_simple_cond_factory( ).
  DATA(amount) = lo_cfac->element( 'CONVERTEDGROSSAMOUNT' ).
  DATA(lv_originalvalue) = 1000 + ( iv_value - 1000 ) / '0.9'.
CASE iv_operator.
  WHEN if_sadl_exit_filter_transform~co_operator-equals.
    ro_condition = amount->less_than( 1000 )->and( amount-
  >equals( iv_value )
    )->or( amount->greater_than( 1000 )->and( amount-
  >equals( lv_originalvalue ) ) .
  WHEN if_sadl_exit_filter_transform~co_operator-less_than.
    ro_condition = amount->less_than( 1000 )->and( amount-
  >less_than( iv_value )
    )->or( amount->greater_than( 1000 )->and( amount-
  >less_than( lv_originalvalue ) ) .
  WHEN if_sadl_exit_filter_transform~co_operator-greater_than.
    ro_condition = amount->less_than( 1000 )->and( amount-
  >greater_than( iv_value )
    )->or( amount->greater_than( 1000 )->and( amount-
  >greater_than( lv_originalvalue ) ) .
  WHEN if_sadl_exit_filter_transform~co_operator-is_null.
    ro_condition = amount->is_null( ) .
  WHEN if_sadl_exit_filter_transform~co_operator-covers_pattern.
    RAISE EXCEPTION TYPE zcx_filter_exit.
ENDCASE.
ENDMETHOD.

```

Use Case 3: Implementing Sorting for Virtual Elements

Preview



The screenshot shows a 'View Settings' dialog for a Fiori application. The 'Sort' tab is selected, highlighting a dropdown menu where 'Gross Amount with Discount' is chosen and set to 'Descending'. Below this, there are 'OK', 'Cancel', and 'Restore' buttons.

Sales Order ID	Total Gross Amount
0000000	1,137.64 EUR >
0000000	2,972.62 EUR >
0000000	3,736.60 USD >
0000000	3,927.00 EUR >
0000000	1,067.43 USD >

Settings for sorting based on the calculated field

Converted Gross Amount	Gross Amount with Discount	Item Position	Product ID	Sales Order ID	Total Gross Amount
5,040.18 EUR	4,636.16 EUR	70	HT-1011	500000000	547,162 JPY >
4,757.62 EUR	4,381.86 EUR	60	HT-1010	500000000	4,757.62 EUR >
3,975.11 EUR	3,677.60 EUR	30	HT-1002	500000000	3,736.60 USD >
3,927.00 EUR	3,634.30 EUR	40	HT-1003	500000000	3,927.00 EUR >
2,972.62 EUR	2,775.36 EUR	20	HT-1001	500000000	2,972.62 EUR >
1,615.00 EUR	1,553.50 EUR	90	HT-1031	500000000	1,017.45 GBP >
1,231.65 EUR	1,208.49 EUR	100	HT-1032	500000000	1,231.65 EUR >
1,137.64 EUR	1,123.88 EUR	10	HT-1000	500000000	1,137.64 EUR >
1,135.56 EUR	1,122.00 EUR	50	HT-1007	500000000	1,067.43 USD >
547.40 EUR	547.40 EUR	80	HT-1030	500000000	547.40 EUR >

Sorting calculated field values on the Fiori UI

Prerequisites

The sorting of field values requires that:

- The following sort annotation is added to the corresponding element at the level of the CDS consumption view:
`@ObjectModel.sort.transformedBy: 'ABAP:<code_exit_class>'`
- Remember
 The annotated view element must not represent a key field.
- An ABAP class is created for implementing the code exit interface `IF_SADL_EXIT_SORT_TRANSFORM`.

Implementation Steps

The code exit contract for sort transformation requires the implementation of the method `MAP_ELEMENT` of the code exit interface `IF_SADL_EXIT_SORT_TRANSFORM`.

For further information, see: [Interface IF_SADL_EXIT_SORT_TRANSFORM \[page 235\]](#)

Example

Listing: Sorting class `ZCL_SORT_DISCOUNT`

```

CLASS zcl_sort_discount DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .
  PUBLIC SECTION.
    INTERFACES:
      if_sadl_exit_sort_transform.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS zcl_sort_discount IMPLEMENTATION.
  METHOD if_sadl_exit_sort_transform~map_element.
    IF iv_element <> 'GROSSAMOUNTWITHDISCOUNT'.
      RAISE EXCEPTION TYPE zcx_sorting_exit EXPORTING textid =
        zcx_sorting_exit=>element_not_expected.
    ENDIF.
  
```

```

APPEND value #( name = `CONVERTEDGROSSAMOUNT` ) TO et_sort_elements.
ENDMETHOD.
ENDCLASS.
```

8.2.2.1 Creating Conditions with the Simple Condition Factory

Complex conditions can be built with the simple condition factory. This factory concatenates expressions with relational and logical operators; both of them represented by methods.

Context

Filtering on virtual elements requires the transformation of the given filter condition into a corresponding condition that does not use any virtual elements for the filter. The implementation of the substituting condition is done with the method `IF_SADL_EXIT_FILTER_TRANSFORM~MAP_ATOM`.

The construction of this condition is most easily done with the simple condition factory created by the method `CREATE_SIMPLE_COND_FACTORY`. This static method is called by the class `CL_SADL_COND_PROV_FACTORY_PUB`. You can build complex conditions with the simple condition factory by concatenating them with relational and logical operators.

Procedure

- Create a simple condition factory by calling the static method `CREATE_SIMPLE_COND_FACTORY` on the public condition provider class `CL_SADL_COND_PROV_FACTORY_PUB`.

```
DATA(lo_cfac) = cl_sadl_cond_prov_factory_pub=>create_simple_cond_factory( ).
```

The method returns a condition factory (a reference variable of type `IF_SADL_SIMPLE_COND_FACTORY` which defines the methods `TRUE`, `FALSE` and `ELEMENT`).

- Use the condition factory to call one of the methods of the interface `IF_SADL_SIMPLE_COND_FACTORY`.

For further information; see: [Interface `IF_SADL_SIMPLE_COND_FACTORY` \[page 222\]](#).

Method	Description
<code>element</code>	Use this method to initialize a condition. It defines the first operand represented by a non-virtual element of the CDS view. Methods of relational operators are executed on its returning value <code>ro_element</code> by using methods of relational operators.

Method	Description
	For the importing parameter <code>iv_element</code> , enter the name of the element that represents the operand 'FIELD1' in the condition. <pre>lo_element = lo_cfac->element('FIELD1').</pre>
	Continue with step 3.
<code>true</code>	Use this method to set the condition to be always true. <pre>ro_condition = lo_cfac->true().</pre>
	To concatenate this condition with other conditions continue with step 4.
<code>false</code>	Use this method to set the condition to be always false. <pre>ro_condition = lo_cfac->false().</pre>
	To concatenate this condition with other conditions continue with step 4.

3. In case you used the method `ELEMENT` in the previous step, choose one of the methods for relational operators to set the operands `FIELD1` (`iv_element`) and `X` (`iv_value`) into relation with each other.

For further information, see [Interface IF_SADL_SIMPLE_COND_ELEMENT \[page 223\]](#).

Relational Operator	Description
<code>equals</code>	Use this method to build an expression like <code>FIELD1 = 72</code> . For the importing parameter <code>iv_value</code> , enter the reference value for the relation (for example a numeric value like 72). <pre>ro_condition = lo_cfac->element('FIELD1')- >equals(72).</pre>
<code>not_equals</code>	Use this method to build a condition like <code>FIELD1 ≠ 72</code> . For the importing parameter <code>iv_value</code> , enter the reference value for the relation (for example a numeric value like 72). <pre>ro_condition = lo_cfac->element('FIELD1')- >not_equals(72).</pre>
<code>less_than</code>	Use this method to build a condition like <code>FIELD1 < 72</code> . For the importing parameter <code>iv_value</code> , enter the reference value for the relation (for example a numeric value like 72). <pre>ro_condition = lo_cfac->element('FIELD1')- >less_than(72).</pre>
<code>greater_than</code>	Use this method to build a condition like <code>FIELD1 > 72</code> .

Relational Operator	Description
	<p>For the importing parameter <code>iv_value</code>, enter the reference value for the relation (for example a numeric value like 72).</p> <pre>ro_condition = lo_cfac->element('FIELD1')- >greater_than(72).</pre>
<code>less_than_or_equals</code>	<p>Use this method to build a condition like <code>FIELD1 ≤ 72</code>.</p> <p>For the importing parameter <code>iv_value</code>, enter the reference value for the relation (for example a numeric value like 72).</p> <pre>ro_condition = lo_cfac->element('FIELD1')- >less_than_or_equals(72).</pre>
<code>greater_than_or_equals</code>	<p>Use this method to build a condition like <code>FIELD1 ≥ 72</code>.</p> <p>For the importing parameter <code>iv_value</code>, enter the reference value for the relation (for example a numeric value like 72).</p> <pre>ro_condition = lo_cfac->element('FIELD1')- >greater_than_or_equals(72).</pre>
<code>covers_pattern</code>	<p>Use this method to build a condition like <code>FIELD1 covers the pattern of ABCDEF</code>.</p> <p>For the importing parameter <code>iv_value</code>, enter the reference value for the relation (for example a string value like <code>ABC*</code>).</p> <pre>ro_condition = lo_cfac->element('FIELD1')- >covers_pattern('ABC*').</pre>
<code>not_covers_pattern</code>	<p>Use this method to build a condition like <code>FIELD1 does not cover the pattern of ABCDEF</code>.</p> <p>For the importing parameter <code>iv_value</code>, enter the reference value for the relation (for example a string value like <code>*DEF</code>).</p> <pre>ro_condition = lo_cfac->element('FIELD1')- >not_covers_pattern('*DEF').</pre>
<code>is_null</code>	<p>Use this method to build a condition like <code>FIELD1 is null</code>.</p> <p>This method does not have an importing parameter.</p> <pre>ro_condition = lo_cfac->element('FIELD1')->is_null.</pre>
<code>is_not_null</code>	<p>Use this method to build a condition like <code>FIELD1 is not null</code>.</p> <p>This method does not have an importing parameter.</p> <pre>ro_condition = lo_cfac->element('FIELD1')->is_not_null.</pre>
<code>is_between</code>	<p>Use this method to build a condition like <code>72 ≤ FIELD1 ≤ 120</code>.</p>

Relational Operator	Description
	<p>For the importing parameter <code>iv_low</code> and <code>iv_high</code>, enter the values that provide the lower limit and upper limit (for example 120) of the range (for example numeric values like 72 and 120).</p> <pre>ro_condition = lo_cfac->element('FIELD1')- >is_between(iv_low = 72 iv_high = 120).</pre>
<code>is_not_between</code>	<p>Use this method to build a condition like $not 72 \leq FIELD1 \leq 120$.</p> <p>For the importing parameter <code>iv_low</code> and <code>iv_high</code>, enter the values that provide the lower limit and upper limit of the range (for example numeric values like 72 and 120).</p> <pre>ro_condition = lo_cfac->element('FIELD1')- >is_not_between(iv_low = 72 iv_high = 120).</pre>

4. You can concatenate the conditions built by the previous steps with the methods of the logical operators AND, OR, NEGATED, and EXISTS.

For further information, see [Interface IF_SADL_COND_PROVIDER_GENERIC \[page 228\]](#).

Logical Operators	Description
<code>and</code>	<p>Use this method to concatenate two conditions if you want the concatenated condition to be true if both expressions (the preceding and the succeeding expression) are true.</p> <p>For the importing parameter enter a condition as built in step 1 to step 3.</p> <pre>ro_condition = lo_condition1->and(lo_condition2).</pre>
<code>or</code>	<p>Use this method to concatenate two conditions if you want the concatenated condition to be true if either of the expressions (the preceding or the succeeding expression) is true.</p> <p>For the importing parameter enter a condition as built in step 1 to step 3.</p> <pre>ro_condition = lo_condition1->or(lo_condition2).</pre>
<code>negated</code>	<p>Use this method to negate the preceding expression.</p> <p>This method does not have an importing parameter.</p> <pre>ro_condition = lo_condition1->negated().</pre> <p>This condition expresses the same content as a condition with the opposite relational operator.</p>
<code>exists</code>	<p>Use this method if the preceding condition refers to an associated CDS entity with cardinality "ONE TO MANY". The condition is true if there are entries in the source entity that have associated entries that satisfy the condition.</p> <p>For the importing parameter (<code>iv_name</code>) enter an alias for the target entity that you have chosen in the preceding condition.</p>

Logical Operators	Description
	<p>For the importing parameter (<code>iv_path</code>) enter the association path that is declared in the CDS entity definition.</p> <pre>ro_condition = lo_cfac->element('alias.element')- >equals(72)->exists(iv_name = 'alias' iv_path = '_assoc').</pre>

Results

You can continuously combine conditions with relational and logical operators:

```
ro_condition = (lo_cfac->element( 'FIELD1' )->equals( 'xyz' )
->and(lo_cfac->element( 'FIELD2' )->less_than( 65 ) ) )->or( lo_cfac ... ).
```

→ Tip

Improve the readability of your condition by predefining the first operand before you use it in the condition:

```
DATA(field1) = lo_cfac->element( 'FIELD1' ).
```

The condition then looks like this:

```
ro_condition = field1->less_than( 100 ).
```

This condition is true for all field values of *FIELD1* that are less than 100.

8.2.2.1.1 API for Conditions

The following interfaces define methods that are used to construct complex conditions with the simple condition factory.

- [Interface IF_SADL_SIMPLE_COND_FACTORY \[page 222\]](#)
- [Interface IF_SADL_SIMPLE_COND_ELEMENT \[page 223\]](#)
- [Interface IF_SADL_COND_PROVIDER_GENERIC \[page 228\]](#)

8.2.2.1.1.1 Interface `IF_SADL_SIMPLE_COND_FACTORY`

The methods of this interface are used to construct complex conditions with the simple condition factory. It creates objects that are set into relation with input values by relational operators.

The method `CREATE_SIMPLE_COND_FACTORY` that is called in the class `CL_SADL_COND_PROV_FACTORY_PUB` returns a value with reference to the interface `IF_SADL_SIMPLE_COND_FACTORY`. This interface defines the following methods to start a complex condition.

Method `element`

This method provides the start of a complex condition. It defines the first operand of a relation.

Signature

```
METHODS element IMPORTING iv_name          TYPE sadl_entity_element
           RETURNING VALUE(ro_element)  TYPE REF TO
           if_sadl_simple_cond_element.
```

Parameters

<code>IV_NAME</code>	Name of a non-virtual element that is set into relation with a value by the methods of relational operators.
<code>RO_ELEMENT</code>	Operand of the condition represented by a CDS element. The methods of relational operators can be executed on this object.

Methods `true` and `false`

These methods set the simple condition factory to true or false.

Signature

```
METHODS true RETURNING VALUE(ro_condition)  TYPE REF TO
           if_sadl_cond_provider_generic.
```

Parameters

<code>RO_CONDITION</code>	Complete condition which is always either true or false. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface <code>IF_SADL_COND_PROVIDER_GENERIC</code> .
---------------------------	---

8.2.2.1.1.2 Interface `IF_SADL_SIMPLE_COND_ELEMENT`

The methods of this interface are used to construct complex conditions with the simple condition factory. They provide relational operators to set operands into relation.

The method `element` of the interface `IF_SADL_SIMPLE_COND_FACTORY` returns the first operand of a relation. The following methods operate on this returning value.

All of the following methods return an object pointing to `IF_SADL_COND_PROVIDER_GENERIC`. The methods of this interface can thus be used consecutively.

Methods equals and not_equals

These methods set two operands into relation with each other. The condition is considered to be true if the value of the first operand is equal to the value of the second operand.

Signature

```
METHODS: equals      IMPORTING iv_value          TYPE simple
          RETURNING VALUE(ro_conditon)  TYPE REF TO
if_sadl_cond_provider_generic.
METHODS: not_equals  IMPORTING iv_value          TYPE simple
          RETURNING VALUE(ro_conditon)  TYPE REF TO
if_sadl_cond_provider_generic.
```

Parameters

IV_VALUE	Value of the second operand that is set into relation with the first operand by EQUALS or NOT_EQUALS
RO_CONDITION	Complete condition. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface IF_SADL_COND_PROVIDER_GENERIC.

Methods less_than and greater_than

These methods set two operands into relation with each other. The condition is considered to be true if the value of the first operand is less than or greater than the value of the second operand.

Signature

```
METHODS: less_than   IMPORTING iv_value          TYPE simple
          RETURNING VALUE(ro_conditon)  TYPE REF TO
if_sadl_cond_provider_generic.
METHODS: greater_than IMPORTING iv_value          TYPE simple
          RETURNING VALUE(ro_conditon)  TYPE REF TO
if_sadl_cond_provider_generic.
```

Parameters

IV_VALUE	Value of the second operand that is set into relation with the first operand by LESS_THAN or GREATER_THAN.
RO_CONDITION	Complete condition. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface IF_SADL_COND_PROVIDER_GENERIC.

• Example

The following code example illustrates how the method greater_than is implemented. The example is taken from a scenario that processes sales orders and sales order items. The condition is implemented in a code exit class for a virtual element.

Assume we have a simple CDS consumption view that processes sales order items. The select list includes a virtual element RECEIVESDISCOUNT that evaluates to true or false, depending on the gross amount of

the item. The virtual element evaluates to `true` if the gross amount is greater than 1000. This virtual element can be filtered by the condition `RECEIVESDISCOUNT eq true` or `RECEIVESDISCOUNT eq false`. This filter condition must be translated into a condition for an element of the original persistent data model. As you can see in the following code, the element `CONVERTEDGROSSAMOUNT` is used for this in the method `IF_SADL_EXIT_FILTER_TRANSFORM~MAP_ATOM`. The method of the relational operator in this case is `greater_than`.

```

METHOD if_sadl_exit_filter_transform~map_atom.
  IF iv_element <> 'RECEIVESDISCOUNT'.
    RAISE EXCEPTION TYPE zcx_filter_exit EXPORTING textid =
      zcx_filter_exit=>element_not_expected.
  ENDIF.
  CASE iv_operator.
    WHEN if_sadl_exit_filter_transform~co_operator>equals.
      DATA(lo_cfac) =
        cl_sadl_cond_prov_factory_pub=>create_simple_cond_factory( ).
      DATA(amount) = lo_cfac->element( 'CONVERTEDGROSSAMOUNT' ) .
      IF iv_value = abap_true.
        ro_condition = amount->greater_than( 1000 ) .
      ELSEIF iv_value = abap_false.
        ro_condition = amount->less_than_or_equals( 1000 ) .
      ELSE.
        RAISE EXCEPTION TYPE zcx_filter_exit.
      ENDIF.
    WHEN OTHERS.
      RAISE EXCEPTION TYPE zcx_filter_exit.
  ENDCASE.
ENDMETHOD.
```

Methods `less_than_or_equals` and `greater_than_or_equals`

These methods set two operands into relation with each other. The condition is considered to be true if the value of the first operand is less than or equal to or greater than or equal to the value of the second operand.

Signature

```

METHODS: less_than_or_equals      IMPORTING iv_value           TYPE simple
          RETURNING VALUE(ro_condition) TYPE REF TO
if_sadl_cond_provider_generic.
METHODS: greater_than_or_equals   IMPORTING iv_value           TYPE simple
          RETURNING VALUE(ro_condition) TYPE REF TO
if_sadl_cond_provider_generic.
```

Parameters

<code>IV_VALUE</code>	Value of the second operand that is set into relation to the first operand by <code>LESS_THAN_OR_EQUALS</code> or <code>GREATER_THAN_OR_EQUALS</code> .
<code>RO_CONDITION</code>	Complete condition. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface <code>IF_SADL_COND_PROVIDER_GENERIC</code> .

Method `is_null` and `is_not_null`

These methods provide a condition that is true if the first operand is (not) null.

Signature

```
METHODS: is_null      RETURNING VALUE(ro_conditon) TYPE REF TO  
if_sadl_cond_provider_generic.  
METHODS: is_not_null RETURNING VALUE(ro_conditon) TYPE REF TO  
if_sadl_cond_provider_generic.
```

Parameters

RO_CONDITION	Complete condition. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface <code>IF_SADL_COND_PROVIDER_GENERIC</code> .
--------------	--

Method `covers_pattern` and `not_covers_pattern`

These methods provide a condition that is true if the first operand covers the pattern or false if the first operand does not cover the pattern of the second operand respectively.

Signature

```
METHODS: covers_pattern      IMPORTING iv_value           TYPE csequence  
                    RETURNING VALUE(ro_conditon) TYPE REF TO  
if_sadl_cond_provider_generic.  
METHODS: not_covers_pattern IMPORTING iv_value           TYPE csequence  
                    RETURNING VALUE(ro_conditon) TYPE REF TO  
if_sadl_cond_provider_generic.
```

Example

The following code example illustrates how the method `COVERS_PATTERN` is implemented. The example is taken from a scenario that processes sales orders and sales order items. The condition is implemented in a code exit class for a virtual element.

Assume we have a simple CDS consumption view that processes sales order items. The select list includes a virtual element `LIMITEDITION` that can take the values `Christmas`, `Spring`, `Holiday`, and `Unlimited`, depending on the product ID of the item. The following code snippet shows the implementation of the method `IF_SADL_EXIT_CALC_ELEMENT_READ~CALCULATE`, which implements this calculation of the virtual element:

```
METHOD if_sadl_exit_calc_element_read~calculate.  
  CHECK NOT it_original_data IS INITIAL.  
  DATA lt_calculated_data TYPE STANDARD TABLE OF zdemo_c_soi_vf_condition  
  WITH DEFAULT KEY.  
  MOVE-CORRESPONDING it_original_data TO lt_calculated_data.  
  LOOP AT lt_calculated_data ASSIGNING FIELD-SYMBOL(<ls_calculated_data>).  
    IF <ls_calculated_data>-product CP 'HT-111*'.  
      <ls_calculated_data>-limitedition = 'Christmas'.  
    ELSEIF <ls_calculated_data>-product CP 'HT-107*'.  
      <ls_calculated_data>-limitedition = 'Spring'.  
    ELSEIF <ls_calculated_data>-product CP 'HT-12*' OR <ls_calculated_data>-product CP 'HT-15*'.  
    ENDIF.
```

```

        <ls_calculated_data>-limitededition = 'Holiday'.
ELSE.
    <ls_calculated_data>-limitededition = 'Unlimited'.
ENDIF.
ENDLOOP.
MOVE-CORRESPONDING lt_calculated_data TO ct_calculated_data.
ENDMETHOD.

```

The virtual element can be filtered. The filter condition must be translated into a condition for an element of the original persistent data model. As can be seen in the following code, the element `PRODUCT` is used for this in the method `IF_SADL_EXIT_FILTER_TRANSFORM~MAP_ATOM`. The method of the relational operator in this case is `COVERS_PATTERN`.

```

METHOD if_sadl_exit_filter_transform~map_atom.
    IF iv_element <> 'LIMITED_EDITION'.
        RAISE EXCEPTION TYPE zcx_filter_exit EXPORTING textid =
zcx_filter_exit=>element_not_expected.
    ENDIF.
    CASE iv_operator.
        WHEN if_sadl_exit_filter_transform~co_operator>equals.
            DATA(lo_cfac) =
cl_sadl_cond_prov_factory_pub=>create_simple_cond_factory( ).
            DATA(product) = lo_cfac->element( 'PRODUCT' ) .
        CASE iv_value.
            WHEN 'Christmas'.
                ro_condition = product->covers_pattern( 'HT-111*' ) .
            WHEN 'Spring'.
                ro_condition = product->covers_pattern( 'HT-107*' ) .
            WHEN 'Holiday'.
                ro_condition = product->covers_pattern( 'HT-12*' )->or(
                    product->covers_pattern( 'HT-15*' ) ) .
            WHEN 'Unlimited'.
                ro_condition = product->not_covers_pattern( 'HT-111*' )->and(
                    product->not_covers_pattern( 'HT-107*' ) )->and(
                    product->not_covers_pattern( 'HT-12*' ) )->and(
                    product->not_covers_pattern( 'HT-15*' ) ) .
            WHEN OTHERS.
                RAISE EXCEPTION TYPE zcx_filter_exit.
        ENDCASE.
    WHEN OTHERS.
        RAISE EXCEPTION TYPE zcx_filter_exit EXPORTING textid =
zcx_filter_exit=>element_not_expected.
    ENDCASE.
ENDMETHOD.

```

Parameters

<code>IV_VALUE</code>	Value of the second operand that is set into relation with the first operand by <code>COVERS_PATTERN</code> or <code>NOT_COVERS_PATTERN</code> .
<code>RO_CONDITION</code>	Complete condition. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface <code>IF_SADL_COND_PROVIDER_GENERIC</code> .

Method `between` and `not_between`

These methods provide a condition that is true if the value of the first operand is (not) between the values of the importing parameters `IV_LOW` and `IV_HIGH`.

Signature

```
METHODS: between      IMPORTING iv_low           TYPE simple
          iv_high          TYPE simple
          RETURNING VALUE(ro_conditon) TYPE REF TO
if_sadl_cond_provider_generic.
METHODS: not_between  IMPORTING iv_low           TYPE simple
          iv_high          TYPE simple
          RETURNING VALUE(ro_conditon) TYPE REF TO
if_sadl_cond_provider_generic.
```

Parameters

IV_LOW	Value of the lower boundary of the interval that is set into relation with the first operand by BETWEEN or NOT_BETWEEN.
IV_HIGH	Value of the upper boundary of the interval that is set into relation with the first operand by BETWEEN or NOT_BETWEEN.
RO_CONDITION	Complete condition. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface IF_SADL_COND_PROVIDER_GENERIC.

8.2.2.1.1.3 Interface IF_SADL_COND_PROVIDER_GENERIC

Interface that provides concatenation methods to build complex conditions. It also provides methods that connect conditions with logical operators.

This interface defines the following methods to concatenate conditions. The return value of these methods point to the exact same interface that enables unlimited concatenation of conditions. The methods are executed on objects pointing to IF_SADL_COND_PROVIDER_GENERIC.

Methods and or

These methods concatenate two conditions constructed by the simple condition factory.

AND operates like the logical operator "and". The concatenated condition is true if both conditions are true.

OR operates like the logical operator "or". The concatenated condition is true if either of the conditions is true or if both conditions are true.

Signature

```
METHODS and
  IMPORTING io_condition      TYPE REF TO if_sadl_cond_provider_generic
  RETURNING VALUE(ro_condition) TYPE REF TO if_sadl_cond_provider_generic.
METHODS or
  IMPORTING io_condition      TYPE REF TO if_sadl_cond_provider_generic
  RETURNING VALUE(ro_condition) TYPE REF TO if_sadl_cond_provider_generic.
```

Parameters

IO_CONDITION	Condition that is constructed with the simple condition factory.
RO_CONDITION	Complete condition. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface <code>IF_SADL_CON_PROVIDER_GENERIC</code> .

Method negated

The method `NEGATED` reverses the truth content of the preceding condition.

It operates like the logical operator "not". The condition is true if the preceding expression is false and vice versa.

Signature

```
METHODS negated
      RETURNING VALUE(ro_condition) TYPE REF TO if_sadl_cond_provider_generic.
```

Parameters

RO_CONDITION	Complete condition. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface <code>IF_SADL_CON_PROVIDER_GENERIC</code> .
--------------	---

Method exists

The method `EXISTS` is used if filter conditions are substituted by conditions on non-virtual elements of associated views with cardinality 1... n. In this case, it ensures that only those entries of the source entity are retrieved for which entries of the filtered target entity exist.

Signature

```
METHODS negated
      RETURNING VALUE(ro_condition) TYPE REF TO if_sadl_cond_provider_generic.
```

Parameters

IV_NAME	Alias for the associated entity, in which the preceding condition is applied. This name needs to match the navigation alias of the importing parameter <code>iv_element</code> , which is used for the method <code>ELEMENT</code> of the preceding condition.
IV_PATH	Name of the association in which the preceding condition is applied. The name of the association is defined in the data definition of the source CDS entity. The path can either be an association name or a concatenation of a predecessor path and an association name separated by a single dot.
RO_CONDITION	Complete condition. It can be used for the substitution of filters for virtual elements or further concatenated by methods of the interface <code>IF_SADL_CON_PROVIDER_GENERIC</code> .

• Example

The following example illustrates how the method `EXISTS` is implemented. The example is taken from a scenario that processes sales orders and sales order items. The condition is implemented in a code exit class for a virtual element.

Assume we have a simple CDS consumption view `ZDEMO_C_SO_VF_CONDITION` that processes sales orders. The select list includes a virtual element `HASDISCOUNT` that can take the values `true` and `false`. These values depend on the gross amount of the sales order items in this sales order. The items, however, are processed in an associated view, `ZDEMO_C_VF_CONDITION` with the alias `_Items`. This is why we need a `SELECT` within the method `IF_SADL_EXIT_CALC_ELEMENT_READ~CALCULATE` to be able to use the element from a foreign view for the calculation of the virtual element. The following code snippet shows the implementation of the calculation for the virtual element `HASDISCOUNT`, which is defined in a view that processes sales orders. The virtual element takes the value `true` if the gross amount of the sales order items is greater than 1000.

```
METHOD if_sadl_exit_calc_element_read~calculate.  
  CHECK NOT it_original_data IS INITIAL.  
  DATA lt_calculated_data TYPE STANDARD TABLE OF zdemo_c_so_vf_condition  
  WITH DEFAULT KEY.  
    MOVE-CORRESPONDING it_original_data TO lt_calculated_data.  
    CHECK NOT lt_calculated_data IS INITIAL .  
    SELECT salesorder FROM zdemo_c_soi_vf_condition INTO TABLE  
    @DATA(lt_discount_items)  
    FOR ALL ENTRIES IN @lt_calculated_data  
      WHERE salesorder = @lt_calculated_data-salesorder AND  
      convertedgrossamount > 1000.  
    LOOP AT lt_calculated_data ASSIGNING FIELD-SYMBOL(<ls_calculated_data>).  
      IF line_exists( lt_discount_items[ salesorder = <ls_calculated_data>-  
      salesorder ] ).  
        <ls_calculated_data>-hasdiscount = abap_true.  
      ELSE.  
        <ls_calculated_data>-hasdiscount = abap_false.  
      ENDIF.  
      MOVE-CORRESPONDING lt_calculated_data TO ct_calculated_data.  
    ENDLOOP.  
ENDMETHOD.
```

The virtual element can be filtered by the condition `HASDISCOUNT eq true` or `HASDISCOUNT eq false`. This filter condition must be translated into a condition for an element of the original, persistent data model. This persistent element (`CONVERTEDGROSSAMOUNT`) is defined in the associated view of sales order items. For this reason, you need the method `EXISTS` to implement the condition translation with the method `IF_SADL_EXIT_FILTER_TRANSFORM~MAP_ATOM`.

```
METHOD if_sadl_exit_filter_transform~map_atom.  
  IF iv_element <> 'HASDISCOUNT'.  
    RAISE EXCEPTION TYPE zcx_filter_exit EXPORTING textid =  
    zcx_filter_exit=>element_not_expected.  
  ENDIF.  
  CASE iv_operator.  
    WHEN if_sadl_exit_filter_transform~co_operator-equals.  
      DATA(lo_cfac) =  
      cl_sadl_cond_prov_pub->create_simple_cond_factory( ).  
      DATA(amount) = lo_cfac->element( 'ITEM.CONVERTEDGROSSAMOUNT' ).  
      DATA(lo_condition) = amount->greater_than( 1000 )->exists( iv_name =  
      'ITEM' iv_path = '_Items' ).  
      IF iv_value = abap_true.  
        ro_condition = lo_condition.  
      ELSEIF iv_value = abap_false.  
        ro_condition = lo_condition->negated( ).
```

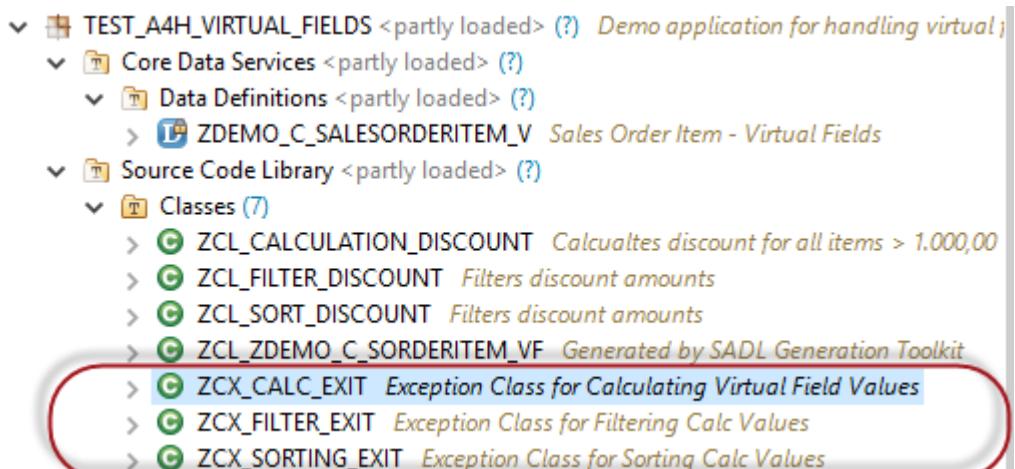
```

ENDIF.
WHEN OTHERS.
  RAISE EXCEPTION TYPE zcx_filter_exit.

ENDCASE.
ENDMETHOD.
```

8.2.3 Creating Application-Specific Exception Classes

To handle application-specific exceptions in the context of virtual elements, you must create exception classes that inherit from the generic SADL exit class CX_SADL_EXIT.



Exception classes for specific use cases

To explicitly raise a predefined exception, add, for example, the constant element_not_expected to the exception class – as shown in the listing below.

Listing: Added constant element_not_expected

```

CLASS zcx_filter_exit DEFINITION
  PUBLIC
    INHERITING FROM cx_sadl_exit
    FINAL
  CREATE PUBLIC .
  PUBLIC SECTION.
    CONSTANTS:
      BEGIN OF element_not_expected,
        msgid TYPE symsgid VALUE '<MY_MESSAGE_ID>',
        msgno TYPE symsgno VALUE '<MY_MESSAGE_NR>',
        attr1 TYPE scx_attrname VALUE '<ATTR1>',
        attr2 TYPE scx_attrname VALUE '<ATTR2>',
        attr3 TYPE scx_attrname VALUE '',
        attr4 TYPE scx_attrname VALUE '',
      END OF element_not_expected .
    INTERFACES if_t100_dyn_msg .
    INTERFACES if_t100_message .
  METHODS constructor
    IMPORTING
      !textid LIKE if_t100_message=>t100key OPTIONAL
      !previous LIKE previous OPTIONAL .
  PROTECTED SECTION.
```

```

PRIVATE SECTION.
ENDCLASS.
CLASS zcx_filter_exit IMPLEMENTATION.
METHOD constructor ##ADT_SUPPRESS_GENERATION.
  CALL METHOD super->constructor
    EXPORTING
      previous = previous.
  CLEAR me->textid.
  IF textid IS INITIAL.
    if_t100_message~t100key = if_t100_message=>default_textid.
  ELSE.
    if_t100_message~t100key = textid.
  ENDIF.
ENDMETHOD.
ENDCLASS.

```

8.2.4 Understanding the ABAP Code Exit API for Virtual Elements

The code exit class for virtual elements is a global ABAP class that implements the corresponding interface(s) - according to the use case:

- Interface [IF_SADL_EXIT_CALC_ELEMENT_READ \[page 232\]](#)
- Interface [IF_SADL_EXIT_FILTER_TRANSFORM \[page 234\]](#)
- Interface [IF_SADL_EXIT_SORT_TRANSFORM \[page 235\]](#)

8.2.4.1 Interface IF_SADL_EXIT_CALC_ELEMENT_READ

Interface that must be implemented by calculation classes annotated at the level of the CDS view element with the `@ObjectModel.virtualElement` annotation.

This interface defines the following methods for implementing the field calculation:

Method GET_CALCULATION_INFO

Provides a list of all elements that are required for calculating the values of the virtual elements

This method is called by the SADL framework prior to the retrieval of data from the database to ensure that all fields necessary for calculation are filled with data

Signature

<i>importing</i>	<i>IT_REQUESTED_CALC_ELEMENTS</i>	<i>type</i>	<i>IF_SADL_EXIT_CALC_ELEMENT_READ=>TT_ELEMENTS</i>
------------------	-----------------------------------	-------------	---

	<i>IV_ENTITY</i>	type	<i>STRING</i>
<i>exporting</i>	<i>ET_REQUESTED_ORIG_ELEMENTS</i>	type	<i>IF_SADL_EXIT_CALC_ELEMENT_READ=>TT_ELEMENTS</i>
<i>raising</i>	<i>CX_SADL_EXIT</i>		

Parameters

Parameter	Description
<i>IT_REQUESTED_CALC_ELEMENTS</i>	List of virtual elements that are requested by the client and are intended for field calculation
<i>IV_ENTITY</i>	Name of the CDS entity, such as a CDS view, that defines the relevant virtual elements
<i>ET_REQUESTED_ORIG_ELEMENTS</i>	List of original persistent fields (in capital letters) that are used to calculate the virtual elements
<i>CX_SADL_EXIT</i>	Abstract exception class that can be used to return exceptions and messages related to the processing of virtual element calculation

Method CALCULATE

Executes the field calculation

This method is called by the SADL framework after data is retrieved from the database. The elements needed for the calculation of the virtual elements are already filled in the data table passed to this method. The method returns a table that contains the values of the requested virtual elements.

i Note

Only elements of the same CDS entity (view) must be used for the calculation.

i Note

Calculated fields cannot be used together with the grouping or the aggregation function.

Signature

<i>importing</i>	<i>IT_ORIGINAL_DATA</i>	type	<i>STANDARD TABLE</i>
	<i>IT_REQUESTED_CALC_ELEMENTS</i>	type	<i>IF_SADL_EXIT_CALC_ELEMENT_READ=>TT_ELEMENTS</i>
<i>exporting</i>	<i>CT_CALCULATED_DATA</i>	type	<i>STANDARD TABLE</i>
<i>raising</i>	<i>CX_SADL_EXIT</i>		

Parameters

Parameter	Description
IT_ORIGINAL_DATA	Table of original data that is filled with the original persistent field values that are used to calculate the virtual elements.
IT_REQUESTED_CALC_ELEMENTS	List of virtual elements that are requested by the client and are intended for field calculation
CT_CALCULATED_DATA	Table of calculated fields which correspond to the original data by index.
CX_SADL_EXIT	Abstract exception class that can be used to return exceptions and messages related to the processing of virtual element calculation.

8.2.4.2 Interface IF_SADL_EXIT_FILTER_TRANSFORM

Interface that must be implemented by filter transformation classes annotated at the level of the CDS view element with the @ObjectModel.filter.transformedBy annotation

This interface defines the following method for implementing the filter transformation:

Method MAP_ATOM

Transforms filter conditions specified for the annotated view element to filter criteria of other view elements

This method is called by the SADL framework before data is retrieved from the database. Note that this method is primarily intended for replacing filter conditions on virtual elements. However, it can also be used to achieve better performance when filtering elements that have been calculated by means of CDS.

Signature

importing	IV_ENTITY	type	STRING
	IV_ELEMENT	type	SADL_ENTITY_ELEMENT
	IV_OPERATOR	type	STRING
	IV_VALUE	type	STRING
returning value	RO_CONDITION	type ref to	IF_SADL_COND_PROVIDER_GENERIC
raising	CX_SADL_EXIT_FILTER_NOT_SUPP		
	CX_SADL_EXIT		

Parameters

Parameter	Description
IV_ENTITY	Name of the CDS entity, such as the CDS view that defines the virtual elements in question
IV_ELEMENT	Name of the filter element, for example 'FIELD1' in the filter condition: FIELD1 = 77
IV_OPERATOR	Condition operator: EQ (equals), LT (less than), LE (less or equal), GE (greater or equal), GT (greater than), NL (is null), or CP (covers pattern)
IV_VALUE	Condition value, for example '77' in the condition: FIELD1 = 77
RO_CONDITION	Filter condition that replaces the condition of the annotated element
CX_SADL_EXIT_FILTER_NOT_SUPPORTED	Exception class from the SADL framework that throws an exception if the filter operation is not supported
CX_SADL_EXIT	Abstract exception class that can be used for returning exceptions and messages related to the processing of virtual elements

8.2.4.3 Interface IF_SADL_EXIT_SORT_TRANSFORM

Interface that must be implemented by sort transformation classes annotated at the level of the CDS view element with the `@ObjectModel.sort.transformedBy` annotation

This interface defines the following method for implementing the sort transformation:

Method MAP_ELEMENT

Transforms the sort criteria specified for the annotated view element to sort criteria of other view elements

This method is called by the SADL framework before data is retrieved from the database.

i Note

You can also use multiple elements to replace the sort criteria. In addition, you can change the sort order if necessary.

i Note

When using this sort transformation along with the grouping or aggregation function, you must ensure that elements to be replaced are part of the requested elements (requested by a client). In addition, you must annotate the element to be replaced with `@Consumption.groupWith: 'replacement_element'`.

Signature

importing	IV_ENTITY	type	STRING
	IV_ELEMENT	type	SADL_ENTITY_ELEMENT
exporting	ET_SORT_ELEMENTS	type	IF_SADL_EXIT_SORT_TRANSFORM=>TT_SORT_ELEMENTS
raising	CX_SADL_EXIT		

Parameters

Parameter	Description
IV_ENTITY	Name of the CDS entity, such as the CDS view that defines the virtual elements in question
IV_ELEMENT	Name of the sort element
ET_SORT_ELEMENTS	Table with replacement elements that will be used for sorting instead of the annotated element For each element, it specifies whether the sort order must be kept or reversed.
CX_SADL_EXIT	Abstract exception class that can be used to return exceptions and messages related to the processing of virtual elements

8.3 Consuming Temporal Data

The following sections describe the concept of temporal data and provide information about how to annotate the data model to apply temporal filtering in OData requests.

We speak of temporal data whenever data properties depend on time, meaning that certain properties are only valid during a specific validity period. If temporal data is stored in the database with table fields that define the beginning and the end of the valid time frame, temporal data can be modeled in CDS entities and retrieved in OData requests.

More on this: [Temporal Data \[page 237\]](#)

i Note

Temporal data is not supported in transactional scenarios.

To be able to retrieve data that is filtered by a certain date, the CDS views of the data model must be equipped with annotations that mark the temporal date fields as filter references. Additionally, you can add an entity annotation which triggers a default filter with the system date if no other filter date is applied by the user.

More on this: [Adding Annotations for Full Temporal Exposure \[page 239\]](#)

To pass the value of the custom query option to a view parameter, you have to add an annotation to the parameter in the data model.

More on this: [Adding Annotations when Using a Temporal Parameter \[page 243\]](#)

As soon as the data model is equipped with the relevant annotations, the temporal filtering can be applied by using a custom query option, which is added to the URI. The value of the custom query option is used to find the matching record for the query.

More on this: [Applying Temporal Filtering in OData Requests \[page 245\]](#)

8.3.1 Temporal Data

Temporal data is data that is time-dependent. The data properties are only valid during a specific validity period. If temporal data is stored in the database with table fields that define the beginning and the end of the valid time frame, temporal data can be modeled in CDS entities and retrieved in OData requests.

There are various scenarios in which data varies over time. Imagine a person moves and gets a new address or think of an employee's income, which increases from time to time. In this case, the properties of a data set (for example an address or salary amount) have different values for each time frame in which they are valid. When using temporal data, not only the current state of information is stored, but every version of the information together with its validity period.

The data becomes temporal as soon as a timestamp is stored as part of the data set. Thus, the database table requires fields (for example `validfrom` and `validto`) that specify the time frame in which the respective properties are valid. To avoid losing old information, instead of overwriting a row with the current information, a new row is attached to the data set. The valid times are stored in separate columns in the database to identify the validity period of the relevant data. Each property change triggers the attachment of a new data set to keep the information up-to-date without losing expired data. For every time instance in the past and present you then find a matching set of data.

You can now retrieve and filter temporal data by validity dates. For this filtering, you can either use the system date as a data reference to retrieve the current state of the data or a time travel query to retrieve data of previously expired validity periods.

Example

A company stores information about employees. Some properties, however, are not static over time, but might change during the work cycle of the employee, for example the job title or the department in which the employee works.

The following table demonstrates the employee information that is dependent on time. There are two columns identifying the beginning and end of the respective validity period.

Exemplary Employee Data

ID	VALIDFROM	VALIDTO	NAME	JOBTITLE	DEPARTMENT
E001	19900101	19990630	Sue Bay	Administrative Assistant	D001

ID	VALIDFROM	VALIDTO	NAME	JOBTITLE	DEPARTMENT
E001	19990701	20091231	Sue Bay	Administrative Manager	D001
E001	20100101	20111130	Sue Bay	Administrative Director	D002
E001	20111201	99991231	Sue Bay	Administrative Director	D003
E002	20010301	20010630	Paul Burke	Area Sales Manager	D004
E002	20010701	20160930	Paul Burke	Brand Ambassador	D004
E004	19810215	20051130	Lindsey Brown	Key Account Manager	D004
E004	20051201	99991231	Lindsey Palmer	Key Account Manager	D004
E005	20140301	99991231	Al Wallace	Contract Recruiter	D005

Sue Bay works in Department D001 as Administrative Assistant during a specific time period: January 1, 1990 until June 30, 1999. Between July 1, 1999 and December 31, 2009 she works as Administrative Manager, becoming Administrative Director on January 1, 2010 in a different department and changes to Department D003 on December 1, 2011.

The example demonstrates that there can be more than one record related to one employee. However, by restricting the data to one specific day, there is only one record which matches exactly one employee at one specific point in time.

Unique Key Fields for Temporal Data

Database tables with temporal data cannot have a key which is only assigned to fields of one column. The value of a single key can be used for several different records as every property change triggers an attachment of a new record with the matching validity period. In this case, a single key field does not guarantee uniqueness. Uniqueness is only achieved by a compound key, which includes one of the temporal date fields.

In the example above, the *ID* field is not an applicable unique key field for the temporal data set. Only by combining the *ID* field with at least one of the temporal date fields is every record clearly identifiable. This makes the key unique.

The following figure demonstrates how a compound key makes the record unique when dealing with temporal data.

Table with Simple Key

KEY	ID	VALIDFROM	VALIDTO	NAME	JOBTITLE	DEP
	E001	19900101	19990630	Sue Bay	Administrative Assistant	D001
	E001	19990701	20091231	Sue Bay	Administrative Manager	D001
	E001	20100101	20111130	Sue Bay	Administrative Director	D002
	E001	20111201	99991231	Sue Bay	Administrative Director	D003
	E002	20010301	20010630	Paul Burke	Area Sales Manager	D004
	E002	20010701	20160930	Paul Burke	Brand Ambassador	D004
	E004	19810215	20051130	Lindsey Brown	Key Account Manager	D004
	E004	20051201	99991231	Lindsey Palmer	Key Account Manager	D004
	E005	20140301	99991231	Al Wallace	Contract Recruiter	D005

Table with Compound Key

KEY	ID	VALIDFROM	VALIDTO	NAME	JOBTITLE	DEP
	E001	19900101	19990630	Sue Bay	Administrative Assistant	D001
	E001	19990701	20091231	Sue Bay	Administrative Manager	D001
	E001	20100101	20111130	Sue Bay	Administrative Director	D002
	E001	20111201	99991231	Sue Bay	Administrative Director	D003
	E002	20010301	20010630	Paul Burke	Area Sales Manager	D004
	E002	20010701	20160930	Paul Burke	Brand Ambassador	D004
	E004	19810215	20051130	Lindsey Brown	Key Account Manager	D004
	E004	20051201	99991231	Lindsey Palmer	Key Account Manager	D004
	E005	20140301	99991231	Al Wallace	Contract Recruiter	D005

Uniqueness of Record by Compound Key

Activities Relevant to the Developer

To be able to retrieve data that is filtered by a certain date, the CDS views of the data model must be equipped with annotations that mark the temporal date fields as filter references. Additionally, you can add an entity annotation which triggers a default filter with the system date if no other filter date is applied by the user.

More on this: [Adding Annotations for Full Temporal Exposure \[page 239\]](#)

To pass the value of the custom query option to a view parameter, you have to add an annotation to the parameter in the data model.

More on this: [Adding Annotations when Using a Temporal Parameter \[page 243\]](#)

As soon as the data model is equipped with the relevant annotations, the temporal filtering can be applied by using a custom query option, which is added to the URL. The value of the custom query option is used to find the matching record for the query.

More on this: [Applying Temporal Filtering in OData Requests \[page 245\]](#)

8.3.1.1 Adding Annotations for Full Temporal Exposure

To enable temporal filtering by a custom query option, several annotations in the data model are required.

Prerequisites

- The retrieval of filtered temporal data is not supported in transactional scenarios.
- The CDS elements that express the boundaries of the validity period must be of type DATS. No other format or greater granularity is supported.
- The data set must not contain overlapping time frames. Overlapping occurs when calendar days coincide in two or more different time frames. Every real calendar date must be clearly identifiable as only one valid time frame in the database (or no valid time frame at all).

i Note

If there are discontinuances between periods, for example breaks between two subsequent periods, the output does not contain any data. The OData request then raises the regular exception “Not found”, as there is no data in the data set matching the request. From a semantic point of view, the entity is considered as nonexistent at the given point in time.

- If a time period has not expired yet the end of the validity period must be set to a sufficiently large value, preferably the largest possible value, for example December 31, 9999.

Element Annotations

Context

The elements specifying the beginning of the validity period and the end of the validity period must be marked as the reference for the temporal filter. This marking is done by two annotations that need to be added to the elements.

Procedure

- Add `@semantics.businessDate.from` to the element that defines the beginning of the validity period.
- Add `@semantics.businessDate.to` to the element that defines the end of the validity period

```
define view <CdsConsumptionView>
  as select from <data_source>
{  key   <view_element>
  @Semantics.businessDate.from
  key   validfrom,
  @Semantics.businessDate.to
  validto,
  ... }
```

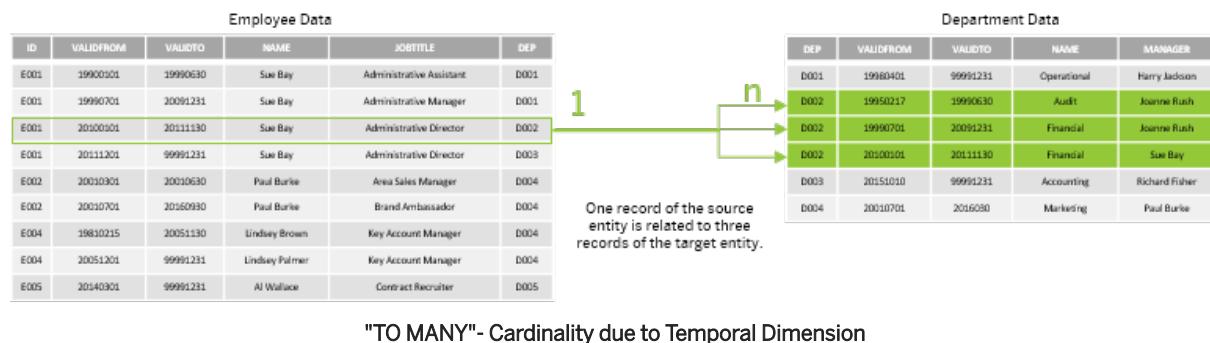
Results

Whenever a temporal custom query option `at=YYYYMMDD` is used in an OData request, the rows in records are filtered by the delivered value and only records with a time frame matching the temporal query option are exposed.

Annotation for Associations to Trigger the Reduction of Association Cardinalities

Context

The cardinality of the target data source (which contains temporal data) is "TO MANY", as every record of the source entity is related to one or more records of the dependent entity due to the additional temporal dimension.



To reduce the cardinality of the target data source, when a temporal filter is used, the foreign key association must use an @consumption annotation.

Procedure

- Add @consumption.filter.businessDate.at:true to the foreign key association.

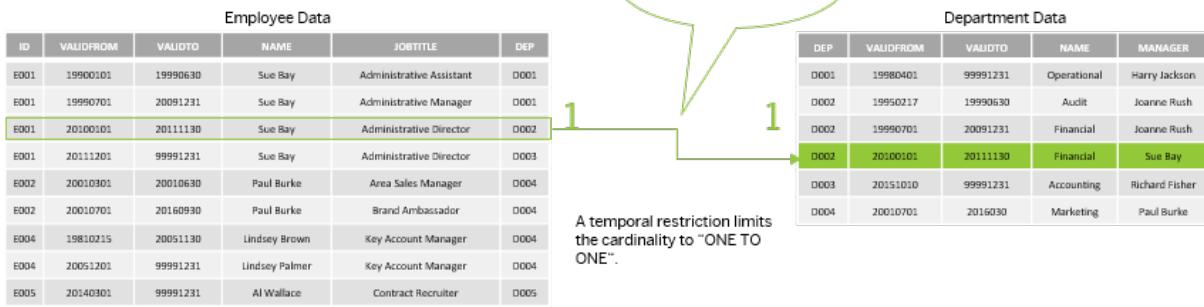
```
...
define view <CdsConsumptionView>
  as select from <data_source>
  association [1..*] to <target_entity> as <_association> on <condition_exp>
  {
    <view_element>
    ...
    @Consumption.filter.businessDate.at: true
    <_association>
  }
```

Results

The @Consumption annotation added to the associated element changes how the cardinality is treated. It specifies that the cardinality of the target data source of the associated view is set to the number of node records that are connected to a single source record ignoring the higher cardinality due to different valid times.

In the example above OData will expose the target source cardinality of the foreign key association _assoc as "ONE TO ONE" ([1..1]). If the temporal filter is included only one record is associated to one record of the source data.

```
@Consumption.filter.businessDate.at: true
<_Department>
```



Annotation to Reduce Association Cardinality

Entity Annotation

Context

In addition to the element annotations, which mark the boundaries of the validity period, there is an entity annotation that enables default filtering by system date.

Procedure

- Add `@consumption.filter.businessDate.at: true` to the CDS view.

```
...
@Consumption.filter.businessDate.at : true
define view <CdsConsumptionView>
  as select from <data_source>
  {   <view_element> }
```

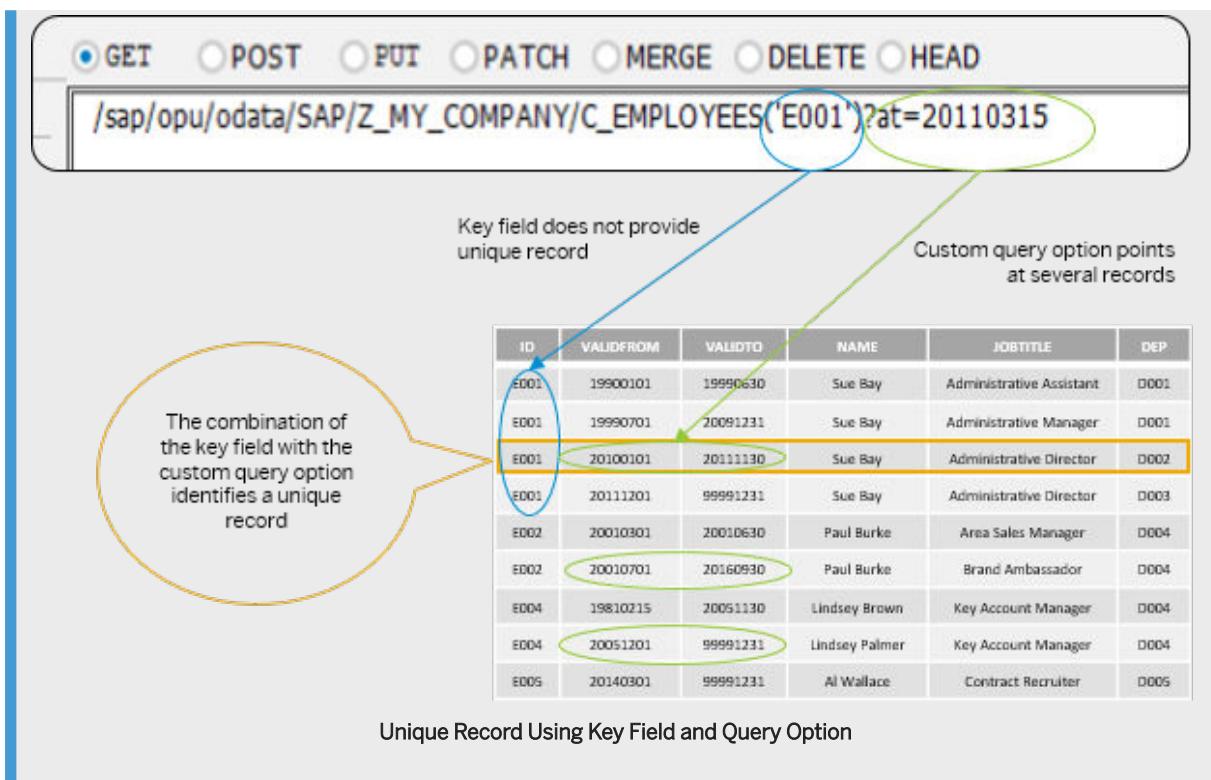
- As soon as the entity uses the `@consumption` annotation, the fields specifying the boundaries of the valid time must be annotated with the `@Semantics` annotation. Otherwise an exception will be thrown.

Results

If no query option is supplied by the OData client, this annotation automatically sets the query option as system date. Thus, every request is filtered automatically either by the date of the given custom query option or by the system date.

i Note

If the entity is annotated with the temporal filter annotation `@Consumption.filter.businessDate.at: true` no second field for the primary key is needed to identify a specific record. It is enough to use the non-temporal primary key in queries, as its uniqueness is guaranteed by the temporal filter (either by an automatic set or by the given date in the time travel query option).



Related Information

[Temporal Data \[page 237\]](#)

[Adding Annotations when Using a Temporal Parameter \[page 243\]](#)

[Applying Temporal Filtering in OData Requests \[page 245\]](#)

8.3.1.2 Adding Annotations when Using a Temporal Parameter

A view parameter can be filled with the value of a temporal custom query option of an OData request. To enable this the data model needs to be equipped with an annotation to the parameter.

Prerequisites

- The retrieval of filtered temporal data is not supported in transactional scenarios.
- The CDS elements that express the boundaries of the validity period must be of type DATS. No other format or greater granularity is supported.

- The data set must not contain overlapping time frames. Overlapping occurs when calendar days coincide in two or more different time frames. Every real calendar date must be clearly identifiable to only one valid time frame in the database or no time frame at all.

i Note

If there are discontinuances between periods, for example breaks between two subsequent periods, the output does not contain any data. The OData request then raises the regular exception “Not found”, as there is no data in the data set matching the request. From a semantic point of view, the entity is considered as nonexistent at the given point in time.

- If a time period has not expired yet the end of the validity period must be set to a sufficiently large value, preferably the largest possible value, for example December 31, 9999.

Context

A temporal parameter can be used in view modeling if the exposure of data depends on a certain condition or calculation where the parameter is used.

To enable the value to be passed from the custom query option `at=YYYYMMDD` in an OData request to a view parameter, the parameter needs an annotation. The parameter is hidden to avoid value confusion if both the parameter and the temporal custom query option are filled with a date value.

Procedure

- Add `@consumption.businessDate.at: true` to the temporal view parameter.

```
...
@Consumption.filter.businessDate.at : true
define view <CdsConsumptionView>
  with parameters
    @Semantics.businessDate.at: true
    @Consumption.hidden: true
    <p_at> : dats
    as select from <data_source>
    {   <view_element>   }
  where (   <validfrom>   <= :<p_at>
    and   <validto>   >= :<p_at> )
    or   <condition>
```

If the `@Consumption` annotation is missing on entity level and no value is supplied to the query option, an error is thrown, as no value can be passed to the parameter. If the annotation is on entity level, the parameter is filled with the system date.

Example

The following code example of a view is modeled to expose employees and their related information restricted by their respective validity periods. You want to retrieve data of every employee that is employed on a specific

date in addition to all founding members of a company. These founding members are always displayed. In this case the example view looks like this:

```
...
@Consumption.filter.businessDate.at : true
define view C_CurrentEmployees
  with parameters
    @Semantics.businessDate.at: true
    @Consumption.hidden: true
    p_at : dats
  as select from my_employees
{  key id,
  validfrom,
  validto,
  name,
  jobtitle,
  department,
  found_member }
where (      validfrom    <= :p_at
  and    validto       >= :p_at )
  or      found_member   = 1
```

The date value of the custom query option is passed to the parameter and is then applied in the where-clause. All employees that are employed on the date given with the custom query option and all founding members are exposed.

Related Information

[Temporal Data \[page 237\]](#)

[Applying Temporal Filtering in OData Requests \[page 245\]](#)

[Adding Annotations for Full Temporal Exposure \[page 239\]](#)

8.3.1.3 Applying Temporal Filtering in OData Requests

Temporal filtering in OData requests is used to filter temporal data by a specific date. This filtering is done by a custom query option added to the URI.

Prerequisites

- The CDS data model is annotated with the obligatory annotations to enable the retrieval of temporal data. For more information, see [Adding Annotations for Full Temporal Exposure \[page 239\]](#). For more information about annotations for a temporal parameter, see [Adding Annotations when Using a Temporal Parameter \[page 243\]](#).

Context

To filter temporal data by a specific validity date, a custom query option filled with the desired value is required. Only the data matching the given date in the custom query option is then retrieved. If the OData client does not supply a date in a custom query option, the system date is used as a filter reference.

→ Tip

Using the transaction `/iwfnd/gw_client` you can imitate an OData client (for example, a SAP Fiori application) and test activated services in the **SAP Gateway Client**.

The following listing demonstrates the form of a URI in the **SAP Gateway Client** with a custom query option:

```
.../sap/opu/odata/SAP/<service_name>/<EntitySet>?at=<time_data>
```

Procedure

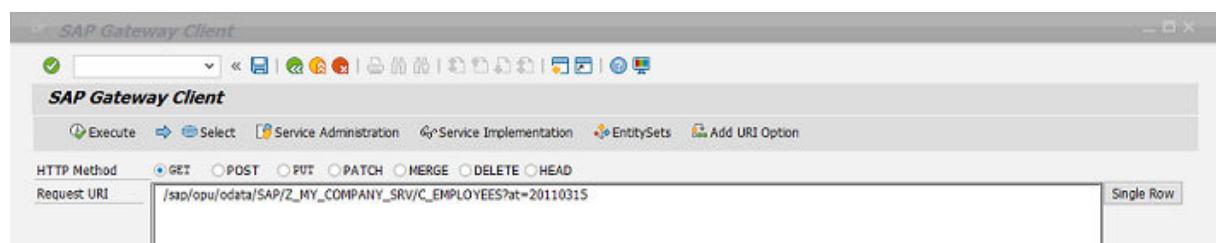
- To trigger the application of temporal filtering by a specific date, add the custom query option `at=YYYYYYMMDD` (DATS format) with the desired validity date to the URI.

Results

The data set is filtered and retrieved by the given date in the OData request or by the system date. Temporal filtering is applied to all requested CDS views that are equipped with the necessary annotations, including those queried with `$expand`. Hence, the OData request is answered while respecting temporal filtering where applicable.

Example

The following example illustrates how the **SAP Gateway Client** can be used to retrieve data with a custom query option to filter by validity date.



URI to Retrieve Temporally Restricted Data with the SAP Gateway Client

If the underlying CDS entities use the relevant annotations, the OData request with the custom query filters the data and exposes only employees and their respective information at the given point in time. The following

extract shows the OData output generated with the **SAP Gateway Client**. The example uses example data about employees and retrieves only that information which is valid on March 15, 2011 by using the custom query option `at=20110315`.

```

1  {
2   "d" : [
3    {
4     "__metadata" : {
5      "id" : "E001",
6      "validfrom" : "\/Date(63382680000)\/",
7      "validto" : "\/Date(129634200000)\/",
8      "name" : "Sue Bay",
9      "jobtitle" : "Administrative Manager",
10     "department" : "D002",
11     },
12    {
13     "__metadata" : {
14      "id" : "E002",
15      "validfrom" : "\/Date(99393840000)\/",
16      "validto" : "\/Date(147518640000)\/",
17      "name" : "Paul Burke",
18      "jobtitle" : "Brand Ambassador",
19      "department" : "D004",
20     },
21    {
22     "__metadata" : {
23      "id" : "E004",
24      "validfrom" : "\/Date(113339160000)\/",
25      "validto" : "\/Date(25340221080000)\/",
26      "name" : "Lindsey Brown",
27      "jobtitle" : "Key Account Manager",
28      "department" : "D004",
29     }
30   }
31 }
32
33
34
35
36
37
38
39
40
41

```

OData Output with Applied Temporal Time Travel Query Option at=20110315

Related Information

[Temporal Data \[page 237\]](#)

[Adding Annotations for Full Temporal Exposure \[page 239\]](#)

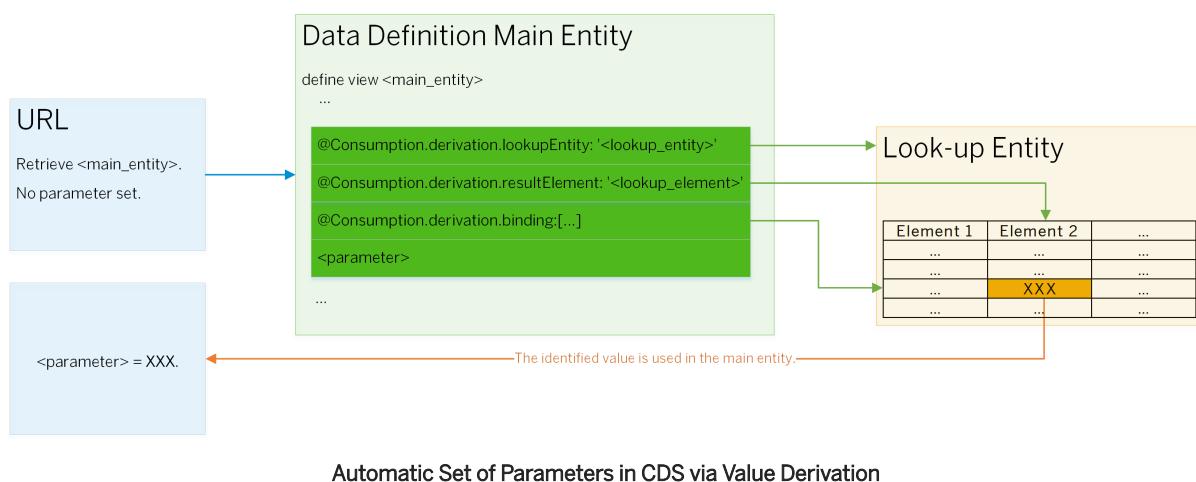
[Adding Annotations when Using a Temporal Parameter \[page 243\]](#)

8.4 Deriving Values from Foreign Entities

Values that are used as a filter on an element or to fill a parameter in a CDS entity can be derived from a foreign entity. This topic explains how this functionality is modeled in CDS.

The values for parameters or filters in CDS entities are usually given by an OData client at runtime. However, the derivation of values enables an automatic fill if no value is passed. A set of annotations triggers the passing of values from a foreign entity (look-up entity) to the main entity. These annotations are added to the parameter or the element in the main entity for which the value is derived.

As depicted in the figure below, the annotation set contains the navigation information to the value that is transferred from the look-up entity to the main entity. It specifies the look-up entity's name, one of its elements and how to single out one distinct rows where the value is taken from. The identified value is then passed back to be used to fill a parameter in the main entity at runtime.



Related Information

[Deriving Values for Parameters \[page 250\]](#)
[Deriving Single-Value Filters \[page 253\]](#)
[Deriving Filter Ranges \[page 256\]](#)
[Annotations for Derivations \[page 248\]](#)

8.4.1 Annotations for Derivations

The `@Consumption.derivation` annotations enable values to be derived from foreign entities by providing the navigation information to those values. These values are then passed to the main entity, where they are used at runtime.

The following list describes how the relevant `@Consumption.derivation` annotations are used. While only one value is derived to fill a parameter in the main entity, one or more values can be derived to be used as a filter in the main entity.

Annotation and Value	Effect
<code>@Consumption.derivation.lookup Entity: '<lookup_entity>'</code>	Identifies the CDS entity where values are derived from.
<code>@Consumption.derivation.result Element: '<lookup_element>'</code>	Identifies the element in the look-up entity from which values are taken. If a filter range is derived, it identifies the element in the look-up entity that specifies the lower limit of the range.

Annotation and Value	Effect	
@Consumption.derivation.result ElementHigh: '<lookup_element>'	Identifies the element in the look-up entity that specifies the upper limit of the range.	
	<p>i Note</p> <p>This annotation is only used for the derivation of filter ranges.</p>	
@Consumption.derivation.bindin g: [{...}]	<p>Singles out one or more rows in the look-up entity from which values are taken.</p> <p>For the binding, provide an array of three values:</p> <pre>[{<binding>: <binding_value>, type: <binding_type>, value: <type_value>}]</pre> <p>The binding specifies whether an input parameter or an element filter is used for the binding.</p>	
	Binding	Description
	<p>targetParameter: '<lookup_parameter>'</p>	Uses an input parameter in the look-up entity to single out one or more rows from which the values for the derivation are taken.
	<p>targetElement: '<lookup_element>'</p>	Applies a filter on an element in the look-up entity to single out one or more rows from which values for the derivation are taken.
	The binding type specifies how the binding is filled:	
	Binding Type and Value	Description
	<p>type: #CONSTANT value: '<constant>'</p>	Uses a constant to fill the value of the binding.
	<p>type: #PARAMETER value: '<main_param>'</p>	Uses a parameter in the main entity to fill the value of the binding.
	<p>type: #SYSTEM_FIELD value: '#<system_variable>'</p>	Uses a system variable to fill the value of the binding.

Related Information

[Deriving Values for Parameters \[page 250\]](#)

[Deriving Single-Value Filters \[page 253\]](#)

[Deriving Filter Ranges \[page 256\]](#)

8.4.1.1 Deriving Values for Parameters

A value for a parameter can be set automatically in CDS by deriving it from another entity. This topic describes how to annotate the parameter for which the value is derived.

Prerequisites

The annotations for the derivation of values are only applied if no value is given by the OData client. This is the case in the following instances

- The parameter is hidden, which means it is annotated with `@Consumption.hidden:true` and therefore is not available for the client
- The OData client supplies the initial value of the parameter.

Context

A parameter of a CDS entity (in the following called main entity) is usually filled with a suitable value by the OData client. Nevertheless, there are use cases in which you want the value for the parameter to be dependent on a value of another entity. In this case, the value for the parameter can be derived from a foreign entity, known as a look-up entity. To do this, equip the parameter with a set of annotations that provides the navigation information to a field in the look-up entity. The value of this field is then passed to fill the parameter in the main entity.

Values for parameters are automatically derived from foreign entities if the parameter is hidden for the OData Client. Otherwise, if the parameter is not hidden, the value is only derived if the OData Client supplies the initial value of the parameter.

Procedure

- Determine the look-up entity from which the value is passed to the parameter in the main entity and add the following annotation to this parameter:

```
@Consumption.derivation.lookupEntity: '<lookup_entity>'
```

Enter the name of the look-up entity as the value of the annotation.

- Determine the element in the look-up entity from which the value is passed to the parameter in the main entity and add the following annotation to this parameter:

```
@Consumption.derivation.resultElement: '<lookup_element>'
```

Enter the name of the element in the look-up entity as the value for the annotation.

- Determine the row in the look-up entity which matches the value that is passed to the main entity.
 - Choose a binding to identify the row in the look-up entity. It might be necessary to use more than one binding to identify the value that is passed to the main entity.

Binding	Description
targetParameter	Choose a parameter binding if you want to single out a row by using a parameter in the look-up entity. Enter the parameter as a literal: '<lookup_parameter>'.
targetElement	Choose an element binding if you want to single out a row by setting a filter on an element in the look-up entity. Enter the element as a literal: '<lookup_element>'.

- Choose a binding type to specify how the binding is filled.

Binding	Description and Value
#CONSTANT	Choose this type if you want to fill the binding with a constant. As the value, enter the constant you want the binding to be filled with as a literal: '<constant>'.
#PARAMETER	Choose this type if you want to fill the binding with the value of a parameter in the main entity. As the value, enter the name of the parameter in the main entity as a literal '<main_param>'.
#SYSTEM_FIELD	Choose this type if you want to fill the binding with a system variable. Enter one of the following as the value: '#SYSTEM_LANGUAGE' '#SYSTEM_DATE' '#SYSTEM_TIME' '#USER' '#CLIENT'.

- Add the following annotation with the relevant binding, type and value.

```
@Consumption.derivation.binding: [{targetParameter | targetElement :  
'<lookup_parameter>' | '<lookup_element>',  
type: #CONSTANT | #PARAMETER | #SYSTEM_FIELD,  
value:'<binding_value>' }]
```

Example

The CDS view `C_PurchaseOrderItems` exposes information about purchase order items for a specific time period. The end of the period is predefined as the system date. The entity has an input parameter `p_start` that defines the beginning of the period. The value of this parameter can be given by the OData client. However, if the value is not supplied, it is set automatically with the start of the current business quarter. This value is stored in the CDS view `C_BusinessQuarters` and must be derived from there.

The annotations identify the look-up entity `C_BusinessQuarters` and the element `q_beginning` from which the value is derived. The binding is applied by the parameter `p-day` in the look-up entity. This binding parameter is filled with the system date because the derivation annotation in the main entity discloses the binding type `#SYSTEM_FIELD`. The system date is associated with the relevant business quarter and the beginning of this quarter is then passed to the parameter `p_start` in the main entity. In this scenario, the entity exposes all purchase order items that were delivered in the current quarter until today.

The following listings display the data models for the main entity and the look-up entity.

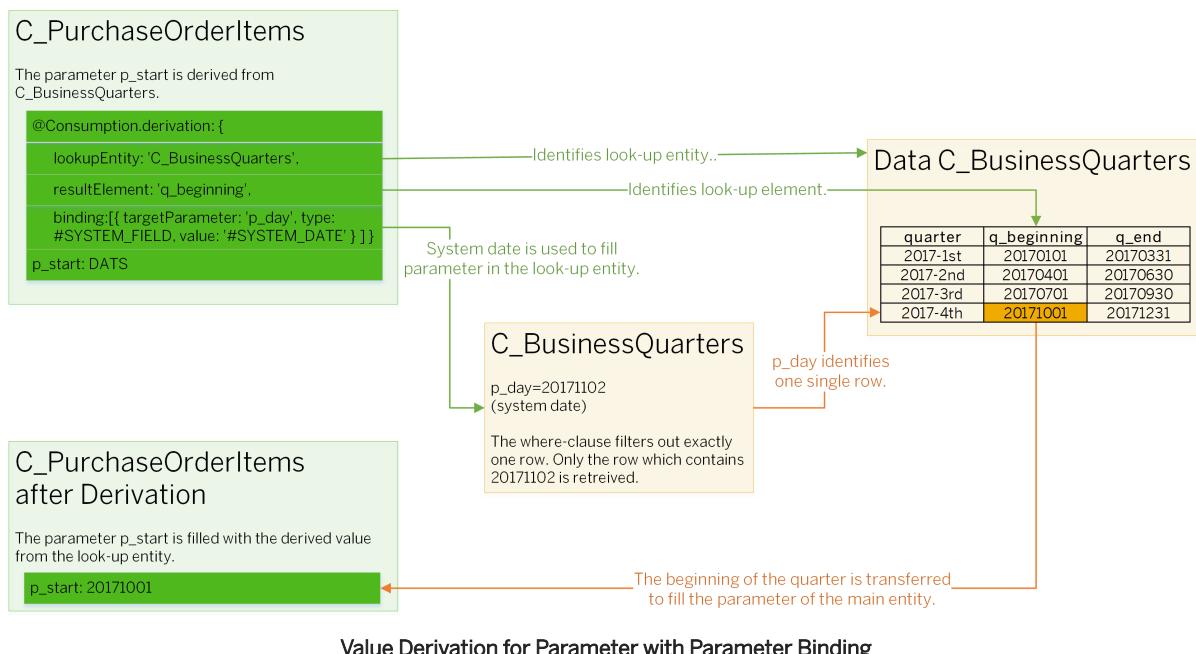
Main Entity:

```
define view C_PurchaseOrderItems
  with parameters
    @Consumption.derivation: {
      lookupEntity: 'C_BusinessQuarters',
      resultElement: 'q_beginning',
      binding:[{targetParameter: 'p_day', type: #SYSTEM_FIELD, value:
        '#SYSTEM_DATE'}] }
      p_start: DATS
    as select from db_poi
    { key      po_id      as PurchaseOrder,
      key      po_item_pos as PurchaseOrderItem,
      ...
      delivery_date   as DeliveryDate       }
    where   (   p_start      <= :DeliveryDate
      and     DeliveryDate    <= $session.system_date      )
```

Look-Up Entity:

```
define view C_BusinessQuarters
  with parameters
    p_day: DATS
  as select from db_quarters
  { key      quarter,
    q_beginning
    q_end    }
  where   (   q_beginning    <= :p_day
    and     q_end        >= :p_day      )
```

The following figure illustrates this example.



Related Information

[Deriving Single-Value Filters \[page 253\]](#)

[Deriving Filter Ranges \[page 256\]](#)

[Annotations for Derivations \[page 248\]](#)

[Annotations for Derivations \[page 248\]](#)

8.4.1.2 Deriving Single-Value Filters

A filter on an element can be set automatically in CDS by deriving it from another entity. This topic describes how the element is annotated to filter for single values.

Prerequisites

The annotations for the filter derivations are only applied if no filter is given by the OData client. If a filter is supplied the annotations in the CDS entity are ignored.

Context

An element of a CDS entity (in the following called main entity) can be filtered by the OData client. Nevertheless, there are use cases, in which you want the filter to be generated automatically depending on a value of another entity. In this case, the filter can be derived from a foreign entity, known as a look-up entity. To

do this, equip the element in the main entity with a set of annotations. These annotations provide the navigation information to the field with the filter criterion in the look-up entity. The value of this field is then passed to the main entity to be used as a filter on the element.

It is also possible that the navigation information leads to several fields of an element in the look-up entity. In this case all values are applied as filter criteria for the filter in the main entity.

If the OData client supplies a filter, the annotations in the CDS entity are ignored.

Procedure

- Determine the look-up entity from which the filter criteria are passed and add the following annotation to the respective element in the main entity:

```
@Consumption.derivation.lookupEntity: '<lookup_entity>'
```

Enter the name of the look-up entity as the value of the annotation.

- Determine the element in the look-up entity from which the filter criteria are passed and add the following annotation to the respective element in the main entity:

```
@Consumption.derivation.resultElement: '<lookup_element>'
```

Enter the name of the element in the look-up entity as the value of the annotation.

- Determine the row or rows in the look-up entity that provide the filter criteria that are passed to the main entity.
 - Choose a binding to identify the row or rows in the look-up entity. It is possible to use more than one binding to single out the filter criteria.

Binding	Description
targetParameter	Choose parameter binding if you want to single out one or more rows by using a parameter in the look-up entity. Enter the parameter as a literal: '<lookup_parameter>'.
targetElement	Choose element binding if you want to single out one or more rows by setting a filter on an element. Enter the element as a literal: '<lookup_element>'.

- Choose a binding type to specify how the binding is filled.

Binding Type	Description and Value
#CONSTANT	Choose this type if you want to fill the binding with a constant. As the value, enter the constant you want the binding to be filled with as literal: '<constant>'.

Binding Type	Description and Value
#PARAMETER	<p>Choose this type if you want to fill the binding with the value of a parameter in the main entity.</p> <p>As the value, enter the name of the parameter in the main entity as literal '<code><main_param></code>'.</p>
#SYSTEM_FIELD	<p>Choose this type if you want to fill the binding with a system variable.</p> <p>Enter one of the following as the value: '#SYSTEM_LANGUAGE' '#SYSTEM_DATE' '#SYSTEM_TIME' '#USER' '#CLIENT'.</p>

- Add the following annotation with the relevant binding, type and value.

```
@Consumption.derivation.binding: [{targetParameter | targetElement : 
'<lookup_parameter>' | '<lookup_element>',
type: #CONSTANT | #PARAMETER | #SYSTEM_FIELD,
value:'<binding_value>' }]
```

Example

The CDS entity `C_SalesOrdersCountries` exposes information about sales orders with buyers from a certain country. The relevant country is given by the OData client in the input parameter `p_country`. A filter is applied to the element that provides the information about the buyers. Only those sales orders are retrieved that list buyers from the given country.

The values for the filter on the element `buyer` in the main entity are derived from the look-up entity `C_BusinessPartners`. In this entity, it is the element `id`, from which the values for the filter are taken. The binding is applied by a filter on the element `country` with the filter criteria given by the input parameter of the main entity `p_country`. The buyers in those rows that were singled out are then passed to the main entity and function as a filter for the element `buyer`. This means that only those buyers that match the country criterion are passed to the main entity as filter criteria for the sales orders.

If the parameter `p_country` is supplied with 'Germany' by the OData client, only the sales orders with buyers located in Germany are retrieved.

The following listings display the data models for the main entity and the look-up entity.

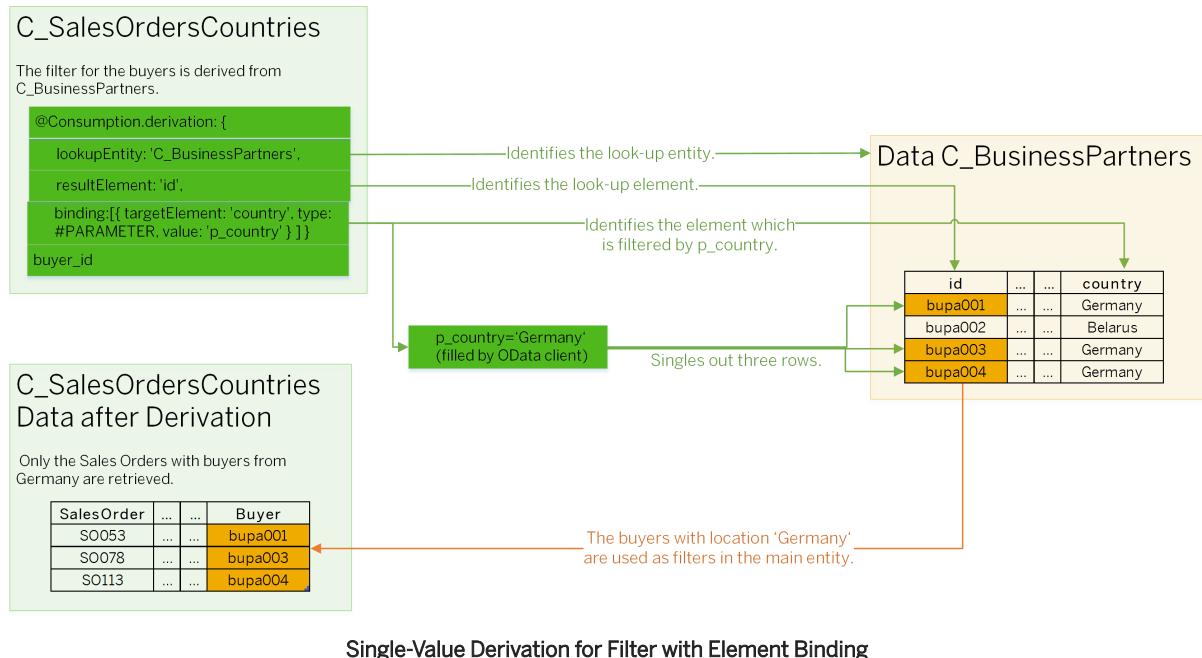
Main Entity:

```
define view C_SalesOrdersCountries
  with parameters
    p_country: string
  as select from db_salesorders
  { key   salesorder_id      as SalesOrder,
  ...
  @Consumption.derivation: {
    lookupEntity: 'C_BusinessPartners',
    resultElement: 'Id',
    binding:[{targetElement: 'country', type: #PARAMETER, value:
'p_country'}] }
    buyer_id       as Buyer      }
```

Look-Up Entity:

```
define view C_BusinessPartners
as select from db_businesspartners
{ key      id,
  ...
  country }
```

The following figure illustrates this example:



Related Information

[Annotations for Derivations \[page 248\]](#)

[Deriving Values for Parameters \[page 250\]](#)

[Deriving Filter Ranges \[page 256\]](#)

8.4.1.3 Deriving Filter Ranges

A filter on an element can be set automatically in CDS by deriving it from another entity. This topic describes how to annotate the element to filter for a range of values.

Prerequisites

The annotations for the filter derivations are only applied if no filter is given by the OData client. If a filter is supplied the annotations in the CDS entity are ignored.

Context

Elements of CDS entities (in the following called main entity) can be filtered by a range of values if you want to retrieve all entries for values that fall into a certain range. If you want this filter to be generated automatically depending on a range of values from a foreign entity, known as a look-up entity, you can derive the values for the filter range. To do this, equip the element in the main entity with a set of annotations. These annotations provide the navigation information to the fields in the look-up entity with the range limits. The values of these fields are then passed to the main entity to be used as filter criteria on the element.

It is possible that the navigation information leads to several ranges in the look-up entity. In this case all ranges are applied as filter criteria for the filter in the main entity.

If the OData client supplies a filter, the annotations in the CDS entity are ignored.

Procedure

- Determine the look-up entity from which the filter criteria are passed and add the following annotation to the respective element in the main entity:

```
@Consumption.derivation.lookupEntity: '<lookup_entity>'
```

Enter the name of the look-up entity as the value of the annotation.

- Determine the elements in the look-up entity that specify the lower limit and the upper limit of the range and add the following annotations to the respective element:

```
@Consumption.derivation.resultElement: '<lower_limit>'  
@Consumption.derivation.resultElementHigh: '<upper_limit>'
```

As the value of these annotations, enter the name of the elements that specify the lower limit and the upper limit in the look-up entity.

- Determine the row or rows in the look-up entity that provide the filter criteria that are passed to the main entity.
 - Choose a binding to identify the row or rows in the look-up entity. It is possible to use more than one binding to single out the filter criteria.

Binding	Description
targetParameter	Choose parameter binding if you want to single out one or more rows by using a parameter in the look-up entity. Enter the parameter as a literal: '<lookup_parameter>'.
targetElement	Choose element binding if you want to single out one or more rows by setting a filter on an element. Enter the element as a literal: '<lookup_element>'.

- Choose a binding type to specify how the binding is filled.

Binding Type	Description and Value
#CONSTANT	<p>Choose this type if you want to fill the binding with a constant.</p> <p>As the value, enter the constant you want the binding to be filled with as a literal: '<constant>'.</p>
#PARAMETER	<p>Choose this type if you want to fill the binding with the value of a parameter in the main entity.</p> <p>As the value, enter the name of the parameter in the main entity as a literal '<main_param>'.</p>
#SYSTEM_FIELD	<p>Choose this type if you want to fill the binding with a system variable.</p> <p>Enter one of the following as the value: '#SYSTEM_LANGUAGE' '#SYSTEM_DATE' '#SYSTEM_TIME' '#USER' '#CLIENT'.</p>

- Add the following annotation with the relevant binding, type and value.

```
@Consumption.derivation.binding: [{targetParameter | targetElement :
'<lookup_parameter>' | '<lookup_element>',
type: #CONSTANT | #PARAMETER | #SYSTEM_FIELD,
value:'<binding_value>' }]
```

Example

The CDS entity `C_SalesOrderQuarterly` exposes information about sales orders for a specific business quarter. The relevant business quarter is given by the OData client in the input parameter `p_quarter` in the main entity. A filter is applied to the element that provides the information about the sales order date to retrieve only those sales orders that fall into the given business quarter.

The sales order dates are filtered for a specific business quarter. The limits for the selected business quarter are derived from the look-up entity `C_BusinessQuarters`. In this entity, it is the element `q_beginning` that defines the lower limit and the element `q_end` that defines the upper limit of the business quarter.

The business quarter is determined by the input parameter `p_quarter`, which singles out one row. The limits of the given business quarter are taken from this row according to the elements that are specified by the annotation. The sales order dates are compared with these two values; and only those sales orders whose dates match the limits are retrieved.

The following listings display the data models for the main entity and the look-up entity.

Main Entity:

```
define view C_SalesOrdersQuarterly
  with parameters
    p_quarter:abap.char( 8 )
  as select from db_so
  { key      so_id          as SalesOrder,
    ...
    @Consumption.derivation: {
      lookupEntity: 'C_BusinessQuarters',
      resultElement: 'q_beginning',
```

```

        resultElementHigh: 'q_end',
        binding:[{targetElement: 'quarter', type: #PARAMETER, value:
'p_quarter'}] }
      so_date           as Date
}

```

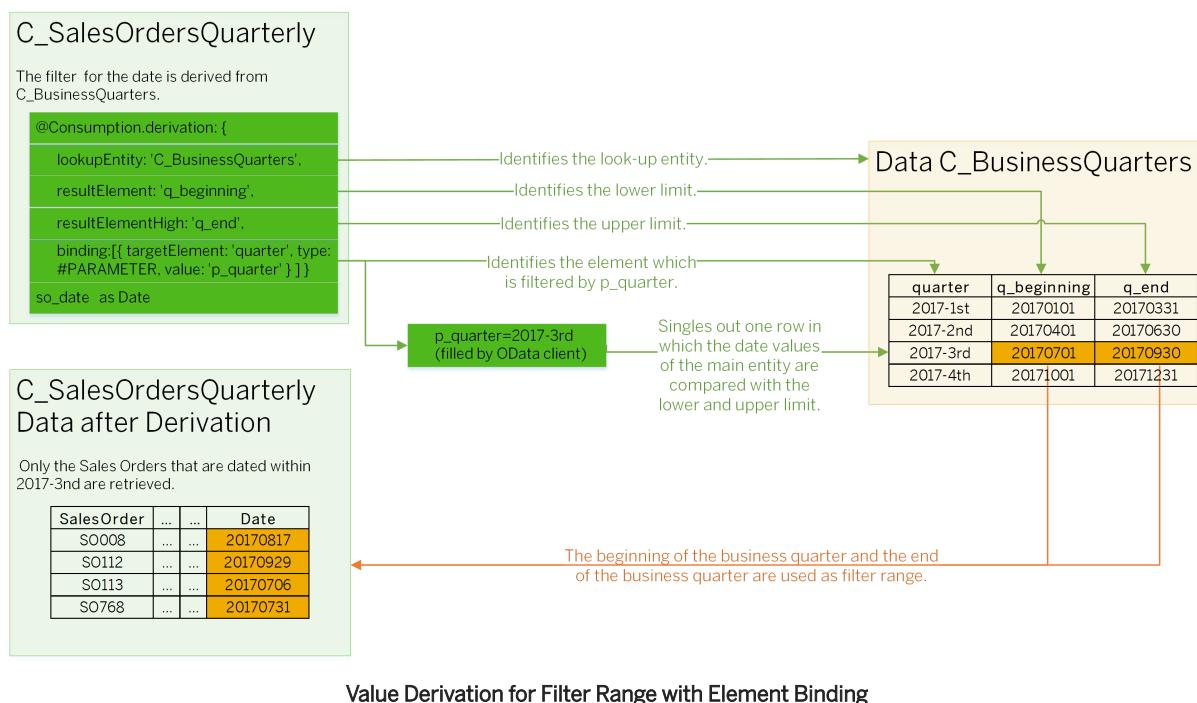
Look-Up Entity:

```

define view C_BusinessQuarters
as select from db_quarters
{ key quarter,
  q_beginning
  q_end
}

```

The following figure illustrates this example:



8.5 Enabling Text and Fuzzy Searches in SAP Fiori Apps

The descriptions in this topic refer to the range of functions for text and fuzzy searches that are provided in the context of SAP HANA.

Text and Fuzzy Searches

The full text searching (or just text search) provides you with the capability to identify natural language terms that satisfy a query and, optionally, to sort them by relevance (ranking) to the query. The most common type of search is to find texts that contain the term specified and return them in the order of their similarity to these terms.

Fuzzy search is a fast and fault-tolerant search feature of SAP HANA. The basic concept behind the **fault-tolerant search** is that a database query returns records even if the search term (user input) contains

additional or missing characters, or even spelling errors. Fuzzy search can be used in various applications -- for example, to trigger a fault-tolerant search in a structured database content, like a search for a product called 'coffe krisp biscuit' and you find 'Toffee Crisp Biscuits'.

Providing Freestyle Search Capabilities in SAP Fiori UI screen

Within the context of the ABAP programming model for SAP Fiori, you only need to enable the text and fuzzy search functionality in your data model definitions. For this purpose, you implement it in designated CDS views using appropriate text and fuzzy search annotations (listed below).

i Note

As an application developer however, you must ensure that your CDS views are suitable for text and fuzzy search enabling. For more information take a look at the corresponding topics in the [SAP HANA Developer Guide](#).

Annotations for Text- and Fuzzy Search

As the name suggests, search annotations enable the search feature on the CDS view elements.

First of all, you need the following CDS annotation at the **view level**:

Annotation and Value	Effect
<code>@Search.searchable: true/false</code>	Defines whether a CDS view is generally relevant for search scenarios. This annotation provides a general switch and a means to quickly detect whether a view is search-relevant or not. Set to value true in order to enable search support by means of @Search annotations. Here, at least one view field must be defined as <code>@defaultSearchElement</code> at element level.

The annotations (required) at the **element level** are:

Annotation and Values	Effect
<code>@Search.defaultSearchElement: true/false</code>	<p>Specifies that the annotated element is to be considered in a full-text search</p> <p>i Note</p> <p>At least one element has to be defined for the default full-text search. Searching in views without default full-text search elements is not supported!</p>
	<p>All view elements that are annotated for the default search define the search scope. (The search will be performed on all elements that have this annotation.).</p> <p>⚠ Caution</p> <p>Such a search must not operate on all elements – for performance reasons and because not all elements qualify for this kind of access.</p>
<code>@Search.fuzzinessThreshold : <value></code>	<p>This annotation specifies the least level of fuzziness the element has to have in order to be considered in a fuzzy search at all.</p> <p>The <code><value></code> defines the threshold for a fuzzy search (how fuzzy scores are calculated when comparing two strings or two terms).</p> <p>Possible values are: 0 .. 1 The default value is 1. The fuzzy search algorithm calculates a fuzzy score for each string comparison. The higher the score, the more similar the strings are. A score of 1.0 means the strings are identical. A score of 0.0 means the strings have nothing in common.</p>
<code>@Search.ranking: <value></code>	<p>This annotation specifies how relevant the values of an element (view field) are for ranking, should the freestyle search terms match the element's value.</p> <p>The ranking can have the following values:</p> <ul style="list-style-type: none">• HIGH - The element is of high relevance; typically, this is useful for IDs and their descriptions.• MEDIUM - The element is of medium relevance; designated usually for important elements.• LOW - Although the element is relevant for a freestyle search, a hit for this element has no real significance for the ranking of a result item.

→ Tip

For the fuzzy search threshold, we recommend using the default value 0.7 to start with. Later on, you can fine-tune the value based on your experiences with the search. You can also fine-tune the search using feedback collected from your users.

Example

The listing below implements a search model for searching products. The model definition results from a join between two data sources that already specify the persistence layer for searching: both database tables are

part of the EPM data model, where SMWD_PD provides product data and the table SMWD_TEXTS serves as the text reference table.

The annotation `@Search.searchable: true` marks the view as searchable. In addition, the elements `Name` and `Category` are annotated with `@Search.defaultSearchElement: true`. This means that a freestyle search is enabled on the search UI where it is possible to search for the annotated elements. The annotation `@Search.fuzzinessThreshold: 0.7 (0.8)` defines that the text search should be applied to the element `Category` with a similarity value of 70% and to the element `Name` with a similarity value of 80%.

```
...
@Search.searchable : true

define view SearchForProducts as select from smwd_pd as Product
  left outer join smwd_texts as ProductDescription
    on Product.desc_guid = ProductDescription.node_key
{
  key Product.product_id as ID,
  @Search.defaultSearchElement : false
  ProductDescription.language as Language,
  @Search.defaultSearchElement : true
  @Search.fuzzinessThreshold : 0.8
  @Search.ranking : #HIGH
  ProductDescription.text as Name,
  @Search.defaultSearchElement : true
  @Search.fuzzinessThreshold : 0.7
  @Search.ranking : #LOW
  Product.category as Category
}
```

Preview:

The screenshot shows a search interface with a filter bar at the top. The filter bar has a red rounded rectangle around it, and a yellow box highlights the input field containing "Notebooks HT-". To the right of the input field is a red close button (X). To the left of the input field is a red "Standard *" label with a red checkmark icon. To the right of the input field is a blue "Hide Filter Bar" link. A tooltip arrow points to the yellow box with the text "To show filters here, add them to the filter bar in Filters". Below the filter bar is a table with two columns: "Product Category" and "Product ID". The table contains four rows, each with "Notebooks" in the category column and a unique ID in the ID column: "HT-1000", "HT-1001", and "HT-1002". At the bottom of the table is a caption: "Standard filter allows to search for product category and product name".

Product Category	Product ID
Notebooks	HT-1000
Notebooks	HT-1001
Notebooks	HT-1002

Standard filter allows to search for product category and product name

Related Information

[Search Annotations \[page 442\]](#)

8.6 Using Aggregate Data in SAP Fiori Apps

This topic explains how you can provide aggregate data for your SAP Fiori application. The available aggregate functions operations are sum, minimum, maximum, and average. Alongside this, the framework also provides options for counting.

What is Aggregate Data?

We speak of aggregate data when numerical values are combined to form a single value that signifies meaning. General assumptions can be drawn from this value that is representative for all values that were included in the calculation of the value.

The classic aggregate functions are:

- sum
- minimum
- maximum
- average

These functions determine a value from which you can assume information relating to all the values that were included in the calculation.

Apart from these functions, the following counting functions are also available:

- count
- distinct count.

These counting options determine a natural number based on the number of entries in the calculation.

All of these functions are supported by the SADL framework.

Aggregate Data in your SAP Fiori App

Aggregate data calculated by the SADL framework provides additional and enhanced information for your list reporting app.

To display aggregate data in your application, annotate the respective elements in CDS with the annotations described in [Annotating Aggregate Functions in CDS \[page 264\]](#). These annotations cause the CDS entity to be respected as an aggregated entity by OData. A thorough description of how OData interprets the annotations is provided in [OData Interpretation of Aggregation Annotations \[page 267\]](#).

Based on the entity that calculates aggregate data, the SAP Fiori user interface displays aggregate data depending on the settings you choose. The aggregated values are displayed in your list reporting app as an additional field in the relevant column.

Sales Order	Item	Company	Product	Category	Gross Amount	Gross Amount in EUR	Net Amount	Tax Amount	Number of Prod...	Distinct Products
500007257	50	Panorama Studios	HT-1036	Flat Screen Monitors	1,023.40 CAD	725.82 EUR	860.00 CAD	163.40 CAD	1	1
500007257	60	Panorama Studios	HT-1037	Flat Screen Monitors	1,463.70 MXN	159.44 EUR	1,230.00 MXN	233.70 MXN	1	1
500007257	70	Panorama Studios	HT-1000	Notebooks	3,412.92 EUR	3,412.92 EUR	2,868.00 EUR	544.92 EUR	1	1
500007257	80	Panorama Studios	HT-1001	Notebooks	1,486.42 EUR	1,486.42 EUR	1,249.09 EUR	237.33 EUR	1	1
Aggregate Data					Show Details	9,367.78 EUR	Show Details	Show Details	8	7

List Reporting App of Sales Order Items with Aggregate Data

8.6.1 Annotating Aggregate Functions in CDS

The elements for which you want to display aggregate data in your SAP Fiori App must be annotated in the CDS entity with the relevant annotation for the aggregate function.

Metadata for Aggregations

The annotation `@Aggregation.Default: #<AGGR_FUNCTION>` enables the aggregation of the values of the annotated element.

Only measures can be annotated with an aggregation annotation. Measures are elements that either represent numerical values, which means they can be summed, averaged, or otherwise mathematically manipulated. In addition, elements with date values, can also be compared with each other to determine the maximum or minimum. Date values can also be measures. Typically, measures are units that express the size, amount, or degree of something, for example prices.

The other elements in a CDS entity are called dimensions. Dimensions provide structured labeling information about otherwise unordered numeric measures. They are relevant for the grouping and the order of the elements in the Fiori App.

The SADL framework supports the following aggregating functions for measures:

Annotation and Value	Effect
<code>@Aggregation.Default: #SUM</code>	Calculates the sum of the values of the annotated element.
<code>@Aggregation.Default: #MAX</code>	Calculates the maximum of the values of the annotated element.
<code>@Aggregation.Default: #MIN</code>	Calculates the minimum of the values of the annotated element.
<code>@Aggregation.Default: #AVG</code>	Calculates the average of the values of the annotated element.

Annotation and Value	Effect
@Aggregation.Default: #COUNT	Counts the number of entries of the annotated element.
	<p>i Note</p> <p>The value of the annotated element is always displayed as 1 for single data records, regardless of the actual value of the element. Only when an aggregated value is requested is the count value displayed.</p> <p>For this reason, it is recommended that a new element for the counting of elements be created. Make sure that you use an adequate numerical type for this element.</p>
<p>@Aggregation.Default: #COUNT_DISTINCT</p> <p>To count distinct values of a different element, use the subannotation @Aggregation.ReferenceElement: ['elementRef'] with the respective element as a value. The aggregate function then counts the values of the element which is specified as a reference.</p>	

Only one aggregate function can be used on one element. You cannot display different aggregated values of the same element.

Metadata for the Presentation in the UI

Apart from the annotations that trigger the aggregation of measure values, it is vital to use an annotation that defines the properties that must always be selected in the OData request. In our analytical operations case, the elements that must always be selected are the technical keys. This means that, even if the columns of the technical keys are not selected by the UI, the OData request selects the technical keys to avoid grouping.

For more information on grouping, see [OData Interpretation of Aggregation Annotations \[page 267\]](#).

If the UI requests group records, the technical keys do not need to be selected since grouping is desired.

The following annotation must be used on entity level to specify the key(s). The annotation ensures that the technical keys are selected when needed, even if the columns with the technical keys are not selected by the UI.

Annotation and Value	Effect
<pre>@UI.presentationVariant: [{requestAtLeast: ['<TECHNICAL_KEY>']}]</pre>	Defines the properties that must always be included in the result of the queried collection if no grouping is desired.

Example

The following listing displays a CDS entity in which all the necessary annotations for analytical operations are used. This view describes sales order combined with associated product and customer information.

```
@UI.presentationVariant: [{requestAtLeast: ['SalesOrderID', 'ItemPosition']}]
define view ZDEMO_C_SOI_ANLY
  as select from <data source> as Item
    association [0..1] to SEPM_I_Product_E as _Product      on $projection.Product =
      _Product.Product
    association [0..1] to ZDEMO_C_SO_ANLY as _SalesOrder on
      $projection.SalesOrderID = _SalesOrder.SalesOrder
  {
    key SalesOrder
    SalesOrderID,                                as
    key SalesOrderItem                           as
    ItemPosition,                               as
    Item._SalesOrder.Customer                  as
    CustomerID,
      @Consumption.groupWithElement: 'CustomerID'
      Item._SalesOrder._Customer.CompanyName   as
    CompanyName,
      @ObjectModel.foreignKey.association: '_Product'
      Product                                as Product,
      @Consumption.groupWithElement: 'Product'
      _Product.ProductCategory                 as
    ProductCategory,
      @Semantics.currencyCode: true
      TransactionCurrency                     as
    CurrencyCode,
      @Semantics.currencyCode: true
      cast( 'EUR' as abap.cuky )             as
    TargetCurrency,
      @Aggregation.Default: #SUM
      @Semantics.amount.currencyCode: 'TargetCurrency'
      CURRENCY_CONVERSION(
        amount          => Item.GrossAmountInTransacCurrency,
        source_currency => Item.TransactionCurrency,
        target_currency => cast( 'EUR' as abap.cuky ),
        exchange_rate_date => cast( '20180315' as abap.dats ),
        error_handling   => 'SET_TO_NULL' )       as
    ConvertedGrossAmount,
      @Semantics.amount.currencyCode: 'CurrencyCode'
      @Aggregation.Default: #AVG
      GrossAmountInTransacCurrency           as
    GrossAmount,
      @Semantics.amount.currencyCode: 'CurrencyCode'
      @Aggregation.Default: #MIN
      NetAmountInTransactionCurrency         as NetAmount,
      @Semantics.amount.currencyCode: 'CurrencyCode'
      @Aggregation.Default: #MAX
      TaxAmountInTransactionCurrency          as TaxAmount,
      @Aggregation.Default: #COUNT
```

```

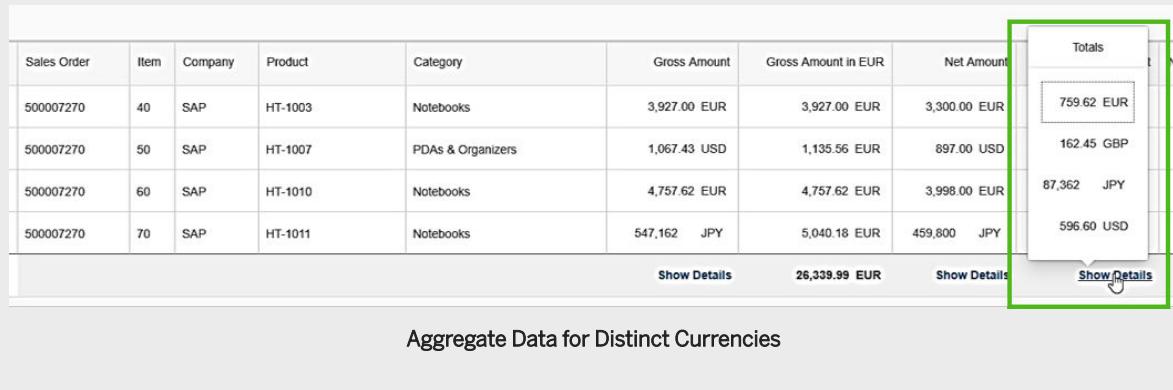
        cast ( 1 as abap.int4 )                                as
AllProducts,
    @Aggregation.referenceElement: ['Product']
    @Aggregation.Default: #COUNT_DISTINCT
    cast( 15 as abap.int4 )                                as
DistinctProducts,
    _SalesOrder,
    _Product
}

```

i Note

The aggregate functions only respect values with the same semantics. This means, that, if you have prices in different currencies that are annotated with an aggregation annotation, you receive aggregated data for each currency.

The following figure displays the maximum tax amount with regard to the respective currency.



The screenshot shows a table titled "Aggregate Data for Distinct Currencies". The table has columns: Sales Order, Item, Company, Product, Category, Gross Amount, Gross Amount in EUR, Net Amount, and Show Details. There are four rows of data. To the right of the table is a summary section titled "Totals" with four entries: 759.62 EUR, 162.45 GBP, 87,362 JPY, and 596.60 USD. Each entry has a "Show Details" link below it. A green box highlights the "Totals" section.

Sales Order	Item	Company	Product	Category	Gross Amount	Gross Amount in EUR	Net Amount	
500007270	40	SAP	HT-1003	Notebooks	3,927.00 EUR	3,927.00 EUR	3,300.00 EUR	Show Details
500007270	50	SAP	HT-1007	PDAs & Organizers	1,067.43 USD	1,135.56 EUR	897.00 USD	Show Details
500007270	60	SAP	HT-1010	Notebooks	4,757.62 EUR	4,757.62 EUR	3,998.00 EUR	Show Details
500007270	70	SAP	HT-1011	Notebooks	547.162 JPY	5,040.18 EUR	459.600 JPY	Show Details

Aggregate Data for Distinct Currencies

Related Information

[Using Aggregate Data in SAP Fiori Apps \[page 263\]](#)

[OData Interpretation of Aggregation Annotations \[page 267\]](#)

8.6.2 OData Interpretation of Aggregation Annotations

The following sections provide an overview of the most prominent features of aggregated entities in OData.

Data models with aggregation annotations are considered as aggregated entities by OData. The behavior of these aggregated OData entities differs from non-aggregated entities.

OData Annotations

The OData entity is given multiple annotations based on the aggregate annotation you use in CDS for your data model. The following figure displays the metadata of an aggregated entity that processes sales order items.

The annotations specific to aggregations are highlighted and labeled. Further descriptions of the annotations in OData are given below.

```

Aggregated OData Entity
- <EntityType sap:content-version="1" sap:label="Sales Order Items" sap:semantics="aggregate" Name="ZDEMO_C_SOI_ANLYType">
  - <Key>
    <PropertyRef Name="ID"/>
Generated<Key>
ID <Property Name="ID" sap:filterable="false" sap:sortable="false" Nullable="false" Type="Edm.String"/>
<Property sap:label="Sales Order ID" Name="SalesOrderID" Type="Edm.String" sap:quickinfo="EPM: Sales Order Number" sap:display-format="UpperCase" sap:aggregation-role="dimension" MaxLength="10" sap:updatable="false" sap:createable="false" sap:value-list="standard"/>
<Property sap:label="Item Position" Name="ItemPosition" Type="Edm.String" sap:quickinfo="EPM: Sales Order Item Position" sap:display-format="UpperCase" sap:aggregation-role="dimension" MaxLength="10"/>
<Property sap:label="Customer" Name="CustomerID" Type="Edm.String" sap:quickinfo="EPM: Customer ID" sap:display-format="UpperCase" sap:aggregation-role="dimension" MaxLength="10"/>
Dimension <Property sap:label="Company Name" Name="CompanyName" Type="Edm.String" sap:quickinfo="EPM: Company Name" MaxLength="80" sap:attribute-for="CustomerID"/>
<Property sap:label="Product ID" Name="Product" Type="Edm.String" sap:quickinfo="EPM: Product ID" sap:text="to_Product/Product_Text" sap:display-format="UpperCase" sap:aggregation-role="dimension" MaxLength="10" sap:value-list="standard" />
<Property sap:label="Product Category" Name="ProductCategory" Type="Edm.String" sap:quickinfo="EPM: Product Category" sap:attribute-for="Product" MaxLength="40" />
Attribute <Property sap:label="ISO Currency Code" sap:semantics="currency-code" Name="CurrencyCode" Type="Edm.String" sap:quickinfo="EPM: Currency Code" sap:aggregation-role="dimension" MaxLength="5"/>
<Property sap:semantics="currency-code" Name="TargetCurrency" Type="Edm.String" sap:aggregation-role="dimension" MaxLength="5"/>
<Property Name="ConvertedGrossAmount" sap:filterable="false" Type="Edm.Decimal" sap:aggregation-role="measure" Precision="16" sap:unit="TargetCurrency" Scale="3"/>
<Property sap:label="Total Gross Amount" Name="GrossAmount" sap:filterable="false" Type="Edm.Decimal" sap:quickinfo="EPM: Total Gross Amount" sap:aggregation-role="measure" Precision="16" sap:unit="CurrencyCode" Scale="3"/>
Measure <Property sap:label="Total Net Amount" Name="NetAmount" sap:filterable="false" Type="Edm.Decimal" sap:quickinfo="EPM: Total Net Amount" sap:aggregation-role="measure" Precision="16" sap:unit="CurrencyCode" Scale="3"/>
<Property sap:label="Total Tax Amount" Name="TaxAmount" sap:filterable="false" Type="Edm.Decimal" sap:quickinfo="EPM: Total Tax Amount" sap:aggregation-role="measure" Precision="16" sap:unit="CurrencyCode" Scale="3"/>
<Property sap:semantics="count" Name="AllItems" sap:filterable="false" Type="Edm.Int32" sap:aggregation-role="measure"/>
<Property Name="DistinctProducts" sap:filterable="false" Type="Edm.Int32" sap:aggregation-role="measure"/>
<NavigationProperty Name="to_Product" ToRole="ToRole_assoc_AAF8BC2D7DDE36AF83ED42DD9AA9D33" FromRole="FromRole_assoc_AAF8BC2D7DDE36AF83ED42DD9AA9D33" Relationship="SERVICE_NAME.assoc_AAF8BC2D7DDE36AF83ED42DD9AA9D33"/>
<NavigationProperty Name="to_SalesOrder" ToRole="ToRole_assoc_FCD485C2E8FB98D874E8452CC7AB576A" FromRole="FromRole_assoc_FCD485C2E8FB98D874E8452CC7AB576A" Relationship="SERVICE_NAME.assoc_FCD485C2E8FB98D874E8452CC7AB576A"/>
</EntityType>

```

Metadata of an Aggregated Entity

Aggregated OData Entities

As soon as one element in the CDS view is annotated with the aggregation annotation `@Aggregation.Default: #<AGGR_FUNCTION>`, the OData entity is annotated with `sap:semantics="aggregate"`. Hence, the OData entity is identified as an aggregated entity.

In the example of the screenshot above, this OData annotation can be found in the first line of the extract of the metadata.

Measures

The aggregated entity is characterized by measures and dimensions. Measures are those properties that are annotated with the annotation relevant for aggregating data in CDS. Measures are given the OData annotation `sap:aggregation-role="measure"`.

In the example of the screenshot above, there are six properties which are marked as measures: ConvertedGrossAmount, GrossAmount, NetAmount, TaxAmount, AllItems, and DistinctProducts.

Dimensions

Dimensions are all properties that are neither marked as measures nor as attributes. Dimensions are given the OData annotation `sap:aggregation-role="dimension"`.

In the example of the screenshot above, there are six OData properties which are marked as dimensions: SalesOrderID, ItemPosition, Customer ID, Product, CurrencyCode, and TargetCurrency.

Each dimension can have a maximum of one text property. A text property is an element that is defined as a text element in CDS, as described in [Defining Text Elements \[page 271\]](#). Dimensions with a text property are annotated by OData with `sap:text="<TEXT_PROPERTY>"`.

In the example of the screenshot above, the dimension Product has a text property.

Attributes

Dimensions can have attributes. Attributes are elements that are annotated with the annotation `@Consumption.groupWithElement: 'ElementRef'`. The attributes are always considered as an addition to their corresponding dimension. Attributes are annotated by OData with `sap:attribute-for="<dimension>"`.

In the example of the screenshot above, the dimension CustomerID has the attribute CompanyName and the dimension Product has the attribute ProductCategory.

Generated ID for Aggregated OData Entities

An aggregated OData entity is given an additional property for a generated ID (`<Property Name="ID"/>`). The generated ID for the aggregate entity uniquely identifies each record so that every entity request for a given ID always returns the same values.

In the example of the screenshot above, the property for the generated ID can be found as the first property on the list.

The ID is also generated for every group record when it is requested for the first time. Following from this, you can use this ID in a further request.

Example

This behavior is exemplified by a request on the entity that supplies the metadata above. It retrieves sales order items. The following request selects the generated ID (`ID`), the dimension `Product`, and some aggregated measures related to the selected dimension.

```
....sap/opu/odata/SAP/<service_name>/ZDEMO_C_SOI_ANLY?  
$select=ID,Product,GrossAmount,NetAmount,TaxAmount,AllProducts
```

This request retrieves the following result:

```
<feed xml:version="1.0"?>  
- <feed xml:base="https://HOST/sap/opu/odata/SAP/SERVICE_NAME/" xmlns:s="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"  
  xmlns="http://www.w3.org/2005/Atom">  
  <id>https://HOST/sap/opu/odata/SAP/SERVICE_NAME/ZDEMO_C_SOI_ANLY</id>  
  <title type="text">ZDEMO_C_SOI_ANLY</title>  
  <updated>2018-04-03T13:27:26Z</updated>  
  - <author>  
    <name/>  
    <link title="ZDEMO_C_SOI_ANLY" rel="self" href="ZDEMO_C_SOI_ANLY"/>  
  </author>  
  <entry>  
    <id>https://HOST/sap/opu/odata/SAP/SERVICE_NAME/ZDEMO_C_SOI_ANLY('4~HT%26c1002.6~USD')</id>  
    <title type="text">ZDEMO_C_SOI_ANLY('4~HT%26c1002.6~USD')</title>  
    <updated>2018-04-03T13:27:26Z</updated>  
    <category scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" term="SERVICE_NAME.ZDEMO_C_SOI_ANLYType"/>  
    <link title="ZDEMO_C_SOI_ANLYType" rel="self" href="ZDEMO_C_SOI_ANLY('4~HT%26c1002.6~USD')/"/>  
    - <content type="application/xml">  
      <m:properties xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:s="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">  
        <d:ID>4~HT%26c1002.6~USD</d:ID>  
        <d:Product>HT1002</d:Product>  
        <d:GrossAmount>3529.01</d:GrossAmount>  
        <d:NetAmount>1570.00</d:NetAmount>  
        <d:TaxAmount>596.60</d:TaxAmount>  
        <d>AllProducts>9</d>AllProducts>  
      </m:properties>  
    </content>  
  </entry>  
  <entry>  
    <id>https://HOST/sap/opu/odata/SAP/SERVICE_NAME/ZDEMO_C_SOI_ANLY('4~HT%26c1007.6~USD')</id>  
    <title type="text">ZDEMO_C_SOI_ANLY('4~HT%26c1007.6~USD')</title>  
    <updated>2018-04-03T13:27:26Z</updated>  
    <category scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" term="SERVICE_NAME.ZDEMO_C_SOI_ANLYType"/>  
    <link title="ZDEMO_C_SOI_ANLYType" rel="self" href="ZDEMO_C_SOI_ANLY('4~HT%26c1007.6~USD')/"/>  
    - <content type="application/xml">  
      <m:properties xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:s="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">  
        <d:ID>4~HT%26c1007.6~USD</d:ID>  
        <d:Product>HT1007</d:Product>  
        <d:GrossAmount>934.00</d:GrossAmount>  
        <d:NetAmount>598.00</d:NetAmount>  
        <d:TaxAmount>170.43</d:TaxAmount>  
        <d>AllProducts>8</d>AllProducts>  
      </m:properties>  
    </content>  
  </entry>  
</feed>
```

Aggregate Data of Group Record with Generated ID for Group Record

Each group record is given its own generated ID which retrieves the same results when requested again. Based on this group ID, you can also execute other requests, as in `/sap/opu/odata/SAP/<service_name>/ZDEMO_C_SOI_ANLY('4~HT%26c1002.6~USD')?$select=AllProducts`, which will only retrieve the count of this group, which is 9.

Requesting Data from an Aggregated Entity

Results of requesting data from aggregated entities depend on the elements you select in your OData request. Grouping and aggregation are both driven by the elements you request with the parameter `$select` in entity set queries. The result of a query consists of aggregated entities with distinct values for the requested dimension properties and requested measures are aggregated using the aggregate function with which the measure elements are annotated in CDS.

If an attribute is requested, the result is grouped by its corresponding dimension and within that group it is grouped by the attribute itself.

i Note

If you use a SAP Fiori app, the `$select` statement of the OData request directly depends on the columns you select in the list reporting app. The following descriptions of requesting data with OData can also be managed by selecting the respective columns in your Fiori App.

Requests that Contain All Primary Key Elements

If all primary key elements of the requested CDS entity are included in the query option `$select`, no grouping is possible since every record is unique. Hence, an OData request with all key elements behaves just like a non-aggregated entity. Consequently, no aggregate data is calculated.

Requests that Do not Contain All Primary Key Elements

If not all key elements are included in the OData request, the requested dimensions are grouped and the requested measures are aggregated according to their annotated aggregate function. For every distinct combination of dimension values (after evaluating `$filter`), the result includes an aggregated entity with aggregated values for the requested measures.

i Note

Each group record is given its own generated ID, which can be reused in requests to retrieve the same results.

Only if no dimension and no attribute are requested does the result show the aggregate data of measures for the whole entity set.

You can still execute other query options, such as `$filter` or `$orderby` on grouped requests. The filtering is executed before the grouping, so that the groups are created from the available records after the filtering. The ordering is executed after the grouping, which means the records within a group are ordered according to the query option.

i Note

You can filter for properties that you do not select, but you cannot order by properties that are not included in the \$select due to the order of executing query options mentioned above.

Navigating and Expanding from a Group Record

You can navigate directly from a group record to one of the properties that are included in the group, for example properties that you have selected previously.

• Example

Assume you have an aggregated entity of sales order items and you group them by product. From this group record, you can navigate directly to the associated entity of products if the association exists in CDS.

The same holds true for the query option \$expand. Only properties that are selected can be expanded.

i Note

The target entity of the navigation or the expand is also given an aggregated ID if the underlying data model is also aggregated.

Parameters and Aggregated Entities

If your CDS view contains visible parameters (not annotated with @Consumption.hidden: true), the OData entity is generated with the same properties for each parameter as in non-aggregated entities. However, the naming pattern differs. Check [Analytical Scenarios \[page 359\]](#) for more information.

Related Information

[Using Aggregate Data in SAP Fiori Apps \[page 263\]](#)

[Annotating Aggregate Functions in CDS \[page 264\]](#)

8.7 Defining Text Elements

This section describes how to determine and provide texts for a CDS view element within the context of the SAP Fiori programming model.

Contents

- [Language-independent Text Elements \[page 272\]](#)
- [Getting Text Through Text Associations \[page 273\]](#)

8.7.1 Language-independent Text Elements

Within the context of CDS views, the text elements establish the link between identifier elements (code values) of the view and its descriptive language-independent texts. For example, you can define a link between a company code and the (descriptive) company name, or between currency code and the currency name. These kinds of descriptive texts are **language-independent**.

Relevant Annotation

Annotation	Effect
<code>@ObjectModel.text.element[]</code>	Establishes the link between the annotated element (that defines an identifier element) and its descriptive language-independent texts

i Note

The usage of this annotation excludes the usage of `@ObjectModel.text.association`.

More on this: [ObjectModel Annotations \[page 428\]](#)

i Note

SADL Runtime Behavior: In OData exposure scenarios, the **first text element** listed in the annotation array will be handled as a descriptive text of the annotated field.

Example

In the listing below, the CDS view `I_Plant` defines the fields `PlantName` and `PlantDescription` that both serve as language-independent descriptions for the view field `Plant`.

```
...
define view I_Plant as ...
{
    @ObjectModel.text.element: ['PlantName', 'PlantDescription']
    key Plant,
    PlantName,
    PlantDescription,
    ...
}
```

⚠ Caution

Obsolete Use: You may be aware that the `Consumption.labelElement` annotation is used in the same context. This annotation also enables consumers to identify through which elements the descriptive texts for the identifier elements can be retrieved. In the example below, the label of currency is set through the `CurrencyText` field:

```
@Consumption.labelElement: 'CurrencyText' --Obsolete use!
currency_code as CurrencyCode,
currency as CurrencyText
```

However, we recommend using `ObjectModel.text.element[]` instead of the `Consumption.labelXElement` annotation:

```
@ObjectModel.text.element: ['CurrencyText']      --Recommended use!
currency_code as CurrencyCode,
currency as CurrencyText
```

8.7.2 Getting Text Through Text Associations

Using the CDS text association, you can define the relationship between an element (field) and its corresponding texts or descriptions. The texts are usually language-dependent and are displayed in the end user's language. If you annotate an element with a text association (as described below), the associated text or description field is added as a field to the referencing entity. At runtime, this field is read from the database and filtered by the logon language of the OData consumer automatically. It is not necessary to use session properties or view parameters in your CDS view.

Steps

To retrieve texts by direct use of text associations, proceed as follows:

1. **Create a data definition with a CDS view that serves as text provider**

The following set of CDS annotations is relevant when defining text views:

Relevant Annotations:

The following annotation is required at the **view level**:

Annotation and Value	Effect
<code>@ObjectModel.dataCategory : #TEXT</code>	Indicates that the annotated CDS view represents texts (defines a text provider view)

i Note

Usually one key element is used to specify the language!

The following annotations are required at the view **element level** to annotate the language key element and the text elements from the view's element list:

Annotation and Values	Effect
<code>@Semantics.language: true</code>	The annotated element identifies the language field.

Annotation and Values	Effect
<code>@Semantics.text: true</code>	<p>Identifies view elements as text elements (fields pointing to a textual description)</p> <p>i Note</p> <p>In general, you can annotate more than one view field as a text field. However, only the first annotated field will be considered in the text consumer view for OData exposure.</p>

More on this:

- [ObjectModel Annotations \[page 428\]](#)
- [Semantics Annotations \[page 445\]](#)

Example

The listing below implements a sample text view. Its entity is named `I_UnitOfMeasureText` and uses database table `T006A` as its data source. In this example, the two fields `MSEHT` (UnitName) and `MSEHL` (UnitLongName) are annotated as text fields with `@Semantics.text: true`. However, note that - in OData exposure scenarios - the first text element listed in the annotation array will be handled as a descriptive text of the annotated field in the consumer view.

```

...
@ObjectModel.dataCategory: #TEXT

@endUserText.label: 'Unit of Measure Text'

define view I_UnitOfMeasureText
  as select from T006A          as TextProvider
{
  key TextProvider.msehi        as UnitOfMeasure,
  @Semantics.language: true    -- identifies the language
  key TextProvider.spras       as Language,
  @Semantics.text: true        -- identifies the first text field for
names
  TextProvider.mseht           as UnitName
  @Semantics.text: true        -- identifies the second text field
  TextProvider.msehl           as UnitLongName
}

```

2. Create a text association from your consumer CDS view to the text provider view

The following CDS annotation is relevant when creating text associations:

Annotation and Values	Effect
<code>@ObjectModel.text.association: '<_AssocToTextProvider>'</code>	Name of an association with a text view that provides descriptive texts for the annotated element. In other words: the annotation indicates that the description for the annotated element is available using the text association <code><_AssocToTextProvider></code> .

More on this:

- [ObjectModel Annotations \[page 428\]](#)

Example

The view `I_UnitOfMeasure` in the listing below serves as a consumer for the text provider that has been implemented in the previous step. For this purpose, the association `_Text` with the target view `UnitOfMeasureText` is defined. To indicate the field for which a text should be made available through the association `_Text`, the annotation `@ObjectModel.text.association` is added. Note that only the first text element (`UnitName`) from the text provider, which is annotated with `@Semantics.text: true`, will be considered for OData exposure.

```
...
define view I_UnitOfMeasure as select from T006 as UnitOfMeasure
    association [0..*] to I_UnitOfMeasureText as _Text
        on $projection.UnitOfMeasure = _Text.UnitOfMeasure
{
    ...
    @ObjectModel.text.association: '_Text'
    key msehi as UnitOfMeasure,
    _Text
}
```

⚠ Caution

If you use the [Referenced Data Source \(RDS\)](#) option for OData service creation, it is important that you always regenerate your gateway project in transaction `SEGW` after you add, change, or remove text associations in the consumer views contained in the project. The same is true if you change the order of fields in the text provider view - since such changes trigger the data type change of the entity set in gateway project.

8.8 Providing Value Help

The implementation of a value help in CDS enables the end user to choose values from a predefined list for input fields on a user interface.

Why and When to Use Value Help

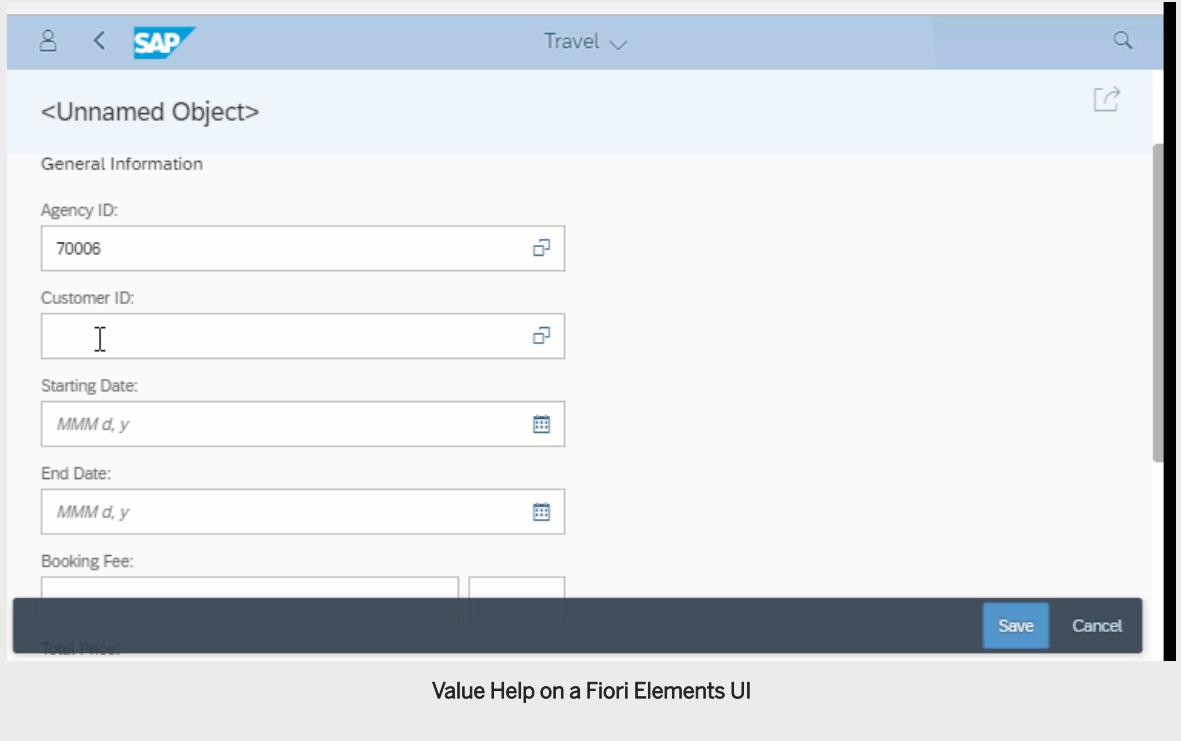
You can define value helps in the CDS data layer for any input field on the UI (for example, selection fields, parameters, or writable fields in transactional scenarios). A value help is useful when the values for the input field have to be taken from a predefined set of values. When you call the value help, it displays all the available values from the value help provider. It appears on the UI when you choose the button  in the input field or press the `F4` key. The end user can filter by related information to identify the correct value. This makes it easier to find the desired value, especially if the value itself contains little or no identifying information, for example, an ID number.

❖ Example

To help the end user to enter the right customer ID to create a new booking, the application developer defines a value help that enables the user to enter the name or any other element from the value help

provider to help find the correct number. The value help provider view in this case is a CDS view that manages customer information. As shown below, the end user is searching for a particular customer ID. The value help offers to filter by the customer last name, so that the end user can choose from the available entries. The value of the customer ID field is then transferred to the respective input field.

Expand the following figure to view the procedure of calling the value help on a Fiori Elements UI.



How are Value Helps Implemented?

Value helps are defined in CDS with an annotation on the element or parameter for which the value help dialog is to appear on the UI. The annotation `@Consumption.valueHelpDefinition` allows you to reference the value help provider without implementing an association. You simply assign a CDS entity as the value help provider and provide an element for the mapping in the annotation. All fields of the value help provider are displayed on the UI. When you choose one of the entries of the value help provider, the value of the referenced element is transferred to the input field.

For the default implementation of the value help, you can use any CDS entity that contains the desired values of the element for the input field. You do not need to explicitly define a CDS entity as the value help provider. However, the value help provider must be exposed for the OData service to make use of the value help.

i Note

Any annotation that is maintained in the value help provider is evaluated. This means that associated entities, value helps, and text associations of the value help provider view are also displayed and can be used in the value help. This means that you can have a value help in the value help.

The following value help options are available within the programming model:

[Simple Value Help \[page 277\]](#)

[Multiple Value Helps on One Element \[page 281\]](#)

[Value Help with Additional Binding \[page 282\]](#)

Related Information

[Consumption Annotations \[page 390\]](#)

8.8.1 Simple Value Help

A simple value help is convenient if you want to display values from the value help provider for an input field.

Context

You want to provide a value help for an input field on the UI.

The following steps implement a value help for a customer ID field, using a booking CDS view as an example.

Procedure

1. Create a data definition for a CDS view that serves as a value help provider. It must contain a field with the available values for the input field in the source view.

Example

The value help provider view contains the customer ID and fields to identify the customer ID, such as the customer's name or address. The end user can then filter by these fields to find the right customer ID.

```
define view /DMO/I_Customer_VH as select from /dmo/customer
{
  key customer_id,
  first_name,
  last_name,
  title,
  street,
  postal_code,
  city,
  country_code,
  phone_number,
  email_address
}
```

2. In your CDS source view, add the following annotation to the element for which you want to provide the value help on the UI.

```
@Consumption.valueHelpDefinition: [{ entity: { name: 'entityRef' ,
```

```
'elementRef'  }      } ]
```

element:

The annotation defines the binding for the value help to the value help providing entity. You must specify the entity name and the element providing the possible values for the annotated element.

Example

The following code example shows how an annotation is used on the element `CustomerID` in `/DMO/I_Booking`. It references the value help provider view (`/DMO/I_CUSTOMER_VH`) for the customer ID and the element providing the possible values (`customer_id`) in the value help provider view.

```
define view /DMO/I_Booking as select from /dmo/booking
{...
    @Consumption.valueHelpDefinition: [{entity: { name: '/DMO/
I_CUSTOMER_VH',
    'customer_id'  }}]
    customer_id as CustomerID,
... }
```

element:

Results

Choosing **F4** in the selection field opens a search mask and the end user can filter by any field in the value help provider view. Selecting an entry transfers the value of the element that is referenced in the annotation to the annotated element in the source view.

The metadata of the OData service displays the value help implementation for the following properties:

- The property in the CDS source view for which a value help is implemented (`sap:value-list="standard"`)
- The value help provider entity type (`sap:value-list="true"`)
- The value help provider entity is marked as `Target` in the `Annotations` property. The value list enumerates every property in the value help provider that is exposed for the value help (`Annotation Term="Common.ValueList"`).
- The element that is defined in the mapping condition is marked as an inbound and outbound parameter `Record Type="Common.ValueListParameterInOut"`.

Expand to see the extracts of the metadata document of a service exposing a booking CDS view and the value help provider for the element `CustomerID`.

```
- <EntityType sap:content-version="1" sap:label="Consumer CDS View - Booking" Name="BookingType">
+ <Key>
<Property sap:label="Travel ID" Name="TravelID" sap:quickinfo="Flight Reference Scenario: Travel ID" sap:display-format="NonNegative" MaxLength="8" Nullable="false"
  Type="Edm.String"/>
<Property sap:label="Booking Number" Name="BookingID" sap:quickinfo="Flight Reference Scenario: Booking ID" sap:display-format="NonNegative" MaxLength="4"
  Nullable="false" Type="Edm.String"/>
<Property sap:label="Booking Date" Name="BookingDate" sap:quickinfo="Flight Reference Scenario: Booking Date" sap:display-format="Date" Type="Edm.DateTime"
  Precision="0"/>
<Property sap:label="Customer ID" Name="CustomerID" sap:quickinfo="Flight Reference Scenario: Customer ID" sap:display-format="NonNegative" MaxLength="6"
  Type="Edm.String" sap:value-list="standard"/>
<Property sap:label="Airline ID" Name="CarrierID" sap:quickinfo="Flight Reference Scenario: Carrier ID" sap:display-format="UpperCase" MaxLength="3"
  Type="Edm.String"/>
<Property sap:label="Flight Number" Name="ConnectionID" sap:quickinfo="Flight Reference Scenario: Connection ID" sap:display-format="NonNegative" MaxLength="4"
  Type="Edm.String"/>
<Property sap:label="Flight Date" Name="FightDate" sap:quickinfo="Flight Reference Scenario: Flight Date" sap:display-format="Date" Type="Edm.DateTime" Precision="0"/>
<Property sap:label="Flight Price" Name="FlightPrice" sap:quickinfo="Flight Reference Scenario: Flight Price" Type="Edm.Decimal" Precision="17" sap:unit="CurrencyCode"
  Scale="3"/>
<Property sap:label="Currency Code" Name="CurrencyCode" sap:quickinfo="Flight Reference Scenario: Currency Code" MaxLength="5" Type="Edm.String"
  sap:semantics="currency-code"/>
</EntityType>
- <EntityType sap:content-version="1" sap:label="CDS View Customer - Value Help Provider" Name="CustomerType" sap:value-list="true">
+ <Key>
<Property sap:label="Customer ID" Name="customer_id" sap:quickinfo="Flight Reference Scenario: Customer ID" sap:display-format="NonNegative" MaxLength="6"
  Nullable="false" Type="Edm.String"/>
<Property sap:label="First Name" Name="first_name" sap:quickinfo="Flight Reference Scenario: First Name" MaxLength="40" Type="Edm.String"/>
<Property sap:label="Last Name" Name="last_name" sap:quickinfo="Flight Reference Scenario: Last Name" MaxLength="40" Type="Edm.String"/>
```

```

- <Annotations xmlns="http://docs.oasis-open.org/odata/ns edm" Target="cds_xdmoBooking_vh.BookingType/CustomerID">
  - <Annotation Term="Common.ValueList">
    - <Record>
      <PropertyValue String="CDS View Customer - Value Help Provider" Property="Label"/>
      <PropertyValue String="Customer" Property="CollectionPath"/>
      <PropertyValue Property="SearchSupported" Bool="true"/>
    - <PropertyValue Property="Parameters">
      - <Collection>
        - <Record Type="Common.ValueListParameterInOut">
          <PropertyValue Property="LocalDataProperty" PropertyPath="CustomerID"/>
          <PropertyValue String="customer_id" Property="ValueListProperty"/>
        </Record>
        - <Record Type="Common.ValueListParameterDisplayOnly">
          <PropertyValue String="first_name" Property="ValueListProperty"/>
        </Record>
        - <Record Type="Common.ValueListParameterDisplayOnly">
          <PropertyValue String="last_name" Property="ValueListProperty"/>
        </Record>
        + <Record Type="Common.ValueListParameterDisplayOnly">
        - <Record Type="Common.ValueListParameterDisplayOnly">
          <PropertyValue String="street" Property="ValueListProperty"/>
        </Record>
        + <Record Type="Common.ValueListParameterDisplayOnly">
        - <Record Type="Common.ValueListParameterDisplayOnly">
          <PropertyValue String="city" Property="ValueListProperty"/>
        </Record>
        - <Record Type="Common.ValueListParameterDisplayOnly">
          <PropertyValue String="country_code" Property="ValueListProperty"/>
        </Record>
        - <Record Type="Common.ValueListParameterDisplayOnly">
          <PropertyValue String="phone_number" Property="ValueListProperty"/>
        </Record>
        + <Record Type="Common.ValueListParameterDisplayOnly">
          <PropertyValue String="email_address" Property="ValueListProperty"/>
        </Record>
      </Collection>
    </PropertyValue>
  </Record>
</Annotation>

```

This value help provider is also search supported.

Metadata Document

Postrequisites

If the source view is exposed for an OData service, you need to include the value help provider view in the service to be able to retrieve the values from the value help.

Other Value Help Options

Other Capabilities for the Value Help Provider

Any annotation that is maintained in the value help provider is evaluated. This means that the following capabilities are possible for the value help:

- Associations:** If the target of the association is included in the OData service, the elements of entities associated with the value help provider are also displayed as additional input fields.
- Search Capabilities:** Including search capabilities in your value help provider enables the end user to search for any detail in an input field.
- Value Helps:** The value help can itself contain value helps.
- Text:** Text that is provided for the value help provider is also displayed.

Value Help as Dropdown List

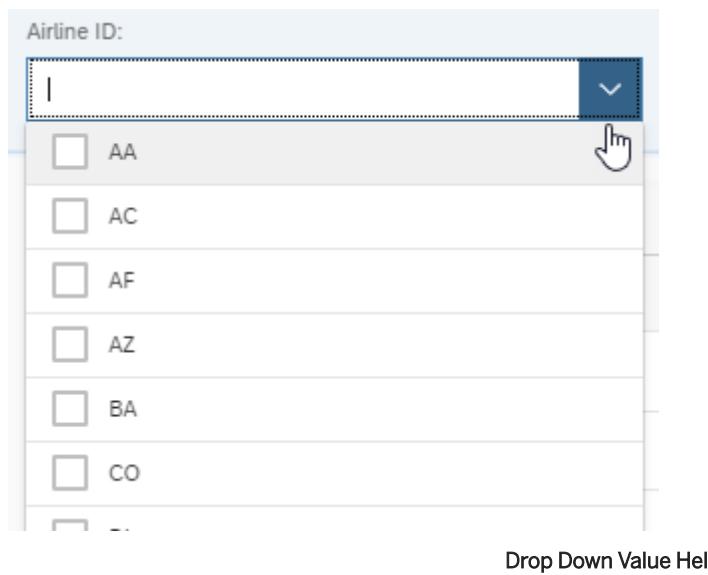
If you want to show the possible values for the input field as a dropdown list only instead of as a search mask with all other elements of the value help provider, annotate the value help provider with `@ObjectModel : { resultSet.sizeCategory: #XS }` on the view level.

For dropdown value lists, the OData \$metadata document includes the annotation `sap:value-list="fixed-values"` instead of `standard` on the property in the source view for which the value help is implemented.

```
<Property Name="CarrierID" sap:label="" sap:display-format="UpperCase" MaxLength="3" Type="Edm.String" sap:value-list="fixed-values"/>
```

OData Metadata for Dropdown Value Helps

The UI displays the possible values for the input field in a dropdown list. The following image shows a value help list for the carrier ID field in the booking scenario.



Drop Down Value Help

Value Help Via Association

If the value help provider is already associated with the source entity, you can use the association `@Consumption.valueHelpDefinition.association: '_Assoc'` and reference the association instead of the entity. The binding condition is then already given in the on-condition of the association.

• Example

The following code example shows how the annotation is used on the element `CustomerID` in `/DMO/I_Booking`. It references the association to the value help provider view (`_Customer`).

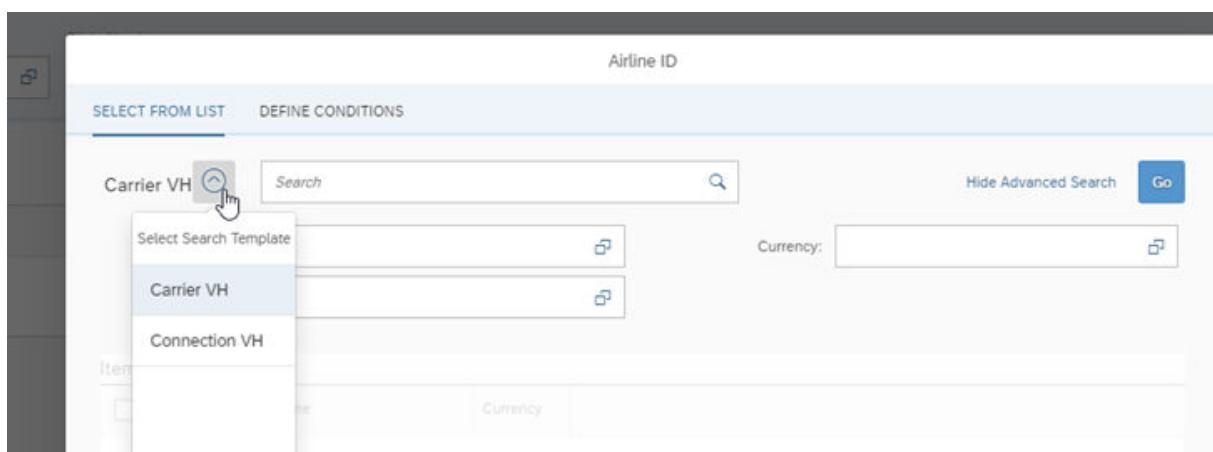
```
define view /DMO/I_Booking as select from /dmo/booking
  association [0..1] to /DMO/I_Customer_VH as _Customer on
  $projection.CustomerID = _Customer.customer_id
  ...
    @Consumption.valueHelpDefinition.association: '_Customer'
    customer_id as CustomerID,
  ...
  _Customer
}
```

8.8.1.1 Multiple Value Helps on One Element

Context

It is possible to provide more than one value help on one element. The end user selects which value help to use to find the correct value.

The following image displays the value helps for the carrier ID element in the booking CDS view. One value help provider is defined by a carrier CDS view and one by a connection CDS view that also contains the carrier ID field.



Procedure

1. To implement two value helps on one element, proceed as described in [Simple Value Help \[page 277\]](#) and add another entity as a value help provider.

```
define view /DMO/I_Booking as select from /dmo/booking
{...
    @Consumption.valueHelpDefinition: [{ entity: { name: '/DMO/
I_Carrier_VH' ,
                                                 element:
'carrier_id' } }
                                         { entity: { name: '/DMO/
I_Connection_VH',
                                                 element:
'carrier_id'} ]
    carrier_id as CarrierID,
... }
```

You can define more than two value helps on one element.

2. Assign labels to the different value helps to differentiate them on the UI.

```
define view /DMO/I_Booking as select from /dmo/booking
{...
    @Consumption.valueHelpDefinition: [{ entity: { name: '/DMO/
I_Carrier_VH' ,
```

```

    'carrier_id' },
                element:
                    label: 'Carrier VH' },
                    { entity: { name: '/DMO/
I_Connection_VH',
                element:
                    'carrier_id' },
                    label: 'Connection VH']
            carrier_id as CarrierID,
... }

```

3. Equip one value help with a qualifier.

```

define view /DMO/I_Booking as select from /dmo/booking
{...           @Consumption.valueHelpDefinition: [{ entity: { name: '/DMO/
I_Carrier_R' ,
                element:
                    'carrier_id' },
                    label: 'Carrier VH' },
                    { entity: { name: '/DMO/
I_Connection_VH',
                element:
                    'carrier_id' },
                    label: 'Connection VH',
                    qualifier: 'Secondary Value
Help'}]
            carrier_id as CarrierID,
... }

```

If you have more than one value help, it is important that all except one are equipped with a qualifier. The default value help is the one without a qualifier. The qualifier marks the value helps that are less important. If all value helps are annotated with the qualifier argument, then none of them are displayed as there is no default.

Results

Choosing **[F4]** in the input field opens a search mask with the fields of the default value help. The default value help is the one without a qualifier. The end user can select which value help to use from a dropdown menu.

8.8.2 Value Help with Additional Binding

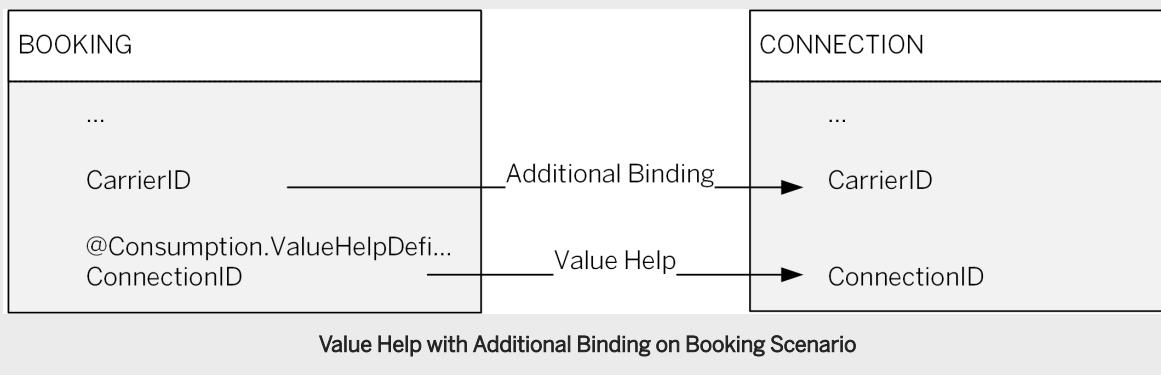
A preset filter condition can be established by an additional binding for the value help.

Context

You use an additional binding to define more than one binding condition between the source view and the value help provider. The value help is then automatically filtered by the additional binding. It proposes only entries that match the additional binding. This additional binding can either be another element or a parameter. These need to be present in the source view and in the value help provider view. When an entry is selected in the value help provider, the values of both binding elements are transferred to the input fields of the source CDS view.

• Example

In our booking scenario, we can apply the value help with an additional binding on the field `ConnectionID`. The value help provider is a view that manages connections. This value help provider contains not only the field for the connection IDs, but also a field for carrier IDs, which is also in the consumer view. We can establish a second binding condition so that the value help provider only displays connections with the prechosen carrier ID.



Procedure

1. Create a data definition for a CDS view that serves as a value help provider. It must contain a field with the available values for the input field in the source view. In addition, it must contain the field for which the additional binding is established.

The value help provider view contains the connection ID and the carrier ID, for which a mapping condition is defined.

```
define view /DMO/I_Connection_VH as select from /dmo/connection
{
  key carrier_id,
  key connection_id,
  airport_from_id,
  airport_to_id,
  departure_time,
  arrival_time,
  distance,
  distance_unit
}
```

2. In your CDS source view, add the following annotation to the element for which you want to provide the value help on the UI.

```
      @Consumption.valueHelpDefinition: [{} entity: {name:
'entityRef', element: 'elementRef',
          additionalBinding: [{} element: 'elementRef',
localElement: 'elementRef' {}]}]
```

The additional binding defines a second condition for the value help on the same target value help provider entity for filtering the value help result list and/or returning values from the selected value help record. The additional binding can be defined for an additional element or parameter.

The annotation requires an array as value.

The following code example shows the usage of the annotation on the element `ConnectionID` in `/DMO/I_Booking`. It references the value help provider view (`/DMO/I_Connection_VH`) and the element providing the possible values (`connection_id`) in the value help provider view, as well as the matching condition on the elements `CarrierID` and `carrier_id` in the consumer view and the value help provider view, respectively.

```
define view /DMO/I_Booking_VH as select from /dmo/booking
{...
    @Consumption.valueHelpDefinition: [{ entity:
        {name: '/DMO/I_Connection_VH', element: 'connection_id' },
        additionalBinding: [{ localElement: 'CarrierID', element:
            'carrier_id' }]}
    ...
connection_id as ConnectionID,
... }
```

Results

Choosing `F4` in the selection field opens a search mask and the end user can display all available entries in the value help provider or filter by a field, for example, the destination airport. If the carrier ID is already filled in the consumer view, the value help provider is prefetched by that value. Selecting an entry in the value help transfers the connection ID as well the carrier ID to the CDS consumer view.

The metadata of the OData service displays the value help implementation on the following properties:

- The property in the CDS source view for which a value help is implemented (`sap:value-list="standard"`)
- The value help provider entity type (`sap:value-list="true"`)
- The value help provider entity is marked as `Target` in the `Annotations` property. The value list enumerates every property in the value help provider that is exposed for the value help (`Annotation Term="Common.ValueList"`).
- The elements that are defined in the mapping conditions are marked as inbound and outbound parameters `Record Type="Common.ValueListParameterInOut"`.

Next Steps

If the consumer view is exposed for an OData service, you need to include the value help provider view in the service to be able to retrieve the value help.

8.9 Adding Field Labels and Descriptions

End-user texts, such as field labels or field descriptions, are taken from ABAP Dictionary data elements to which the corresponding element is bound - unless you redefine the texts using CDS annotations. Unlike technical element names, the header texts, field labels and descriptions are **language-dependent**. For example, the field 'SalesOrder' would have a language-dependent label 'Sales Order Header'.

Such texts must be translated. Therefore, the CDS development infrastructure is able to extract them from the source code and transfer the extracted texts to the actual translation infrastructure of the corresponding delivery package.

Relevant Annotations

Annotation	Effect
@EndUserText.label: '<text>'	This annotation is used to define translatable semantic short texts (with maximum 60 characters) for an element of the CDS view (label text).
@EndUserText.quickInfo: '<text>'	The annotation defines a human-readable <text> that provides additional information compared to the label text. The quick info is used for accessibility hints or the mouse over function.

→ Remember

The <text> specified in the source code should consist of text in the original language of the CDS source code and will be translated into the required languages.

Example

The listing below demonstrates the usage of @EndUserText annotation at the view and element (field) level:

```
...
@EndUserText.label: 'List with Sales Orders' -- Annotation at the view level
DEFINE VIEW SalesOrderHeader as ... {
  ...
  -- Annotation at the field level
  @EndUserText: { label: 'Sales Order Header',
    quickinfo: 'Sales Order Header that provides data relevant for all items' }
  SalesOrder as Header;
  ...
}
```

→ Tip

Press **F1** in the CDS source code editor for extended help content on @EndUserText annotation!

8.10 Mapping CDS View-Elements onto Table Fields

Motivation

When creating data models for business objects, database tables are often used as a data source for the CDS views that define the business object's data model.

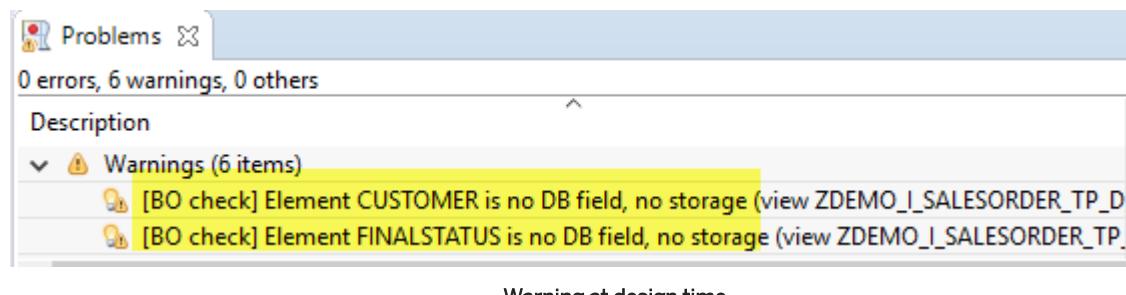
In such cases, it may also happen that existing legacy tables (which originate from older releases) are to be reused in the definition of the new data model. Tables like these may sometimes contain quite short or cryptic field names. On the other hand, when describing the data model using CDS, you should try to use semantically descriptive names.

Example

In the following example, the table field names `bpartner` and `ostatus` are mapped to the alias names `Customer` and `FinalStatus`.

```
...
@ObjectModel.foreignKey.association: '_BusinessPartner'
bpartner           as Customer,
...
@ObjectModel.foreignKey.association: '_Status'
ostatus            as FinalStatus,
...
```

Whenever the table field names and the corresponding element names in the CDS view do not match (except for camel-case differences), this causes problems when activating the CDS view. As shown in the following figure, these problems are only noted as a warning at design time.



At runtime, however, a mapping problem like this would cause the relevant fields to be unable to write records to the active business object's instances.

Sales Order ID	Customer	Status
700000004 Draft	SAP (100000000)	In Progress (I)
700000013		
700000014		
700000012		

Relevant fields for active instances are empty at runtime

Solution

One such problem can be solved in quite a simple way by using a classic database view for mappings between table fields and the element names of the CDS view and by using this database view for writing/storing the active data.

Steps in Detail

- Start the transaction **SE11** and create a classic database view for mapping table fields onto the corresponding elements of the CDS view.

The screenshot shows the SAP Dictionary: Change View interface. The database view is named **ZDEMO_SOH_V**. The short description is **Mapping Sales Order Fields**. The view fields tab is selected, showing the mapping between view fields and table fields:

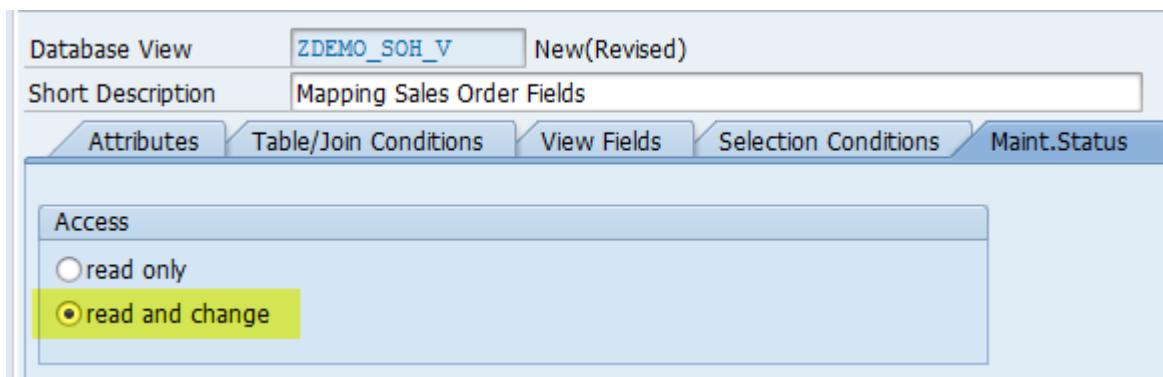
View field	Table	Field	Key	Data elem.
CLIENT	ZDEMO_SOH	CLIENT	<input checked="" type="checkbox"/>	MANDT
SALESORDERUUID	ZDEMO_SOH	SALESORDERUUID	<input checked="" type="checkbox"/>	SNWD_NODE_KEY
SALESORDER	ZDEMO_SOH	SALESORDER	<input type="checkbox"/>	SNWD_SO_ID
CUSTOMER	ZDEMO_SOH	BPARTNER	<input type="checkbox"/>	
FINALSTATUS	ZDEMO_SOH	OSTATUS	<input type="checkbox"/>	
CHANGEDAT	ZDEMO_SOH	CHANGEDAT	<input type="checkbox"/>	TIMESTAMPL

Editing the database view in SE11

More on this: [Creating a Database View](#)

- Set the maintenance status of the new database view to **Read and change**.

The maintenance status defines whether you can only read data with the view, or whether you can also insert and change data with it.



Maintenance status for writable database view in SE11

3. Activate the database view in transaction SE11.
4. In the source code of view (which defines the business object's data model), specify the database view for storing active data by using the corresponding @ObjectModel sub-annotation:

```
@ObjectModel: {
    transactionalProcessingEnabled: true,
    ...
    writeActivePersistence: '<database_view>',
    ...
}
```

8.11 Implementing Draft-Enabled Associations

A draft-enabled association retrieves active data, if you follow the association from the active source instance, and draft data, if you follow it from the draft source instance. An association that is not draft-enabled always retrieves active data even if the source entity is a draft instance.

Associations are used to navigate from a source entity of a business object to a related target entity. If the target entity is draft-enabled, there might be two versions of data: active data and draft data. Associations that are not draft-enabled always return the active data of the target entity, regardless of the draft status of the source entity. On the other hand, draft-enabled associations always return the version of data that matches the draft status of the source entity.

Associations that are Draft-Enabled by Default

In many cases, associations are draft-enabled by default as there is a composition or another close relationship between source and target entity.

The association types that are presented in the following list and in the figure below are automatically draft-enabled. Every other association is **not** draft-enabled and must be annotated with the relevant information to enable draft capabilities.

- Association to a child entity

Example: Association type #TO_COMPOSITION_CHILD from Sales Order (ROOT) to Sales Order Item (CHILD) or from Sales Order Item (PARENT) to Schedule Line (CHILD)

- Association to a parent entity

Example: Association type #TO_COMPOSITION_PARENT from Schedule Line (CHILD) to Sales Order Item (PARENT)

- Association to the root entity

Example: Association type #TO_COMPOSITION_ROOT from Schedule Line to Sales Order

- Inverse association of an association to the root entity

Example: Association from Sales Order (ROOT) to Schedule Line (CHILD of CHILD)

- Specialization of an association that is draft-enabled by default

We speak of a specialization if an association has a filter applied on it:

Example: Association type #TO_COMPOSITION_CHILD from Sales Order to Sales Order Item restricted to items in Euro.

i Note

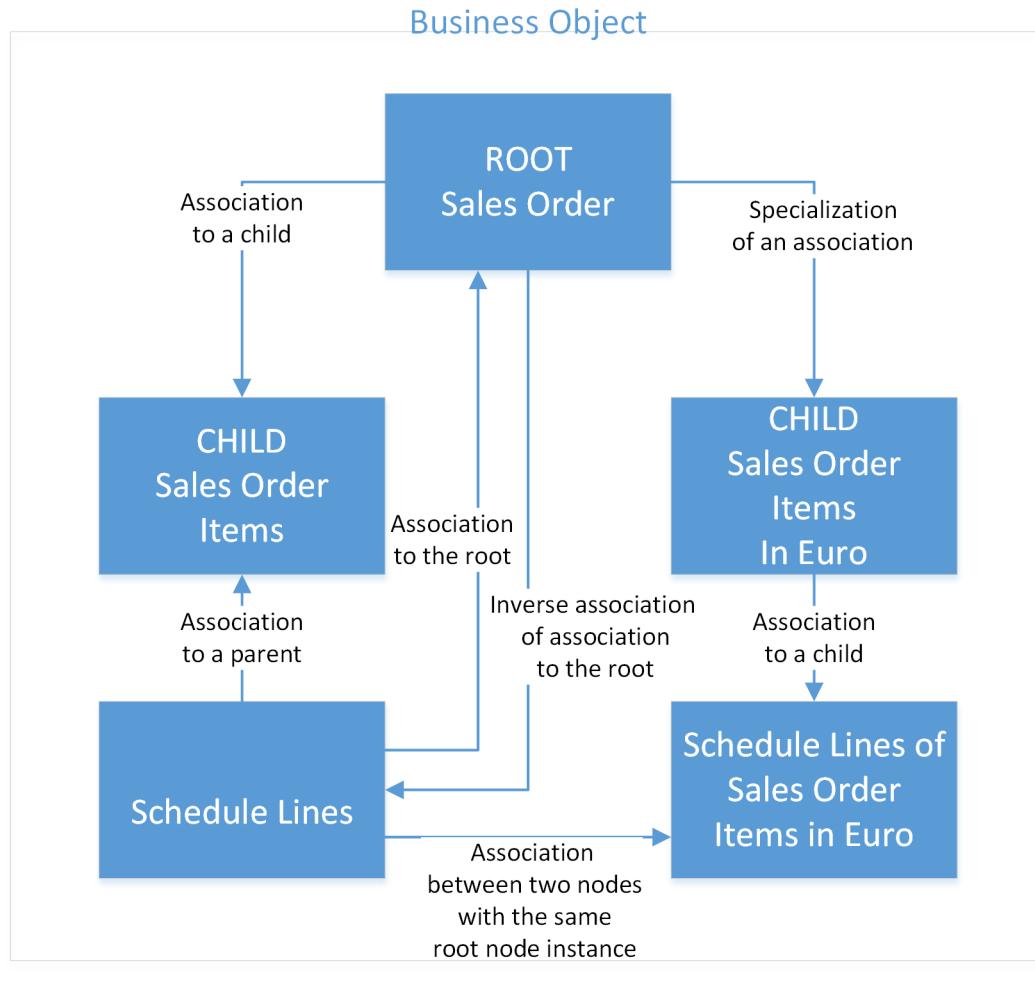
The restriction is implemented on the association in the element list: _item[CurrencyCode = 'EUR'] as _itemInEuro

- Association between two nodes that have the same root node instance

Example: Association from Schedule Line (CHILD of CHILD) to another Schedule Line (CHILD of CHILD) that belongs to the same Sales Order (ROOT) or

Association from Sales Order Item (CHILD) to another Sales Order Item (CHILD)

The following image illustrates the associations that are draft-enabled by default. The scenario that is used is a business object with a sales order root, sales order items, and schedule lines.



Associations that are draft-enabled by default

Associations that are not Draft-Enabled by Default

Apart from the association types listed above, all other associations are not draft-enabled automatically. They need to be annotated to draft-enable them. The procedure differs depending on the association cardinalities.

More on this:

[Draft-Enabling for Associations with Cardinality To-One \[page 291\]](#).

[Draft-Enabling for Reverse Associations \[page 294\]](#)

8.11.1 Draft-Enabling for Associations with Cardinality To-One

Associations that are not draft-enabled by default can be annotated to enable them with draft capabilities. This topic explains which annotations are used for associations with cardinality to-one and why they are needed.

Context

Associations can be automatically draft-enabled or can be made draft-enabled explicitly by implementation.

A list of automatically draft-enabled associations is provided here: [Associations that are Draft-Enabled by Default \[page 288\]](#)

All other associations must be annotated to draft-enable them. The most frequent use case among those associations that need annotations to draft-enable them is:

- Associations between two nodes of the same business object (BO) that neither have the same parent nor the same root node instance.

❖ Example

An association from a sales order item of sales order A to a sales order item of sales order B.

As the items belong to different sales orders, their root node instance is different. However, they are part of the same BO.

Prerequisites

- Both, the source entity and the target entity are draft-enabled. Draft-enabled entities have the CDS annotations specific to transactional processing and the CDS annotations specific to draft enablement. More on this: [CDS Annotations Specific to Transactional Processing \[page 114\]](#) and [CDS Annotations Specific to Draft Enablement \[page 115\]](#)
- The join condition uses the primary key of the target entity.
- The association cardinality is `[0..1]`. This means that a maximum of one target entity is associated with one source entity.

For associations with other cardinalities, check [Draft-Enabling for Reverse Associations \[page 294\]](#).

Relevant Annotations

To enable an association with draft capabilities, use the following `@ObjectModel` annotation on the association:

```
@ObjectModel.association.draft.Enabled: true  
_Assoc
```

This annotation ensures that the association path leads to active data if the source entity is active and to draft data if the source entity is a draft instance.

i Note

In UUID-based scenarios, draft data is automatically given a unique key as the UUID field (which is derived from the active table) is filled with a new UUID for every draft data record. As the join condition uses the primary key of the target entity (in this case the UUID field), the navigation to the draft data can use the same fields as the navigation between the active instances of the entity.

Additional Annotation for Scenarios with Semantic Key

In contrast to the UUID scenario, the semantic key is not unique for draft data as multiple draft entries might have the same key. For this reason, a new field (`DRAFTUUID`) is generated in the draft table to serve as a unique key field for draft entities. The UUID of the associated entity is then used for the join condition of the association and stored in an additional field in the draft source entity. This field, named `<prefix>DRAFTUUID`, is also generated automatically and is given a prefix that needs to be defined in an annotation. A prefix is necessary here to avoid a duplicate of the field `DRAFTUUID` in the source entity, in which the entity's own UUID is stored.

To determine the prefix, the following `@ObjectModel` annotation must be used on the association in the CDS view of the source entity in addition to the annotation mentioned above:

```
@ObjectModel.association.draft.fieldNamePrefix: 'prefix'  
_Assoc
```

In this way, the join condition between active entities, for which semantic keys are used, can be translated for the association between entities in draft versions, using the new draft UUID key of the target entity in the join condition.

Example

An association between instances of the same business object that do not have the same root node instance exemplifies this scenario. The business object in our case manages employee data; in particular the responsible manager for the respective employee. The reduced data model for this semantic key scenario is the following:

Entity	Element
employee	key empl_id
	first_name
	last_name
	manager_empl_id

Employees are listed with their names and an ID and their manager.

An association from an employee to their manager is an association that is not draft-enabled by default and fulfills the requirements to draft-enable it:

- The source and the target entity are the same (employee entity), which can be draft-enabled.
- The join condition uses the primary key of the target entity as the association is defined on the element `manager_empl_id` of the source entity and the element `empl_id` of the target entity.
- The association cardinality is [0..1] as one employee can just have one manager.

The following code example shows the CDS data definition for the BO-view of the employee with the relevant annotations to draft-enable the association. In this example the BO uses semantic key and therefore the association must be annotated with both annotations explained above.

```

@AbapCatalog.sqlViewName: 'I_EMPL'
@AbapCatalog.compiler.compareFilter: true
@AbapCatalog.preserveKey: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'BO-view employees'
@ObjectModel: {
    transactionalProcessingEnabled: true,
    compositionRoot: true,
    createEnabled: true,
    deleteEnabled: true,
    updateEnabled: true,
    writeActivePersistence: 'db_empl',
    semanticKey: ['empl_id'],
    draftEnabled: true,
    writeDraftPersistence: 'db_empl_d',
}
define view I_empl_tp_d
    as select from db_empl
        association [0..1] to I_empl_tp_d as _manager on $projection.manager_empl_id =
    manager.empl_id
{
    key empl_id,
    first_name,
    last_name,
    manager_empl_id,
    @ObjectModel.association.draft.enabled: true
    @ObjectModel.association.draft.fieldNamePrefix: 'manager'
    _manager
}

```

The following figure illustrates the table header of the draft table `db_empl_d` (in our example) with the generated fields.

DRAFTUUID	EMPL_ID	FIRST_NAME	LAST_NAME	MANAGER_EMPL_ID	MANAGERDRAFTUUID
FB75B6B371DC1E	500	Alexander	Barger	334	005H932487JHU902384

Draft Table Header of the Employee Entity

Related Information

[Implementing Draft-Enabled Associations \[page 288\]](#)

[Draft-Enabling for Reverse Associations \[page 294\]](#)

8.11.2 Draft-Enabling for Reverse Associations

Associations that are not draft-enabled by default can be annotated to enable them with draft capabilities. This topic explains which annotations are used for associations that are defined in the reverse direction of another association.

Context

Associations can be automatically draft-enabled or can be made draft-enabled explicitly by implementation. A list of automatically draft-enabled associations is provided here: [Associations that are Draft-Enabled by Default \[page 288\]](#)

The following topic describes how reverse associations are draft-enabled if they are not draft-enabled by default.

i Note

A reverse association is an association that connects two nodes for which an association in the other direction already exists. The source and the target entity are exchanged but the join condition uses exactly the same elements. This means that reverse associations do not necessarily use the primary key of the target entity in the join condition.

Use cases for defining reverse associations are associations that do not meet the prerequisites for draft-enabling them as described in the topic [Draft-Enabling for Associations with Cardinality To-One \[page 291\]](#). These are:

- Associations that do not use the primary key of the target source entity in the join condition.
- Associations with cardinality to-many.

These associations can be defined as reverse associations using an annotation. In this way, the navigation information of their counterparts can be used for these associations as well. This is crucial when semantic keys are used.

Prerequisites

- Both the source entity and the target entity must be draft-enabled. Draft-enabled entities must have the CDS annotations specific to transactional processing and the CDS annotations specific to draft enablement.
More on this: [CDS Annotations Specific to Transactional Processing \[page 114\]](#) and [CDS Annotations Specific to Draft Enablement \[page 115\]](#)
- The association is a reverse association of a draft-enabled association with cardinality to-one.

Relevant Annotations

To explicitly enable an association with draft capabilities, use the following `@ObjectModel` annotation on the association:

```
@ObjectModel.association.draft.Enabled: true  
_Assoc
```

This annotation ensures that for the defined association `_Assoc`, the association path leads to active data if the source entity is active and to draft data if the source entity is a draft instance.

To mark the association as a reverse association, add the following `@ObjectModel` annotation to the association:

```
@ObjectModel.association.reverseAssociation: 'Inverse_Assoc'  
_Assoc
```

For the value, specify the alias of the association to which the association `_Assoc` is a reverse association. This annotation enables the association to use the same join condition for draft instances as its counterpart (for example field prefixes in scenarios with semantic keys). In the same way the correct navigation is ensured when following an association with cardinality to-many.

Example

We consider the same example scenario as in the topic: [Draft-Enabling for Associations with Cardinality To-One \[page 291\]](#): an association between instances of the same business object. From one employee instance we want to be able to navigate to the employee's manager and in the other direction we want to get all the managed employees of one manager.

The association from manager to employees is the reverse association to the association from one employee to their manager, which has cardinality to one. We take for granted that this association is draft-enabled, which means, it is annotated with the relevant annotations, as it is done in the previous topic.

The reverse association, however, is `[1..*]` as one manager can manage multiple employees. The relevant annotations for draft-enabling this association are the subject of this example. The association must be annotated with `@ObjectModel.association.draftEnabled: true` and `@ObjectModel.association.reverseAssociation: 'Inverse_Assoc'`.

The following code example shows the CDS data definition for the business partner BO view with the relevant annotations for draft-enabling this association.

```
@AbapCatalog.sqlViewName: 'I_EMPL'  
@AbapCatalog.compiler.compareFilter: true  
@AbapCatalog.preserveKey: true  
@AccessControl.authorizationCheck: #CHECK  
@EndUserText.label: 'BO-View employees'  
@ObjectModel: {  
    transactionalProcessingEnabled: true,  
    compositionRoot: true,  
    createEnabled: true,  
    deleteEnabled: true,  
    updateEnabled: true,  
    writeActivePersistence: 'db_empl',  
    semanticKey: ['empl_id'],
```

```

        draftEnabled: true,
        writeDraftPersistence: 'db_empl_d',
    }
define view I_EMPL_TP_D
    as select from db_empl
    association [0..1] to I_EMPL_TP_D as _manager on $projection.manager_empl_id =
    _manager.empl_id
    association [0..*] to I_EMPL_TP_D as _employees on $projection.empl_id =
    _employees.manager_empl_id
{
    key empl_id,
    first_name,
    last_name,
    manager_empl_id,
    @ObjectModel.association.draft.enabled: true
    @ObjectModel.association.draft.fieldNamePrefix: 'manager'
    _manager,
    @ObjectModel.association.draft.enabled: true
    @ObjectModel.association.reverseAssociation: '_manager'
    _employees
}

```

The association _employees is labeled as reverse association for the association _manager. It adopts all the necessary features of its counterpart.

Related Information

[Implementing Draft-Enabled Associations \[page 288\]](#)

[Draft-Enabling for Associations with Cardinality To-One \[page 291\]](#)

8.12 Defining CDS Annotations for Metadata-Driven UIs

Metadata-driven UIs are dynamic UIs because metadata, namely CDS annotations in this context, are stored in a repository and can be retrieved from the client as needed. CDS annotations depend on the UI in which they are supposed to be used.

UIs might differ from user to user. Even though if, for example, three different users use the same application, each of them might have different permissions or different preferences, which results in different UI perspectives. Users might want to personalize their UIs and see different columns in tables, for example. CDS annotations offer default views for modelling UIs, however, CDS annotations can be overruled by personalization preferences.

The following chapters inform you about CDS annotations that you can use to define metadata-driven UIs, and answer the following questions:

- [How can I define the title of a field or a table? \[page 298\]](#)
- [How can I define the columns of a field or a table? \[page 299\]](#)
- [How can I define fields for filtering? \[page 301\]](#)
- [How can I define the header of an object-page floorplan? \[page 303\]](#)
- [How can I define the body of an object-page floorplan? \[page 305\]](#)

- How can I group fields? [page 306]
- How can I expose elements to UIs? [page 308]
- How can I overwrite default labels? [page 309]
- How can I position fields? [page 309]
- How can I prioritize fields? [page 310]
- How can I define charts? [page 311]
- How can I visualize criticality? [page 315]
- How can I visualize trends? [page 317]
- How can I visualize criticality based on trend calculation? [page 319]
- How can I visualize a person responsible and a reference period? [page 321]
- How can I use the dataField type #AS_DATAPOINT? [page 324]
- How can I define a contact? [page 324]
- How can I define navigation between screens? [page 326]
- How can I define navigation to external web sites? [page 327]
- How can I define navigation based on actions executed on semantic objects? [page 329]
- How can I define actions? [page 331]
- How can I display fields in a text area? [page 333]
- How can I mask fields? [page 334]
- How can I hide fields? [page 335]
- How can I define interaction between annotations? [page 336]

8.12.1 Tables and Lists

Get an overview of how to use UI annotations for lists and tables for SAP Fiori UIs.

In Fiori we distinguish tables and list. Both mostly hold homogeneous data, but lists hold in general rather simple data whereas tables hold usually more complex ones.

Lists are mostly used in the master list section of the master-detail floorplan and in popovers or dialogs. Sure there is also the possibility to have them in full-screen floorplans for certain use cases.

A table contains a set of line items and usually comprises rows (one row showing one line item) and columns. Line items can contain data of any kind, but also interactive controls, for example, for editing the data, navigating, or triggering actions relating to the line item.

To display large amounts of data in tabular form, several table controls are provided. These are divided into two groups, each of which is defined by a consistent feature set:

- Fully responsive tables
- Desktop-centric tables

In order to expose a CDS view in a table-like or list-like format, you can use the annotations explained in the following sections::

- Title [page 298]
- Columns [page 299]
- Selection Fields [page 301]

8.12.1.1 Title

Get information about what UI annotations to use to work with titles of lists or tables for SAP Fiori UIs.

If necessary, you can provide a header for your list or table.

The screenshot shows a SAP Fiori application interface. At the top left is a search bar with placeholder text 'Company:' and a magnifying glass icon. To the right are buttons for 'Hide Filter Bar', 'Filters', and a blue 'Go' button. Below the search bar is a title bar with the text 'Sales Orders (1,190)' in yellow and 'Standard * ⓘ'. On the far right of the title bar are a gear icon and a copy icon. The main area is a table with the following data:

Sales Order ID	Company	Currency Code	Gross Amount
500000000	SAP	EUR	25,867.03 EUR >
500000001	DelBont Industries	EUR	14,602.49 EUR >
500000002	TECUM	EUR	5,631.08 EUR >
500000003	Asia High tech	EUR	1,704.04 EUR >
500000004	Asia High tech	EUR	761.24 EUR >
500000005	AVANTEL	EUR	101,299.22 EUR >
500000006	Talpa	EUR	250.73 EUR >
500000007	Panorama Studios	EUR	10,311.35 EUR >
500000008	Telecomunicaciones Star	EUR	195.16 EUR >
500000009	SAP	EUR	3,972.22 EUR >
500000010	DelBont Industries	EUR	827.95 EUR >
500000011	Panorama Studios	EUR	325.94 EUR >
500000012	TECUM	EUR	24,704.40 EUR >
500000013	Asia High tech	EUR	8,256.22 EUR >
500000014	Asia High tech	EUR	3,459.33 EUR >
500000015	AVANTEL	EUR	862.73 EUR >
500000016	Panorama Studios	EUR	70.18 EUR >

Example of title of table

You can use the following UI annotation to define what can be displayed in the title of a table or a list:

-

↳ Sample Code

```
...
@UI.headerInfo: { typeNamePlural: 'Sales Orders' }
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    ...
}
```

Related Information

[Tables and Lists \[page 297\]](#)

[Columns \[page 299\]](#)

[Selection Fields \[page 301\]](#)

8.12.1.2 Columns

Get information about what UI annotations to use to work with columns of lists or tables for SAP Fiori UIs.

What columns are needed for a table or list depends on the use case, for example, an overview table may require more fields than a value help list. For this reason, you can define several list layouts and table layouts that are distinguished by a qualifier, for example 'ValueList'.

If a CDS view contains analytical annotations, for example the `@DefaultAggregation` annotation, the UI automatically takes this into consideration and no additional UI annotations are required.

The screenshot shows a Fiori application interface for sales orders. At the top, there's a search bar and filter controls. Below that, a table lists sales orders. The table has columns for Sales Order ID, Company, Currency Code, and Gross Amount. Some rows are highlighted in yellow, indicating they are aggregated. For example, the first two rows show sales from 'African Gold ...' and 'Alpine Systems' respectively, with a total gross amount of 84,676.32 EUR. Below these, a section for 'Company: Anav Ideon' is expanded, showing individual sales entries for each order ID, all of which sum up to a total gross amount of 101,299.22 EUR.

<input type="checkbox"/>	Sales Order ID	Company	Currency Code	Gross Amount
>	Company: African Gold ...	African Gold And Diamond Corporation	EUR	84,676.32 EUR
>	Company: Alpine Systems	Alpine Systems	EUR	84,676.32 EUR
▼	Company: Anav Ideon			
<input type="checkbox"/>	500000105	Anav Ideon	EUR	101,299.22 EUR
<input type="checkbox"/>	500000115	Anav Ideon	EUR	862.73 EUR
<input type="checkbox"/>	500000123	Anav Ideon	EUR	411.50 EUR
<input type="checkbox"/>	500000128	Anav Ideon	EUR	1,704.04 EUR
<input type="checkbox"/>	500000140	Anav Ideon	EUR	541.31 EUR
<input type="checkbox"/>	500000147	Anav Ideon	EUR	521.22 EUR
<input type="checkbox"/>	500000355	Anav Ideon	EUR	101,299.22 EUR
<input type="checkbox"/>	500000365	Anav Ideon	EUR	862.73 EUR
<input type="checkbox"/>	500000373	Anav Ideon	EUR	411.50 EUR
<input type="checkbox"/>	500000378	Anav Ideon	EUR	1,704.04 EUR
<input type="checkbox"/>	500000390	Anav Ideon	EUR	541.31 EUR
<input type="checkbox"/>	500000397	Anav Ideon	EUR	521.22 EUR
<input type="checkbox"/>	500000605	Anav Ideon	EUR	101,299.22 EUR
<input type="checkbox"/>	500000615	Anav Ideon	EUR	862.73 EUR
<input type="checkbox"/>	500000623	Anav Ideon	EUR	411.50 EUR
<input type="checkbox"/>	500000628	Anav Ideon	EUR	1,704.04 EUR
<input type="checkbox"/>	500000640	Anav Ideon	EUR	541.31 EUR
<input type="checkbox"/>	500000647	Anav Ideon	EUR	521.22 EUR
<input type="checkbox"/>	500000855	Anav Ideon	EUR	101,299.22 EUR

Example of @DefaultAggregation annotation

For more information about the `@DefaultAggregation` annotation, see section *DefaultAggregation Annotations* linked below.

You can use the following UI annotation to define what can be displayed in the title of a table or a list:

-

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    @UI.lineItem: [ { position: 10 }, { qualifier: 'ValueList', position:
    10 } ]
    key so.sales_order_id as SalesOrder,
    @UI.lineItem: [ { position: 20 }, { qualifier: 'ValueList', position:
    20 } ]
    so.customer.company_name as CompanyName,
    @UI.lineItem: [ { position: 30 } ]
    so.currency_code as CurrencyCode,
    @DefaultAggregation: #SUM
}
```

```
@UI.lineItem: [ { position: 40 } ]  
  so.gross_amount as GrossAmount  
}
```

For more information about positioning, see section *Positioning Fields* linked below.

Related Information

[Aggregation Annotations \[page 414\]](#)

[Positioning Fields \[page 309\]](#)

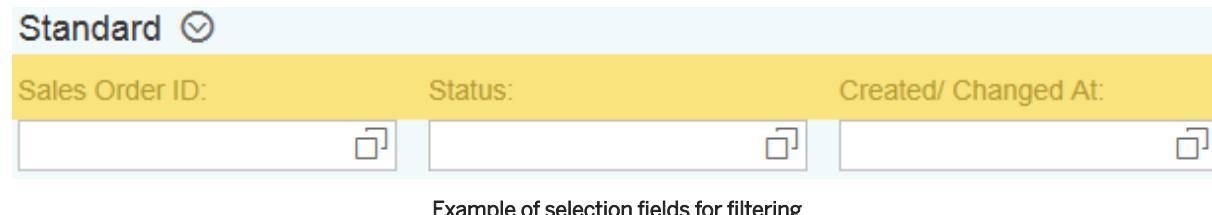
[Tables and Lists \[page 297\]](#)

[Title \[page 298\]](#)

[Selection Fields \[page 301\]](#)

8.12.1.3 Selection Fields

Get information about what UI annotations to use to work with selection fields for SAP Fiori UIs.



Example of selection fields for filtering

If a CDS is annotated as [@Search.searchable \[page 443\]](#), the UI automatically takes this into consideration and no additional UI annotations are required to expose a search field or a value help. If your table or list contains many data and therefore many rows, it gets hard to find the information you need. To facilitate finding the desired information, you can use selection fields to specify the range of information that you are looking for.

The screenshot shows a Fiori application interface. At the top, there is a search bar with the placeholder "Search" and a dropdown menu set to "Standard". Below the search bar is a filter bar with a "Company:" field containing a placeholder icon. To the right of the filter bar are buttons for "Hide Filter Bar", "Filters", and "Go".

The main area displays a table titled "Sales Orders (1,190) | Standard". The table has columns: "Sales Order ID", "Company", "Currency Code", and "Gross Amount". The data is grouped by company, with African Gold, Alpine Systems, and Anav Ideon each having two entries. The table includes a header row and a footer row showing totals for each group.

	Sales Order ID	Company	Currency Code	Gross Amount
> Company: African Gold ...	African Gold And Diamond Corporation	EUR	84,676.32 EUR	
> Company: Alpine Systems	Alpine Systems	EUR	84,676.32 EUR	
▼ Company: Anav Ideon				
500000105	Anav Ideon	EUR	101,299.22 EUR	
500000115	Anav Ideon	EUR	862.73 EUR	
500000123	Anav Ideon	EUR	411.50 EUR	
500000128	Anav Ideon	EUR	1,704.04 EUR	
500000140	Anav Ideon	EUR	541.31 EUR	
500000147	Anav Ideon	EUR	521.22 EUR	
500000355	Anav Ideon	EUR	101,299.22 EUR	
500000365	Anav Ideon	EUR	862.73 EUR	
500000373	Anav Ideon	EUR	411.50 EUR	
500000378	Anav Ideon	EUR	1,704.04 EUR	
500000390	Anav Ideon	EUR	541.31 EUR	
500000397	Anav Ideon	EUR	521.22 EUR	
500000605	Anav Ideon	EUR	101,299.22 EUR	
500000615	Anav Ideon	EUR	862.73 EUR	
500000623	Anav Ideon	EUR	411.50 EUR	
500000628	Anav Ideon	EUR	1,704.04 EUR	
500000640	Anav Ideon	EUR	541.31 EUR	
500000647	Anav Ideon	EUR	521.22 EUR	
500000855	Anav Ideon	EUR	101,299.22 EUR	

Example for @Search.searchable annotation

If your table or list contains many data and therefore many rows, it gets hard to find the You can use the following UI annotation to enable specific elements for selection, for example using a filter bar:

-

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    key so.sales_order_id as SalesOrder,
    @UI.selectionField: [ { position: 10 } ]
    so.customer.company_name as CompanyName,
    ...
}
```

Related Information

[Tables and Lists \[page 297\]](#)

[Title \[page 298\]](#)

[Columns \[page 299\]](#)

8.12.2 Detail Pages

Get an overview of how to use UI annotations for object-page floorplans for SAP Fiori UIs.

You can use the object-page floorplan if you need to display, create, or edit any object regardless of its complexity level. You can use the object-page floorplan with either a facet (tabs) or flat (anchors) approach.

To expose a CDS view in an object-page floorplan, you can use the annotations explained in the following sections:

- [Page Header \[page 303\]](#)
- [Page Body \[page 305\]](#)

8.12.2.1 Page Header

Get information about what UI annotations to use to work with page headers of object-page floorplans for SAP Fiori UIs.

The page header contains information on the object you are editing in the object-page floorplan, for example.

The screenshot shows a SAP Fiori UI page for a Sales Order. At the top, there is a header bar with a back arrow on the left and a 'Sales Order' tab on the right. Below the header, there are two green rectangular boxes containing annotations: 'Sales Order: 500000000' and 'Customer: SAP'. The main content area is titled 'GENERAL INFORMATION' and contains a section titled 'General Information' with the following details:

Sales Order ID:	500000000
Company:	SAP
Currency Code:	EUR
Gross Amount:	25,867.03 EUR

Example of @UI.headerInfo for page header of object-page floorplan

You can use the following UI annotations to use several properties to influence the header section of the object-page floorplan:

- [@UI.headerInfo \[page 478\]](#)

« Sample Code

```
@UI.headerInfo: {
    typeName: 'Sales Order',
    title: {
        label: 'Sales Order',
        value: 'SalesOrder'      -- Reference to element in element list
    },
    description: {
        label: 'Customer',
        value: 'CompanyName'     -- Reference to element in element list
    }
}
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    key so.sales_order_id as SalesOrder,
    so.customer.company_name as CompanyName,
    ...
}
```

This UI annotation can be considered as the combination of the UI annotations [@UI.headerInfo \[page 478\]](#) and . The properties imageUrl, typeImageUrl and title should usually correspond to the properties of the UI annotation [@UI.headerInfo \[page 478\]](#). In addition to the title, a headLine, mainInfo and secondaryInfo of the same format can be specified.

« Sample Code

```
@UI.badge: {
    title: {
        label: 'Sales Order',
        value: 'SalesOrderID'      -- Reference to element in element list
    },
    headLine: {
        label: 'Customer',
        value: 'CompanyName'     -- Reference to element in element list
    },
    mainInfo: {
        label: 'Gross Amount',
        value: 'GrossAmount'      -- Reference to element in element list
    },
    secondaryInfo: {
        label: 'Billing Status',
        value: 'BillingStatus'    -- Reference to element in element list
    }
}
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    key so.sales_order_id as SalesOrder,
    so.customer.company_name as CompanyName,
    so.gross_amount as GrossAmount,
    so.billing_status as BillingStatus,
    ...
}
```

This UI annotation can be used to display status information of an entity on the UI, for example the delivery status or payment status of an entity. This annotation is similar to the annotation. However, the annotation is usually used together with the criticality property instead of the qualifier property.

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    key so.sales_order_id as SalesOrder,
    @UI.statusInfo: [ { position: 10 } ]
    so.delivery_status as DeliveryStatus,
    @UI.statusInfo: [ { position: 20 } ]
    so.billing_status as BillingStatus,
    @UI.statusInfo: [ { position: 30 } ]
    so.lifecycle_status as LifecycleStatus,
    ...
}
```

Related Information

[Detail Pages \[page 303\]](#)

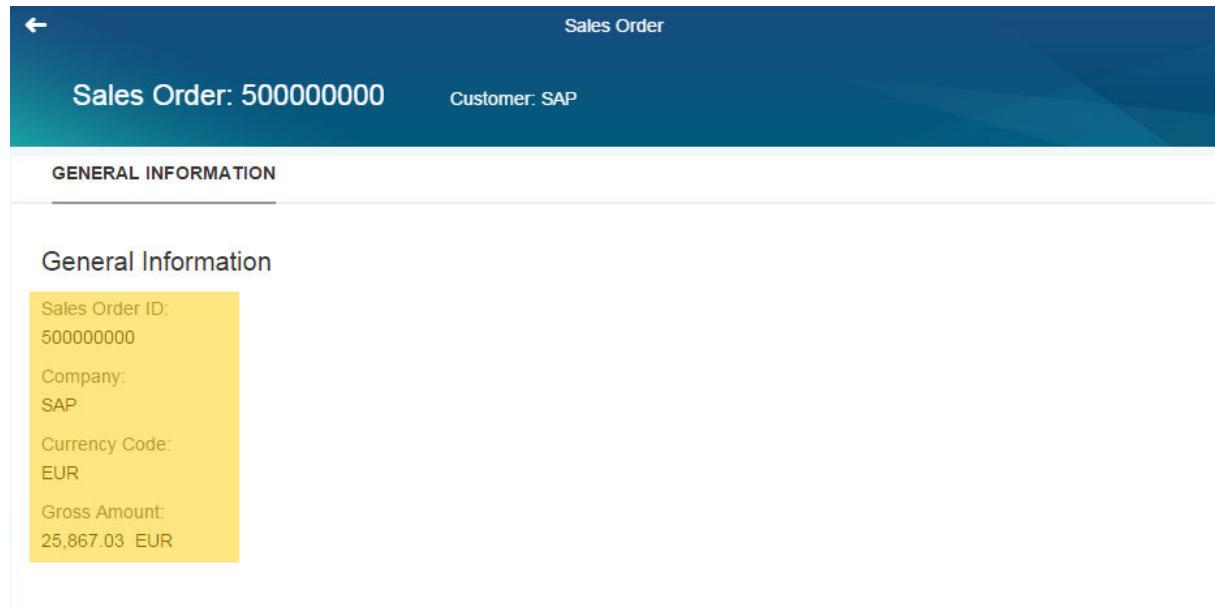
[Page Body \[page 305\]](#)

[Defining Criticality of Field Values \[page 311\]](#)

8.12.2.2 Page Body

Get information about what UI annotations to use to work with page bodies of object-page floorplans for SAP Fiori UIs.

The page body can consist of a list or a table, for example, in which you can see and edit details of an object from the master-detail floorplan.



The screenshot shows a SAP Fiori Sales Order page. At the top, there's a header bar with a back arrow, the title "Sales Order", and a sub-title "Customer: SAP". Below the header, the sales order number "Sales Order: 500000000" is displayed. The main content area is titled "GENERAL INFORMATION". A table is shown with the following data:

General Information	
Sales Order ID:	500000000
Company:	SAP
Currency Code:	EUR
Gross Amount:	25,867.03 EUR

Example of columns of table on object-page floorplan

You can use the following UI annotation to define what elements are displayed in the page body of the object-page floorplan:

- This annotation is similar to the UI annotation , but has no qualifier.

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    @UI.identification: [ { position: 10 } ]
    key so.sales_order_id as SalesOrder,
    @UI.identification: [ { position: 20 } ]
    so.customer.company_name as CompanyName,
    @UI.identification: [ { position: 30 } ]
    so.currency_code as CurrencyCode,
    @UI.identification: [ { position: 40 } ]
    so.gross_amount as GrossAmount
}
```

Related Information

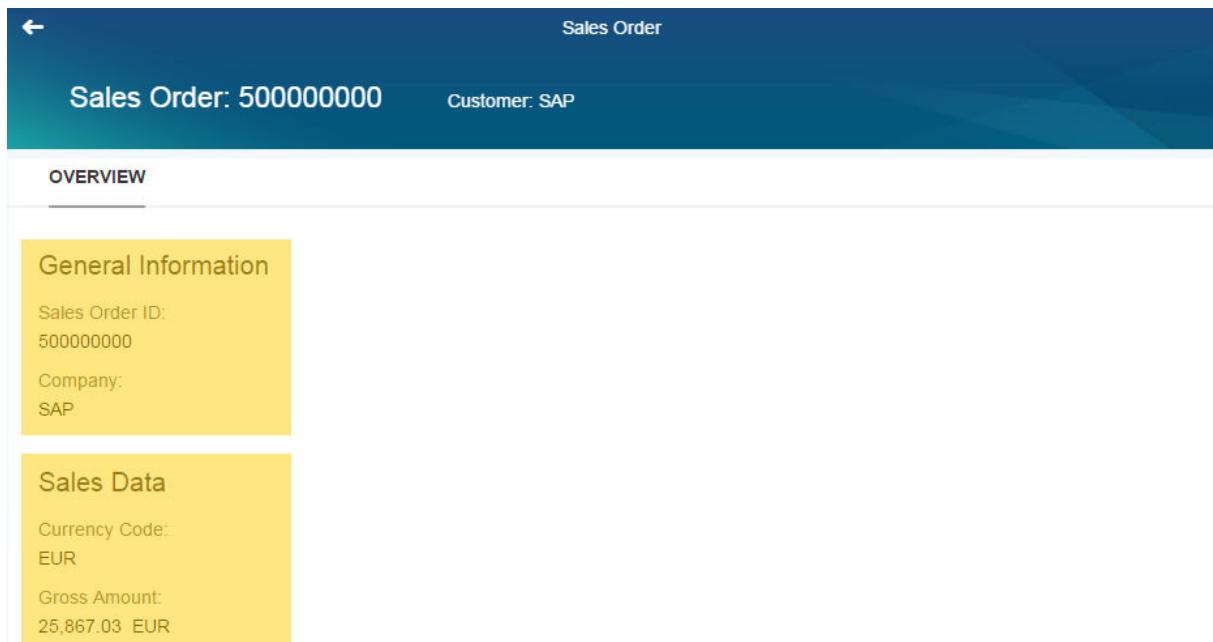
[Detail Pages \[page 303\]](#)

[Page Header \[page 303\]](#)

8.12.3 Field Groups

Get information about what UI annotations to use to work with field groups for SAP Fiori UIs.

If you want to group fields under one heading to consolidate semantically connected information, you can use field groups. With field groups, you can build sections for forms, for example.



Example of field group

You can use the following UI annotation to group several fields:

- This annotation is similar to the UI annotation because the different field groups have unique qualifiers.

↳ Sample Code

```
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    @UI.fieldGroup: [ { qualifier: 'GeneralInformation', position: 10 } ]
    key so.sales_order_id as SalesOrder,
    @UI.fieldGroup: [ { qualifier: 'GeneralInformation', position: 20 } ]
    so.customer.company_name as CompanyName,
    @UI.fieldGroup: [ { qualifier: 'SalesData', position: 10 } ]
    so.currency_code as CurrencyCode,
    @UI.fieldGroup: [ { qualifier: 'SalesData', position: 20 } ]
    so.gross_amount as GrossAmount,
}
```

8.12.4 Annotations Similar to `dataField`

Get an overview of how to use UI annotations that are similar to the OData annotation `dataField`.

The OData annotation `dataField` refers to a property of the OData service that is used.

Some annotations are syntactically similar or even identical. These annotations are the following:

-
-

-
-
-

These annotations are called **dataField-like annotations** in the following sections:

- [Exposing Elements \[page 308\]](#)
- [Overwriting Default Labels \[page 309\]](#)
- [Positioning Fields \[page 309\]](#)
- [Prioritizing UI Elements \[page 310\]](#)

8.12.4.1 Exposing Elements

Get information about how to expose elements to SAP Fiori UIs.

You can use dataField-like annotations to reference elements from a different CDS view using to-one-associations. You therefore need to explicitly define the elements with a value property. These elements are then exposed to the UI.

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so
  association [0..1] to sepm_cds_business_partner as _BusinessPartner
    on $projection.buyer_guid = _BusinessPartner.business_partner_key
{
  key so.sales_order_id as SalesOrder,
  so.buyer_guid,
  ...
  @UI.Identification: [
    { value: '_BusinessPartner.company_name', position: 110 },
    { value: '_BusinessPartner.bp_role', position: 120 }
  ]
  _BusinessPartner
}
```

Related Information

[Annotations Similar to dataField \[page 307\]](#)

[Overwriting Default Labels \[page 309\]](#)

[Positioning Fields \[page 309\]](#)

[Prioritizing UI Elements \[page 310\]](#)

8.12.4.2 Overwriting Default Labels

Get information about how to overwrite default labels for SAP Fiori UIs.

If a CDS element is exposed via a dataField-like annotation, the label is by default derived from the CDS annotation `@EndUserText.label` if available, or from a DDIC element.

If you want a default label to be overwritten by a specific label, for example *Customer* instead of *Business*, you can use the label property.

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so
  association [0..1] to sepm_cds_business_partner as _BusinessPartner
    on $projection.buyer_guid = _BusinessPartner.business_partner_key
{
  key so.sales_order_id as SalesOrder,
  so.buyer_guid,
  ...
  @UI.Identification: [
    { value: '_BusinessPartner.company_name', position: 110, label: 'Customer
Name' },
    { value: '_BusinessPartner.bp_role', position: 120, label: 'Customer
Role' }
  ]
  _BusinessPartner
}
```

Related Information

[Annotations Similar to dataField \[page 307\]](#)

[Exposing Elements \[page 308\]](#)

[Positioning Fields \[page 309\]](#)

[Prioritizing UI Elements \[page 310\]](#)

8.12.4.3 Positioning Fields

Get information about how to change the position of fields on SAP Fiori UIs.

To define the order of fields in the UI, you can use the position property of dataField-like annotations. Only the positioning order is relevant, so you can use any decimal number as value for the positioning property.

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
  @UI.identification: [ { position: 1 } ]
  key so.sales_order_id as SalesOrder,
```

```

@UI.identification: [ { position: -5 } ]
so.customer.company_name as CompanyName,
@UI.identification: [ { position: 9999 } ]
so.currency_code as CurrencyCode,
@UI.identification: [ { position: '1.1' } ]
so.gross_amount as GrossAmount
}

```

Related Information

[Annotations Similar to dataField \[page 307\]](#)

[Exposing Elements \[page 308\]](#)

[Overwriting Default Labels \[page 309\]](#)

[Prioritizing UI Elements \[page 310\]](#)

8.12.4.4 Prioritizing UI Elements

Get information about how to set the priority of elements displayed on SAP Fiori UIs.

To define the priority of elements, you can use the `importance` property of `dataField`-like annotations. This information is relevant for adaptive UIs. If a UI is displayed on a small screen, elements with low priority can automatically be hidden. To define importance, you can choose the following values:

- #HIGH
- #MEDIUM
- #LOW
- not set

↳ Sample Code

```

...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    @UI.identification: [ { position: 10, importance: #HIGH } ]
    key so.sales_order_id as SalesOrder,
    @UI.identification: [ { position: 20, importance: #MEDIUM } ]
    so.customer.company_name as CompanyName,
    @UI.identification: [ { position: 30, importance: #LOW } ]
    so.currency_code as CurrencyCode,
    @UI.identification: [ { position: 40 } ]
    so.gross_amount as GrossAmount
    ...
}

```

Related Information

[Annotations Similar to dataField \[page 307\]](#)

[Exposing Elements \[page 308\]](#)

[Overwriting Default Labels \[page 309\]](#)

[Positioning Fields \[page 309\]](#)

8.12.4.5 Defining Criticality of Field Values

Get information about how to define the criticality of field values for SAP Fiori UIs.

To define if a field value is negative, critical, or positive, you can use the `criticality` property of `dataField`-like annotations. This property must refer to a CDS element that has the value 1 (negative), 2 (critical), or 3 (positive).

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    @UI.identification: [ { position: 10, importance: #HIGH } ]
    key so.sales_order_id as SalesOrder,
    ...
    @UI.statusInfo: [ { position: 10, criticality: 'GrossAmountCrit' } ]
    so.billing_status as BillingStatus,
    so.billing_status_crit as BillingStatusCrit,
    ...
}
```

Related Information

[Annotations Similar to dataField \[page 307\]](#)

[Exposing Elements \[page 308\]](#)

[Overwriting Default Labels \[page 309\]](#)

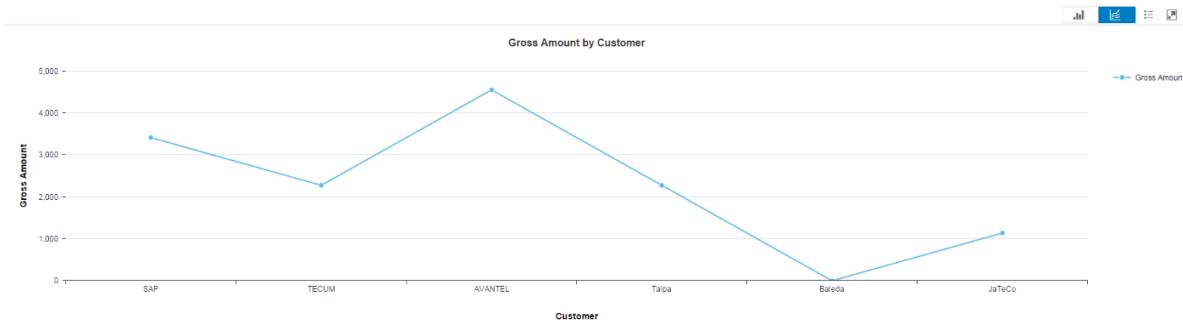
[Positioning Fields \[page 309\]](#)

[Prioritizing UI Elements \[page 310\]](#)

8.12.5 Charts

Get information about what UI annotations to visualize data on SAP Fiori UIs.

If you want to visualize data, you can use a chart.



Example of a line chart

You can use the following UI annotation to define the properties of a chart:

- You define this UI annotation at view level. It refers to the elements that are to be used in the chart. Additionally, you can provide a title and description.

↳ Sample Code

```
...
@UI.chart: {
    title: 'Gross Amount by Customer',
    description: 'Line-chart displaying the gross amount by customer',
    chartType: #LINE,
    dimensions: [ 'CompanyName' ],      -- Reference to one element
    measures: [ 'GrossAmount' ]         -- Reference to one or more elements
}
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    key so.sales_order_id as SalesOrder,
    so.customer.company_name as CompanyName,
    so.currency_code as CurrencyCode,
    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.gross_amount as GrossAmount,
    ...
}
```

Related Information

[Charts \[page 312\]](#)

8.12.5.1 Charts

Get an overview of what chart types you can use to visualize data on SAP Fiori UIs.

Each chart type has different restrictions referring to how many dimensions and measures are required or allowed. The following table lists the admissible types and their restrictions.

Chart Types for Data Visualization

Type	Dimensions	Measures
COLUMN	One dimension	One or more measures
COLUMN_STACKED	Displayed on the x-axis	Displayed on the y-axis
COL- UMN_STACKED_100		
AREA		
AREA_STACKED		
AREA_100		
LINE		
BAR	One dimension	One or more measures
BAR_STACKED	Displayed on the y-axis	Displayed on the x-axis
BAR_STACKED_100		
HORIZONTAL_AREA		
HORIZON- TAL_AREA_STACKED		
HORIZON- TAL_AREA_100		
PIE	One dimension	One Measure
DONUT	For segmentation	For size of segment
SCATTER	Two dimensions	Up to two measures (symbol and color)
BUBBLE	One for the x-axis, one for the y-axis	One measure (size of bubble)
RADAR	Three or more dimen- sions	No measures
HEAT_MAP	Two dimensions	One measure (color)
	One for the x-axis, one for the y-axis	

Type	Dimensions	Measures
TREE_MAP	One or more hierarchical dimensions	One measure (rectangle size)
		One optional measure (color)
WATERFALL	One dimension	One measure
	Displayed on the x-axis	Displayed on the y-axis

Related Information

[Charts \[page 311\]](#)

8.12.6 Data Points

Get an overview of how to use data points to display criticality, trends, and references to people and time periods on SAP Fiori UIs.

In some cases, you want to visualize a single point of data that typically is a number that can be enriched with business-relevant data but may also be textual, for example a status value.

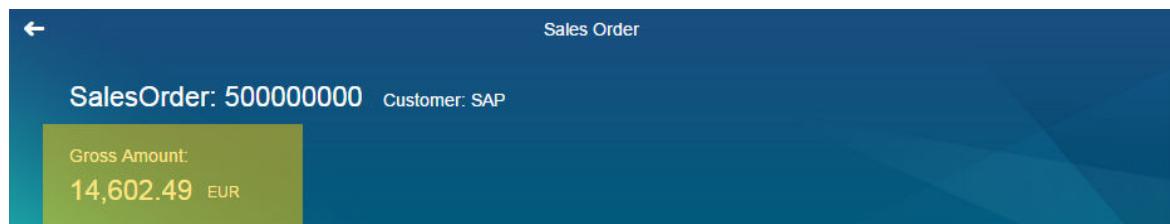


Example of data points

You can use the following UI annotation to define a single point of data:

-

You can, for example, express if a high or a low value is desired, or if a value is increasing or decreasing. The simplest variant of the UI annotation consists of the `title` property.



Example of simple variant of data point

In the following example, only the title is exposed to the UI.

Sample Code

```
...
define view ZExample_SalesOrdersByCustomer as select from ... as so {
    key so.buyer_guid as BuyerGuid,
    @Semantics.currencyCode: true
    so.currency_code as CurrencyCode,

    @UI.dataPoint: { title: 'Gross Amount' }
    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.actual_amount as ActualAmount
}
```

Related Information

[Criticality \[page 315\]](#)

[Trends \[page 317\]](#)

[Trend-Criticality Calculation \[page 319\]](#)

[Person Responsible and Reference Period \[page 321\]](#)

[DataField Type: #AS_DATAPOINT \[page 324\]](#)

8.12.6.1 Criticality

Get information about how to use data points to display criticality on SAP Fiori UIs.

A more usable variant of the UI annotation also contains information about the criticality, the trend, and the name of a person responsible.

You can use the sub-annotation to express if a value is positive or negative, for example.

You can use the sub-annotation to express if a value has decreased or increased, for example.

In this case, the properties `targetValue`, `criticality`, and `trend` are already evaluated in the CDS view. In the CDS view, the target value is already calculated, and if the current value thus is negative or positive, and if the current value has improved or declined, for example. These values are only referred to from the annotation.

Data can be defined as being either positive, critical, or negative. These data can be statuses, for example.

You can use the following sub-annotation to highlight criticality:

- You define this UI annotation at view level. It refers to the elements that are to be used in the chart. Additionally, you can provide a title and description. The table below lists the values that are valid for the UI annotation , and shows how these values are visualized on the UI:

Values and Visualization of Criticality

Value	Description	Visualization in Color
1	Negative	Red
2	Critical	Yellow
3	Positive	Green

↳ Sample Code

```
...
define view ZExample_SalesOrdersByCustomer as select from ... as so {
    key so.buyer_guid as BuyerGuid,
    @Semantics.currencyCode: true
    so.currency_code as CurrencyCode,

    @UI.dataPoint: {
        title: 'Gross Amount',
        targetValueElement: 'TargetAmount',      -- Reference to element
        criticality: 'AmountCriticality',       -- Reference to element
        trend: 'AmountTrend',                  -- Reference to element
    }
    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.actual_amount as ActualAmount,

    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.target_amount as TargetAmount,

    so.criticality as AmountCriticality,
    so.trend as AmountTrend
}
```

Related Information

[Data Points \[page 314\]](#)

[Trends \[page 317\]](#)

[Trend-Criticality Calculation \[page 319\]](#)

[Person Responsible and Reference Period \[page 321\]](#)

[DataField Type: #AS_DATAPOINT \[page 324\]](#)

8.12.6.2 Trends

Get information about how to use data points to display trends on SAP Fiori UIs.

Data can be defined as being either increasing, decreasing, or stable. These data can be measured over a certain period of time and visualized on the UI.

You can use the following sub-annotations to highlight trends:

-

❖ Example

For an example, see the example code in section *Criticality* linked below.

-

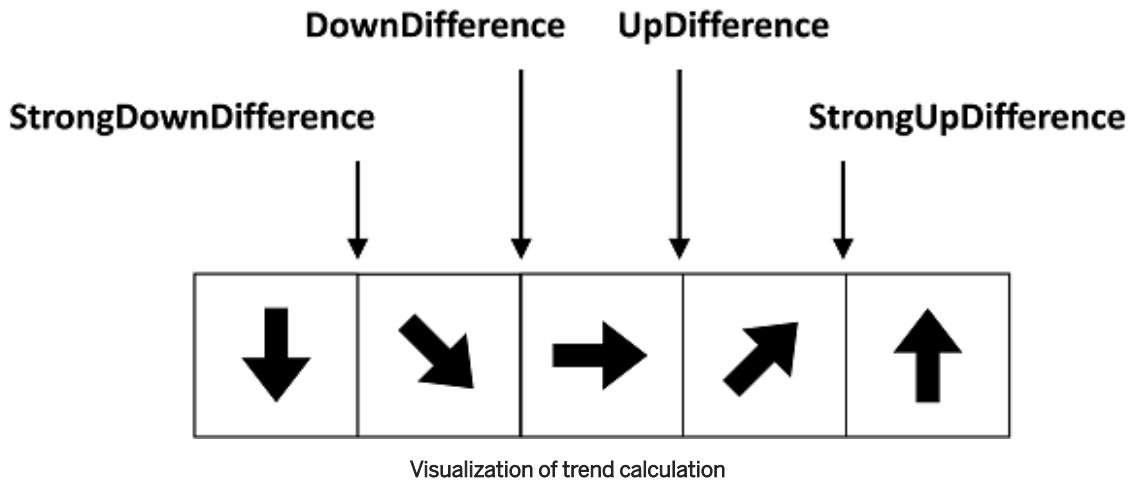
❖ Example

The table below lists the values that are valid for the UI annotation , and shows how these values are visualized on the UI:

Values and Visualization of Trend

Value	Description	Visualization
1	Strong up	
2	Up	
3	Sideways	
4	Down	
5	Strong down	

For the trend calculation, the flag `isRelativeDifference` indicates whether the absolute or the relative difference between the actual value and the reference value is used to calculate the trend.



↳ Sample Code

```
...
define view ZExample_SalesOrdersByCustomer as select from ... as so {
    key so.buyer_guid as BuyerGuid,
    @Semantics.currencyCode: true
    so.currency_code as CurrencyCode,

    @UI.dataPoint: {
        title: 'Gross Amount',
        //...
        trendCalculation: {
            referenceValue: 'ReferenceAmount',      -- Reference to element
            isRelativeDifference: true,           -- Comparison of ratio
            strongUpDifference: 1.25,
            upDifference: 1.1,
            downDifference: 0.9,
            strongDownDifference: 0.75
        }
    }
    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.target_amount as TargetAmount,
    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.reference_amount as ReferenceAmount
}
```

Related Information

[Criticality \[page 315\]](#)

[Data Points \[page 314\]](#)

[Trend-Criticality Calculation \[page 319\]](#)

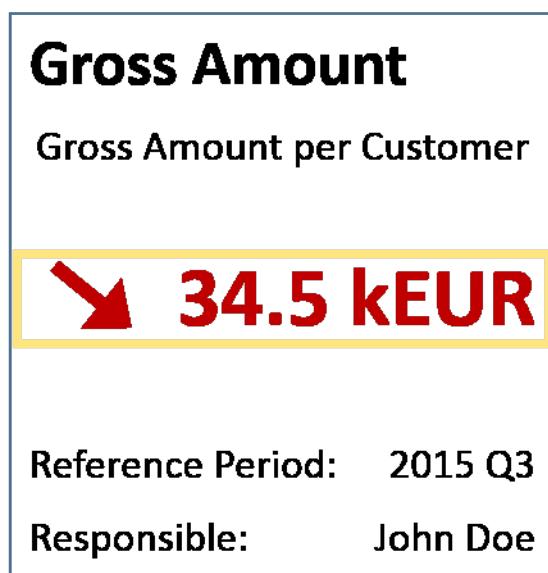
[Person Responsible and Reference Period \[page 321\]](#)

[DataField Type: #AS_DATAPOINT \[page 324\]](#)

8.12.6.3 Trend-Criticality Calculation

Get information about how to use data points to calculate and display trend-criticality relations.

Another way to specify properties of criticality and trend is to define rules for criticality and trend within the UI annotation.



Example of visualization of trend-criticality calculation

You can use the following sub-annotations to calculate trends and derive from these calculation the criticality of data:

-
-

Sample Code

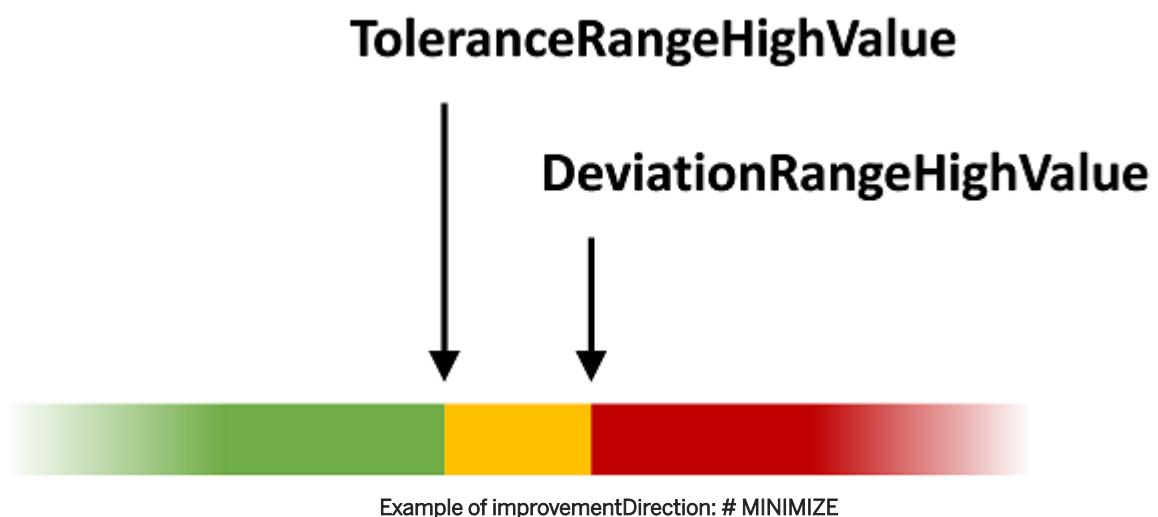
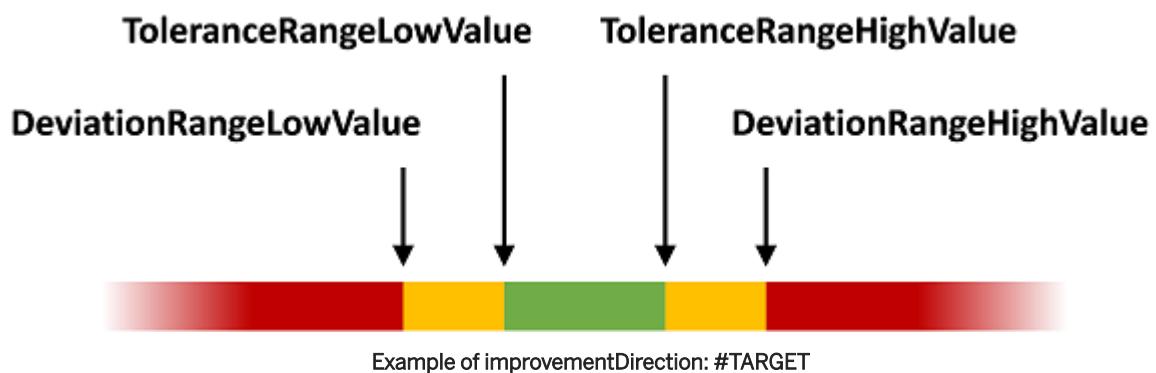
```
...
define view ZExample_SalesOrdersByCustomer as select from ... as so {
    key so.buyer_guid as BuyerGuid,
    @Semantics.currencyCode: true
    so.currency_code as CurrencyCode,
    ...
    @UI.dataPoint: {
        title: 'Gross Amount',
        targetValue: 9216,
        criticalityCalculation: {
            improvementDirection: #TARGET,
            toleranceRangeLowValue: 9200,
            toleranceRangeHighValue: 9300,
            deviationRangeLowValue: 8800,
            deviationRangeHighValue: 9700
        },
        trendCalculation: {
            referenceValue: 'ReferenceAmount',      -- Reference to element
            isRelativeDifference: false,           -- Comparison of difference
            strongUpDifference: 100,
            upDifference: 10,
            downDifference: -10,
            strongDownDifference: -100
        }
    }
}
```

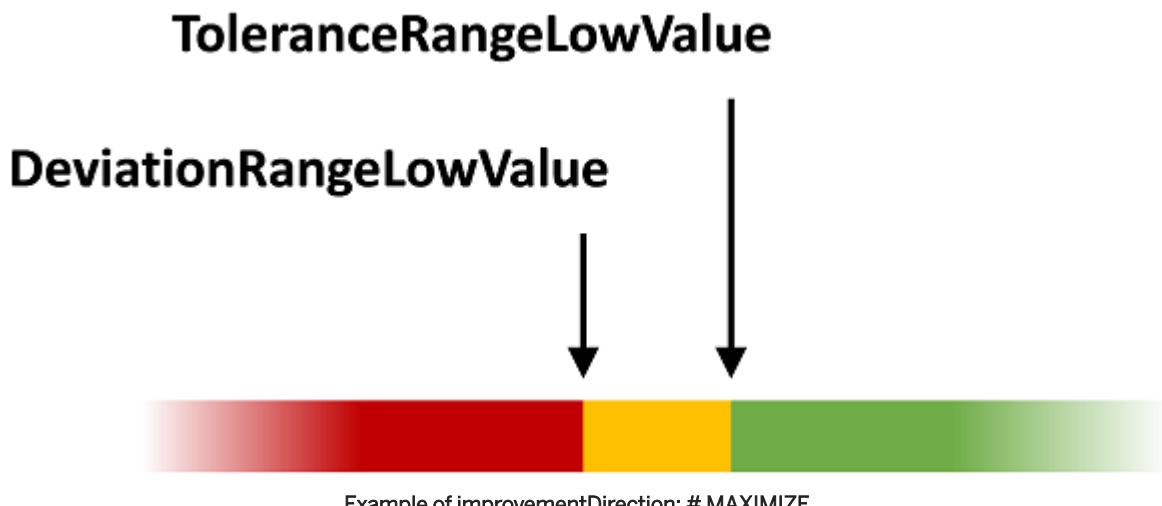
```

        }
    }
    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.target_amount as TargetAmount,
    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.reference_amount as ReferenceAmount
}

```

For the criticality calculation, the value of the property `improvementDirection` is crucial because this value determines what further properties are needed. If, for example, the value is `#MINIMIZE`, the properties `ToleranceRangeHighValue` and `DeviationRangeHighValue` are relevant.





The properties of the sub-annotation can have either constant values or derive values from referencing to other elements. If a property references to another element, the suffix `Element` must be added to the name of the property.

i Note

This also applies to the properties of the sub-annotation , except for the property `referenceValue`. This property always references to another element.

• Example

`toleranceRangeLowValue` becomes `toleranceRangeLowValueElement`.

Related Information

- [Data Points \[page 314\]](#)
- [Criticality \[page 315\]](#)
- [Trends \[page 317\]](#)
- [Person Responsible and Reference Period \[page 321\]](#)
- [DataField Type: #AS_DATAPOINT \[page 324\]](#)

8.12.6.4 Person Responsible and Reference Period

Get information about how to use data points to display references to persons responsible and to reference periods on SAP Fiori UIs.

You can add the following properties to the UI annotation :

- `referencePeriod`

- responsibleName

You can define both properties either in the UI annotation directly, or in another element and reference from the UI annotation to this element.

❖ Example

In the following example, the data point has a static reference period and a static person responsible. The value of the gross amount is formatted with the `valueFormat` property. The value is thus scaled with factor 1000 and is displayed with one decimal place, this is the value 34500 EUR would be displayed as `34.5 kEUR`.



Example of visualization of person responsible, reference period, and value format

↳ Sample Code

```
...
define view ZExample_SalesOrdersByCustomer as select from ... as so {
    key so.buyer_guid as BuyerGuid,
    @Semantics.currencyCode: true
    so.currency_code as CurrencyCode,

    @UI.dataPoint: {
        title: 'Gross Amount',
        description: 'Gross Amount per Customer',
        longDescription: 'The gross amount per customer ...',
        valueFormat: {
            scaleFactor: 1000,
            number_of_fractional_digits: 1
        },
        referencePeriod: { description: '2015 Q3' },
        responsibleName: 'John Doe'
    }
    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.actual_amount as ActualAmount
}
```

• Example

In the following example, a dynamic reference period is used that is supplied by the following parameters:

- start
- end

These parameters have to be aliased in the element list before they can be used in the annotation. The responsible property must refer to a to-one-association. The target entity of this association should contain the contact data of the person responsible.

↳ Sample Code

```
...
define view ZExample_SalesOrdersByCust
with parameters p_StartDate : abap.dats,
      p_EndDate   : abap.dats
as select from ... as so
  association [0..1] to Employees as _PersonResponsible
    on _PersonResponsible.EmployeeId = $projection.PersonResponsible
{
  ...
  $parameters.p_StartDate as StartDate,          -- Alias is required for
annotation
  $parameters.p_EndDate as EndDate,              -- Alias is required for
annotation
  so.person_responsible as PersonResponsible,
  @Semantics.currencyCode: true
  so.currency_code as CurrencyCode,
  @UI.dataPoint: {
    title: 'Gross Amount',
    referencePeriod: {
      start: 'StartDate',                      -- Reference to element
      end: 'EndDate'                         -- Reference to element
    },
    responsible: '_PersonResponsible'         -- Reference to association
  }
  @Semantics.amount.currencyCode: 'CurrencyCode'
  @DefaultAggregation: #SUM
  so.actual_amount as ActualAmount,
  _PersonResponsible
}
where so.validity_date >= $parameters.p_StartDate
and   so.validity_date <= $parameters.p_EndDate
```

For a definition of element list, see section *Glossary* linked below.

Related Information

[Glossary \[page 606\]](#)

[Data Points \[page 314\]](#)

[Criticality \[page 315\]](#)

[Trends \[page 317\]](#)

[Trend-Criticality Calculation \[page 319\]](#)

[DataField Type: #AS_DATAPOINT \[page 324\]](#)

8.12.6.5 DataField Type: #AS_DATAPOINT

Get information about how to use the type #AS_DATAPOINT to refer to other annotations.

The type #AS_DATAPOINT maps to *DataFieldForAnnotation*. *DataFieldForAnnotation* is used to refer to other annotations using the *Edm.AnnotationPath* abstract type. The annotation path must end in *vCard.Address* or *UI.dataPoint*.

You can use the following type to reference an exposed data point from dataField-like annotations:

- #AS_DATAPOINT

You use this type to include a microchart in the UI annotation , for example.

❖ Example

In this example, the UI annotation has to be defined at the same CDS element as the UI annotation itself.

❖ Sample Code

```
...
define view ZExample_SalesOrdersByCustomer as select from ... as so {
    key so.buyer_guid as BuyerGuid,
    ...
    @Semantics.currencyCode: true
    so.currency_code as CurrencyCode,
    @UI.dataPoint: { title: 'Gross Amount' }
    @UI.lineItem: [ { type: #AS_DATAPOINT } ]
    @Semantics.amount.currencyCode: 'CurrencyCode'
    so.actual_amount as ActualAmount
}
```

Related Information

[Data Points \[page 314\]](#)

[Criticality \[page 315\]](#)

[Trends \[page 317\]](#)

[Trend-Criticality Calculation \[page 319\]](#)

[Person Responsible and Reference Period \[page 321\]](#)

8.12.7 Contact Data

Get information about what UI annotations to use to display contact data on SAP Fiori UIs.

In some cases users of an application need to see contact data, for example, of business partners, customers, or employees.

You can use the following annotation set to inform a client that an entity contains contact information and map the CDS elements to the corresponding address field:

- [@Semantics](#)

This annotation set contains annotations to inform about telephone numbers, email addresses, names, addresses, and contacts.

❖ Example

The following example contains sub-annotations belonging to the annotation set [@Semantics](#). For a complete list, see section *Semantics Annotations* linked below.

↳ Sample Code

```
...
define view Employees as select from ...
{
    key EmployeeId,
    @Semantics.name.givenName
    FirstName,
    @Semantics.name.additionalName
    MiddleName,
    @Semantics.name.familyName
    LastName,
    GenderCode,
    @Semantics.telephone.type: [#WORK, #PREF]
    PhoneNumber,
    @Semantics.telephone.type: [#FAX]
    FaxNumber,
    @Semantics.telephone.type: [#CELL]
    MobilePhoneNumber,
    @Semantics.eMail.address
    EmailAddress,
    PreferredLanguage,
    @Semantics.contact.birthDate
    BirthDate
}
```

Related Information

[Semantics Annotations \[page 445\]](#)

8.12.8 Navigation

Get an overview of how to use *dataField* types to provide means of navigation on SAP Fiori UIs.

It often is not sufficient to stay on one screen. Users might need to navigate between screens or even to web sites outside an application. You can use the following *dataField* types to include navigation concepts:

- [#WITH_NAVIGATION_PATH \[page 326\]](#)

Used for navigation within an application.

- [#WITH_URL \[page 327\]](#)

Used for navigation from an application to an external web site.

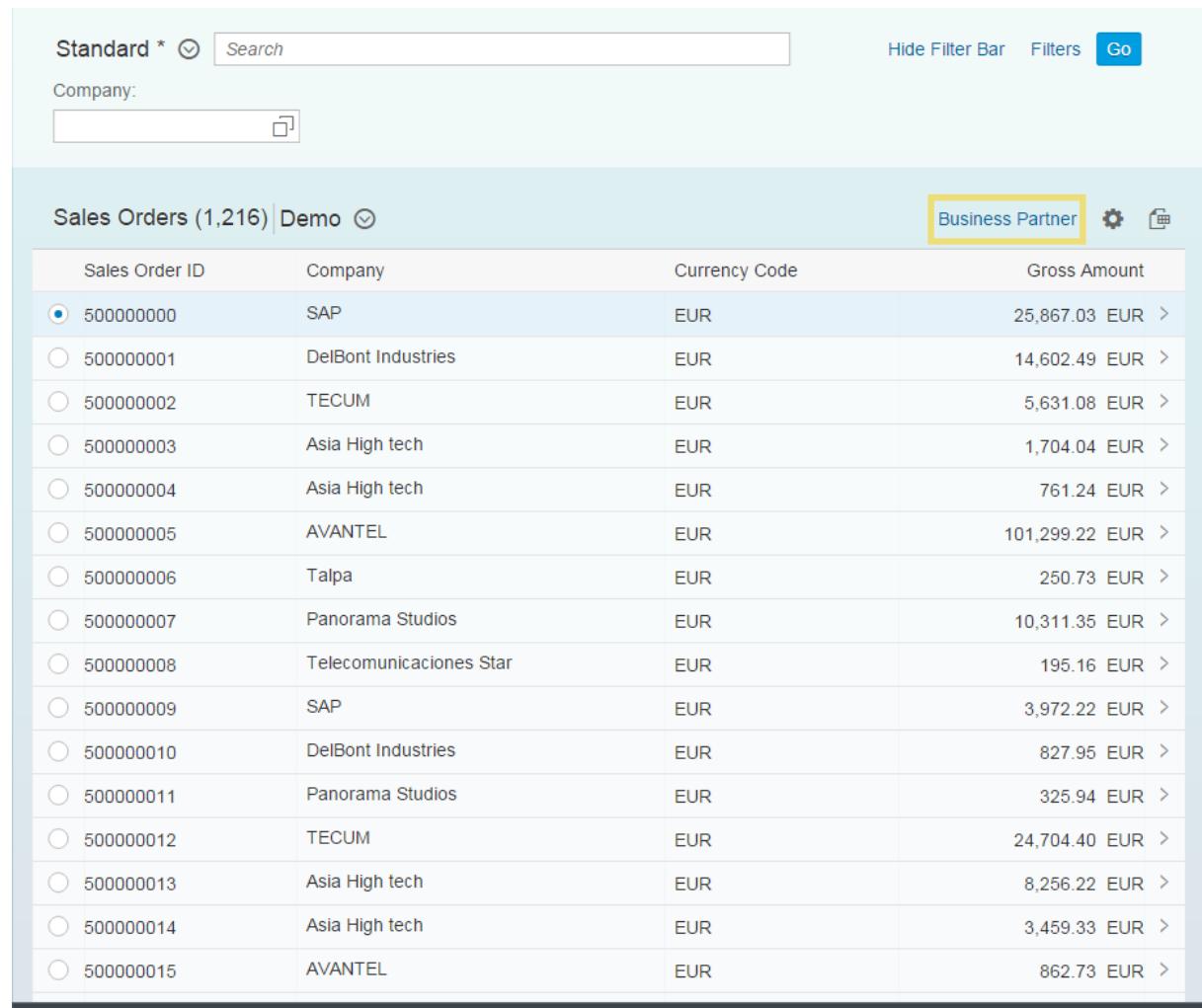
- [#FOR_INTENT_BASED_NAVIGATION \[page 329\]](#)

Used for navigation based on an action that is related to a semantic object.

8.12.8.1 With Navigation Path

Get information about how to provide navigation between UI screens and pages on SAP Fiori UIs.

This navigation type contains either a navigation property or a term cast. The term either is of type Edm.EntityType, a concrete entity type, or a collection of these types.



Sales Orders (1,216)		Demo	Business Partner	Settings	Print
Sales Order ID	Company	Currency Code	Gross Amount		
500000000	SAP	EUR	25,867.03 EUR >		
500000001	DelBont Industries	EUR	14,602.49 EUR >		
500000002	TECUM	EUR	5,631.08 EUR >		
500000003	Asia High tech	EUR	1,704.04 EUR >		
500000004	Asia High tech	EUR	761.24 EUR >		
500000005	AVANTEL	EUR	101,299.22 EUR >		
500000006	Talpa	EUR	250.73 EUR >		
500000007	Panorama Studios	EUR	10,311.35 EUR >		
500000008	Telecomunicaciones Star	EUR	195.16 EUR >		
500000009	SAP	EUR	3,972.22 EUR >		
500000010	DelBont Industries	EUR	827.95 EUR >		
500000011	Panorama Studios	EUR	325.94 EUR >		
500000012	TECUM	EUR	24,704.40 EUR >		
500000013	Asia High tech	EUR	8,256.22 EUR >		
500000014	Asia High tech	EUR	3,459.33 EUR >		
500000015	AVANTEL	EUR	862.73 EUR >		

Example of dataField of type #WITH_NAVIGATION_PATH

You can use the following dataField type to expose a link to other pages of a UI:

- #WITH_NAVIGATION_PATH

Example

In the following example, CompanyName is displayed as link referring to the association _BusinessPartner.

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as
so
    association [0..1] to sepm_cds_business_partner as _BusinessPartner
    on $projection.buyer_guid = _BusinessPartner.business_partner_key
{
    key so.sales_order_id as SalesOrder,
    so.buyer_guid,
    ...

    @UI.lineItem: [ {
        position: 20,
        type: #WITH_NAVIGATION_PATH,
        targetElement: '_BusinessPartner'      -- Reference to association
    } ]
    so.customer.company_name as CompanyName,
    ...
    _BusinessPartner
}
```

Related Information

[Navigation \[page 325\]](#)

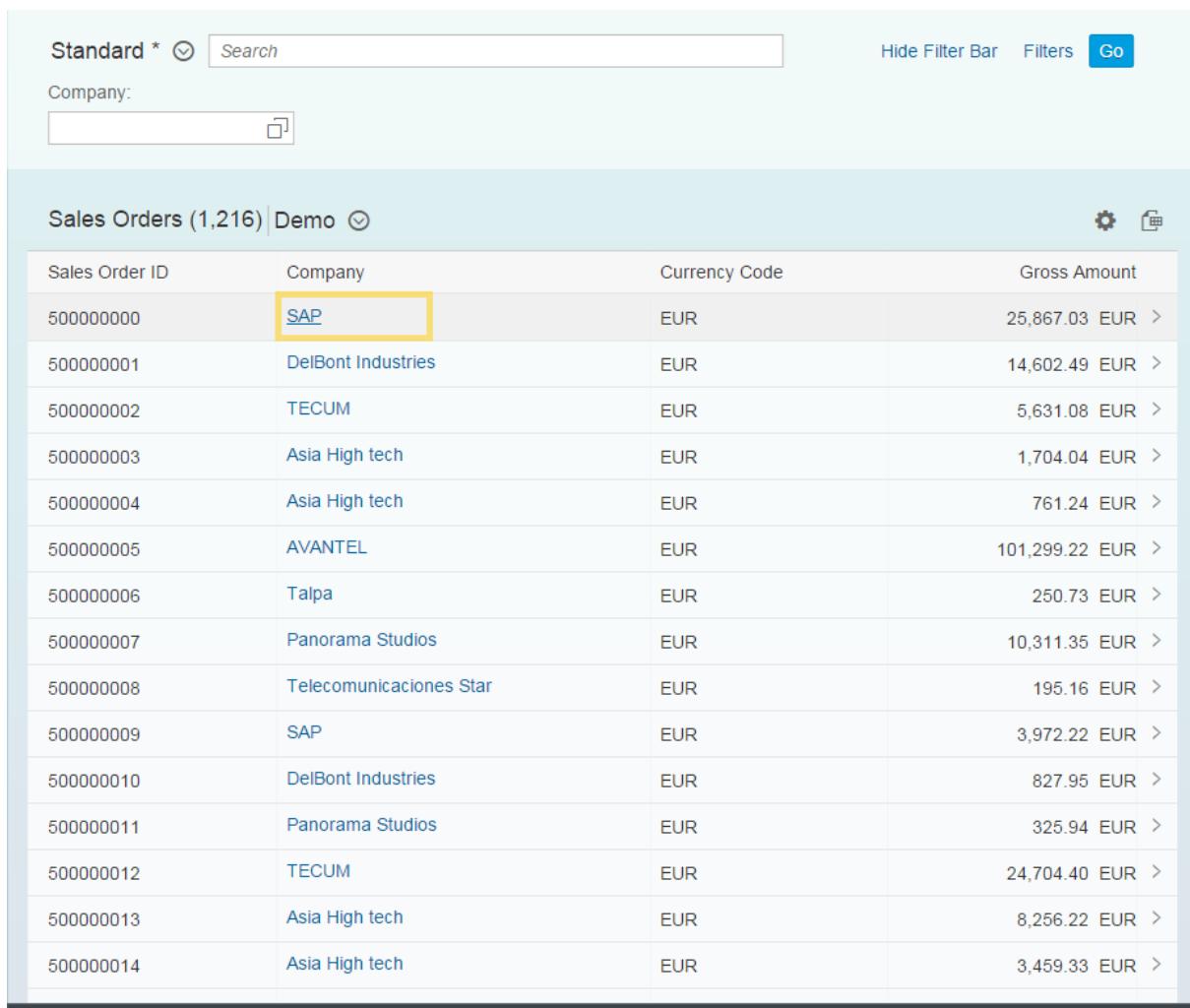
[With URL \[page 327\]](#)

[Based on Intent \[page 329\]](#)

8.12.8.2 With URL

Get information about how to provide navigation from SAP Fiori UIs to external web sites, for example.

This type navigation type contains a reference to a URL to navigate to specific web sites, for example.



Sales Order ID	Company	Currency Code	Gross Amount
500000000	SAP	EUR	25,867.03 EUR >
500000001	DelBont Industries	EUR	14,602.49 EUR >
500000002	TECUM	EUR	5,631.08 EUR >
500000003	Asia High tech	EUR	1,704.04 EUR >
500000004	Asia High tech	EUR	761.24 EUR >
500000005	AVANTEL	EUR	101,299.22 EUR >
500000006	Talpa	EUR	250.73 EUR >
500000007	Panorama Studios	EUR	10,311.35 EUR >
500000008	Telecomunicaciones Star	EUR	195.16 EUR >
500000009	SAP	EUR	3,972.22 EUR >
500000010	DelBont Industries	EUR	827.95 EUR >
500000011	Panorama Studios	EUR	325.94 EUR >
500000012	TECUM	EUR	24,704.40 EUR >
500000013	Asia High tech	EUR	8,256.22 EUR >
500000014	Asia High tech	EUR	3,459.33 EUR >

Example of dataField of type #WITH_URL

You can use the following `dataField` type to display links to external websites:

- `#WITH_URL`

❖ Example

In the following example, `CompanyName` is displayed as link referring to the CDS element `WebsiteUrl`.

≡ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as
so
{
  key so.sales_order_id as SalesOrder,
  ...

  @UI.lineItem: [ {
    position: 20,
    type: #WITH_URL,
    url: 'WebsiteUrl'      -- Reference to element
  } ]
  so.customer.company_name as CompanyName,
```

```
    so.customer.web_address as WebsiteUrl,  
    ...  
}
```

Related Information

[Navigation \[page 325\]](#)

[With Navigation Path \[page 326\]](#)

[Based on Intent \[page 329\]](#)

8.12.8.3 Based on Intent

Get information about how to provide navigation related on actions that are executed on SAP Fiori UIs.

This navigation type contains an action that is related to a semantic object. This combination of action and semantic object is an **intent**. The annotation is required for navigation based on intent. The client decides how to react when this navigation is triggered.

The screenshot shows a SAP Fiori application interface. At the top, there is a header with the SAP logo and a user dropdown set to 'Default User'. Below the header, the page title is 'AnnoDokuExmaple'. A navigation bar includes a back arrow, a search input field, and buttons for 'Hide Filter Bar', 'Filters', and 'Go'. The main content area displays a table titled 'Sales Orders (1,216) | Demo'. The table has four columns: 'Sales Order ID', 'Company', 'Currency Code', and 'Gross Amount'. The first row is selected, indicated by a blue circle. The 'Show customer-details' button in the top right corner of the table is highlighted with a yellow box. The table contains 15 rows of sales order data.

Example of dataField of type #FOR_INTENT_BASED_NAVIGATION

You can use the following `dataField` type to expose the intent to navigate without specifying how this navigation is to be resolved:

- `#FOR_INTENT_BASED_NAVIGATION`

❖ Example

In the following example, the intent 'Show' (action) 'BusinessPartner' (semantic object) is expressed. The client can, for example, open a separate application to display the details of the corresponding business partner.

« Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as
so
{
```

```
key so.sales_order_id as SalesOrder,
...
@UI.lineItem: [ {
  position: 20,
  label: 'Show customer-details',
  type: #FOR_INTENT_BASED_NAVIGATION,
  semanticObjectAction: 'Show' -- Action
} ]
@Consumption.semanticObject: 'BusinessPartner' -- Semantic Object
so.customer.company_name as CompanyName,
...
}
```

Related Information

[Navigation \[page 325\]](#)

[With Navigation Path \[page 326\]](#)

[With URL \[page 327\]](#)

8.12.9 Actions

Get information about how to use *dataField* types to provide means of executing actions on SAP Fiori UIs.

Actions are directly related to items that you can see in a table on a master-detail floorplan, for example. Users can select items and execute certain actions on the selected items.

The screenshot shows a SAP Fiori application interface. At the top, there's a header with the SAP logo and a user dropdown set to 'Default User'. Below the header, the page title is 'AnnoDokuExmaple'. A search bar with placeholder 'Search' and a dropdown menu are visible. The main content area displays a table titled 'Sales Orders (1,216) | Demo'. The table has four columns: 'Sales Order ID', 'Company', 'Currency Code', and 'Gross Amount'. Each row contains a radio button next to the Sales Order ID. The first row, which has the radio button selected, corresponds to 'SAP' with a Gross Amount of 25,867.03 EUR. To the right of the table are three icons: 'Copy' (highlighted with a yellow box), 'Settings', and 'Print'. The bottom right corner of the content area has a small navigation icon.

Example of action 'Copy' on master-detail floorplan

You can use the following `dataField` type to expose actions to the client:

- `#FOR_ACTION`

This property has to be assigned to some arbitrary element. It is thereby irrelevant if the property refers to the element to which the property is assigned.

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so {
    @UI.lineItem: [
        -- Standard Lineitem
        { position: 10 },
        -- Action Lineitem
        { type: #FOR_ACTION, dataAction: 'BOPF:Copy', label: 'Copy' }
    ]
    key so.sales_order_id as SalesOrder,
    ...
}
```

```
}
```

8.12.10 Field Manipulation

Get information about what UI annotations to use to manipulate fields for SAP Fiori UIs.

This chapter describes annotations that influence the appearance exposed fields. When a field is marked with these annotations, it is manipulated no matter in what other annotations the field is used. The reason for this is that annotations for manipulation are self-contained annotations and not properties of other annotations.

For example, when a field is marked with the annotation, the field is masked regardless if it is used in a annotation or a annotation.

To manipulate the appearance of fields on SAP Fiori UIs, you can use the annotations explained in the following sections:

- [Multi-Line Text \[page 333\]](#)
- [Field Masking \[page 334\]](#)
- [Field Hiding \[page 335\]](#)
- [Interaction with Other Annotations \[page 336\]](#)

8.12.10.1 Multi-Line Text

Get information about what UI annotations to use to display fields as multi-line text on SAP Fiori UIs.

You can use the following annotation to mark a field to be displayed by a control that supports multi-line input, for example a text area:

-

↳ Sample Code

```
...
define view Product as select from ... {
    @UI.identification: [ { position: 10 } ]
    key ProductID,
    @UI.identification: [ { position: 20 } ]
    ProductName,
    @UI.identification: [ { position: 30 } ]
    @UI.multiLineText: true
    Description,
    ...
}
```

Related Information

[Field Manipulation \[page 333\]](#)

[Field Masking \[page 334\]](#)

[Field Hiding \[page 335\]](#)

[Interaction with Other Annotations \[page 336\]](#)

8.12.10.2 Field Masking

Get information about what UI annotations to use to mask fields, for example for password input, on SAP Fiori UIs.

In some cases, data of fields need to be consumed by the client, but must not be visible on the UI. This field behavior is required when users need to enter passwords, for example.

You can use the following annotation to mark a field to not to be displayed in clear text by the client because, for example, it contains sensitive data:

-

This annotation does not influence how data is transferred. If a field is marked with the annotation, the data belonging to this field is still transferred to the client like any other property in clear text.

↳ Sample Code

```
...
define view Destination as select from ... {
    @UI.identification: [ { position: 10 } ]
    key DestinationID,
    ...

    @UI.identification: [ { position: 20 } ]
    AuthType,          -- None, Basic, SSO, ...
    ...

    @UI.identification: [ { position: 30 } ]
    BasicAuthUserName,
    @UI.identification: [ { position: 40 } ]
    @UI.masked
    BasicAuthPassword,
    ...
}
```

Related Information

[Field Manipulation \[page 333\]](#)

[Multi-Line Text \[page 333\]](#)

[Field Hiding \[page 335\]](#)

[Interaction with Other Annotations \[page 336\]](#)

8.12.10.3 Field Hiding

Get information about what UI annotations to use hide fields from SAP Fiori UIs.

Generally, all fields that are exposed by the OData service are available to the client, regardless if the fields are exposed explicitly using UI annotations. To enable end-user personalization, the client may offer the possibility to add fields that are hidden by default, for example to a list report.

You can use the following annotation to prevent fields from being displayed on a UI and in the personalization dialog, but leaving the field available for client:

- You can use this annotation if, for example, a CDS view contains technical keys, for example GUIDs, that have to be exposed to the OData service to work. These keys are usually not supposed to be displayed on the UI. You can also use this annotation if fields are required in calculations, but are not supposed to be displayed on a UI.

❖ Example

In the following example, the annotation with pre-calculated criticality and trend is exposed. The hidden fields `AmountCriticality` and `AmountTrend` are required by the client to calculate the corresponding values, but are not supposed to be displayed on the UI.

❖ Sample Code

```
...
define view ZExample_SalesOrdersByCustomer as select from ... as so {
    @UI.hidden
    key so.buyer_guid as BuyerGuid,
    ...

    @UI.dataPoint: {
        criticality: 'AmountCriticality',          -- Reference to element
        trend: 'AmountTrend',                      -- Reference to element
    }
    so.actual_amount as ActualAmount,
    @UI.hidden
    so.criticality as AmountCriticality,
    @UI.hidden
    so.trend as AmountTrend
}
```

You can use the following annotation to prevent fields from being available to a client:

- Preventing fields from being available to a client is necessary for system parameters. These parameters are filled by the runtime engine, but must not be available to the client.

❖ Sample Code

```
...
define view OverdueSalesOrder with parameters
    @Consumption.hidden : true
    @Environment.systemField : #SYSTEM_DATE
    P_Date : sydate,
as select from ...
{
```

```
}
```

For more information about consumption annotations, see section *Consumption Annotations* linked below.

❖ Example

In the following example, the field `buyer_guid` is required by the association to `_BusinessPartner` only. This means, the field must be included in the element list of the CDS view, but must **not** be transferred to the client.

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so
  association [0..1] to sepm_cds_business_partner as _BusinessPartner
  on $projection.buyer_guid = _BusinessPartner.business_partner_key
{
  key so.sales_order_id as SalesOrder,
  @Consumption.hidden: true
  so.buyer_guid,
  ...
  _BusinessPartner
}
```

There may be cases, where a field is needed in the client, for example for calculations, but should not be displayed directly in a list or table, or on an object-page floorplan. In this case the annotation is not suitable.

Related Information

[Consumption Annotations \[page 390\]](#)

[Field Manipulation \[page 333\]](#)

[Multi-Line Text \[page 333\]](#)

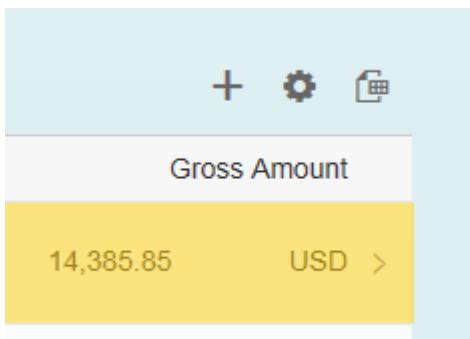
[Field Masking \[page 334\]](#)

[Interaction with Other Annotations \[page 336\]](#)

8.12.10.4 Interaction with Other Annotations

Get information about how to provide interaction between annotations.

In addition to UI annotation, you can use model-specific annotations that affect the desired client behavior. You can implement, for example, unit-currency-mappings, ID-text-mappings, or properties for field control that are represented by model-specific annotations. These model-specific annotations can be evaluated by the client and **no** additional UI annotations are required.



Example of interaction between CDS annotations

• Example

In the following example, the field `CurrencyCode` is marked with a [@Semantics.currencyCode \[page 456\]](#) and is referenced by field `GrossAmount`. This means that the field `GrossAmount` is always displayed with the corresponding currency. The field `CurrencyCode` does not need to be exposed explicitly.

Furthermore the field `GrossAmount` is marked as being mandatory. This means that the field is treated accordingly by the client: The field is marked with an asterisk, and if users do not fill in a value for this property, an error is raised. The administrative fields such as `CreatedAt` and `CreatedBy`, are set in a back-end validation and must **not** be changed by the client. For this reason, these fields are marked as being read-only.

↳ Sample Code

```
...
define view ZExample_SalesOrder as select from sepm_cds_sales_order as so
{
  @UI.identification: [ { position: 10 } ]
  key so.sales_order_id as SalesOrder,
  @Semantics.currencyCode: true
  so.currency_code as CurrencyCode,

  @Semantics.amount.currencyCode: 'CurrencyCode'
  @ObjectModel.mandatory: true
  @UI.identification: [ { position: '20' } ]
  so.gross_amount as GrossAmount,
  ...
  @ObjectModel.readOnly: true
  @UI.identification: [ { position: '110' } ]
  so.created_at as CreatedAt,
  ...
  @ObjectModel.readOnly: true
  @UI.identification: [ { position: '120' } ]
  so.created_by as CreatedBy,
  ...
  @ObjectModel.readOnly: true
  @UI.identification: [ { position: '130' } ]
  so.changed_at as ChangedAt,
  ...
  @ObjectModel.readOnly: true
  @UI.identification: [ { position: '140' } ]
  so.changed_by as ChangedBy,
}
```

Related Information

[Field Manipulation \[page 333\]](#)

[Multi-Line Text \[page 333\]](#)

[Field Masking \[page 334\]](#)

[Field Hiding \[page 335\]](#)

8.12.10.5 Inheritance of Annotations

Get information about what property to use to prevent elements from being inherited from an underlying CDS view.

By default, all UI annotation elements are inherited from the underlying CDS view. You can explicitly disable this behavior. You can use the following property to prevent a UI annotation element from being inherited:

- *exclude*

The following sample code depicts the CDS view `ZP_SalesOrder` that inherits elements from the underlying CDS view `SEPM_CDS_SALES_ORDER`, and uses the UI annotation :

↳ Sample Code

```
...
define view ZP_SalesOrder as select from sepm_cds_sales_order as so {
    @UI.identification: [ { position: 10 } ]
    key so.sales_order_id as SalesOrder,
    @UI.identification: [ { position: 20 } ]
    so.customer.company_name as CompanyName,
}
...
```

The following sample code depicts the CDS view `ZI_SalesOrder` that inherits elements from the underlying CDS view `ZP_SalesOrder`. In this view, the element `key SalesOrder` is inherited from the underlying CDS view as UI annotation by default. The element `so.customer.company_name as CompanyName`, however, is not inherited as UI annotation because of the property `exclude`:

↳ Sample Code

```
...
define view ZI_SalesOrder as select from ZP_SalesOrder as so {
    key SalesOrder,
    @UI.identification: [ { exclude } ]
    so.customer.company_name as CompanyName,
}
...
```

8.13 Exposing CDS Entities as OData Service

You have different options to generate a CDS view as OData service:

Options to Generate OData Services

Option	Use Case	Tool	More Information
Auto-Exposure	<p>Let us assume that you created a quite elementary data model based on multiple CDS views. All these views together form a quite simple composition: One CDS view serves as root and the other CDS entities are children of your root CDS view. The CDS views of this composition might also have associations to other entities.</p> <p>With this option, a simple way of creating OData services has been introduced. Here, the OData model definition as well as the OData service runtime is provided generically and with low manual effort.</p>	ABAP Development Tools	Generating OData Service With Auto-Exposure [page 19]

Use this option if:

- You want to expose such an elementary CDS data model as an OData service, together with first the level of associations.

Do not use this option if:

- Your data model composition is more complex and/or you need to include deeper association levels in your OData service.
- You want to create a hybrid scenario where implementations are based on CDS views and on custom logic.

Option	Use Case	Tool	More Information
Referenced Data Source (RDS)	<p>Use Case 1: You created several CDS views and complex CDS view compositions. You need to include in the OData service several levels of associations from CDS views to other entities.</p> <p>Use Case 2: You created a CDS view that contains features that are not supported by CDS or complex CDS view compositions. You need to implement these features manually in the model provider extension class (MPC_EXT class) or in the data provider extension class of your <i>Service Builder</i> project. To provide, for example, dynamic field control, you need to implement custom logic in a model provider extension class.</p> <p>Use this option if:</p> <ul style="list-style-type: none"> The CDS model and the CDS annotations fully specify the OData Model and the runtime behavior Your data model is based on CDS, but is more complex than simple compositions and includes advanced features, such as transactional processing You want to modify the service with custom code implementations. <p>Do not use this option if:</p> <ul style="list-style-type: none"> Your data model is not only based on CDS. 	Transaction SEGW	
Mapped Data Source (MDS)	<p>You modeled an entity set in transaction SEGW from a DDIC structure. You want to map fields of a CDS view to your custom model. You furthermore want to reference from your custom model to another CDS view, for example, and therefore need to map the two different entity sets.</p> <p>⚠ Caution</p> <p>Future changes of the entities might invalidate the service runtime.</p> <p>Use this option if:</p> <ul style="list-style-type: none"> Your data model includes non-CDS entities RDS is not available for your release version. <p>Do not use this option if:</p> <ul style="list-style-type: none"> One of the other options is applicable. 	Transaction SEGW	Generating an OData Service Using the Mapping Editor [page 350]

8.13.1 Generating OData Service With Auto-Exposure

OData Exposure

With the concept of auto-exposure, a new and simple way of creating OData services has been introduced. Here, the OData model definition as well as the OData service runtime is provided generically, based on *SADL* (*Service Adaptation Description Language*).

The requirement here is that the annotation `@OData.publish: true` is specified at the CDS data model (CDS consumption view) level as follows:

```
@AbapCatalog.sqlViewName: 'SQL_VIEW_SAMPLE'  
...  
@OData.publish: true  
define view CDS_VIEW_NAME as select from  
...  
}
```

→ Remember

We recommend using the **auto-exposure** option in the case of elementary data model compositions: for example, if you defined the entire CDS data model based on a root CDS view so that several other CDS views are children of this root CDS view. In addition, the CDS views of this composition might have associations to other entities. In cases like this, all the CDS views together form a quite simple data model composition that you want to expose as an OData service, together with first the level of associations.

Do not use the auto-exposure option if:

- Your data model composition is more complex and you need to include deeper association levels in your OData service.
- You want to generate a hybrid scenario where implementations are based on CDS views and on custom logic.

More on this: [Exposing CDS Entities as OData Service \[page 339\]](#)

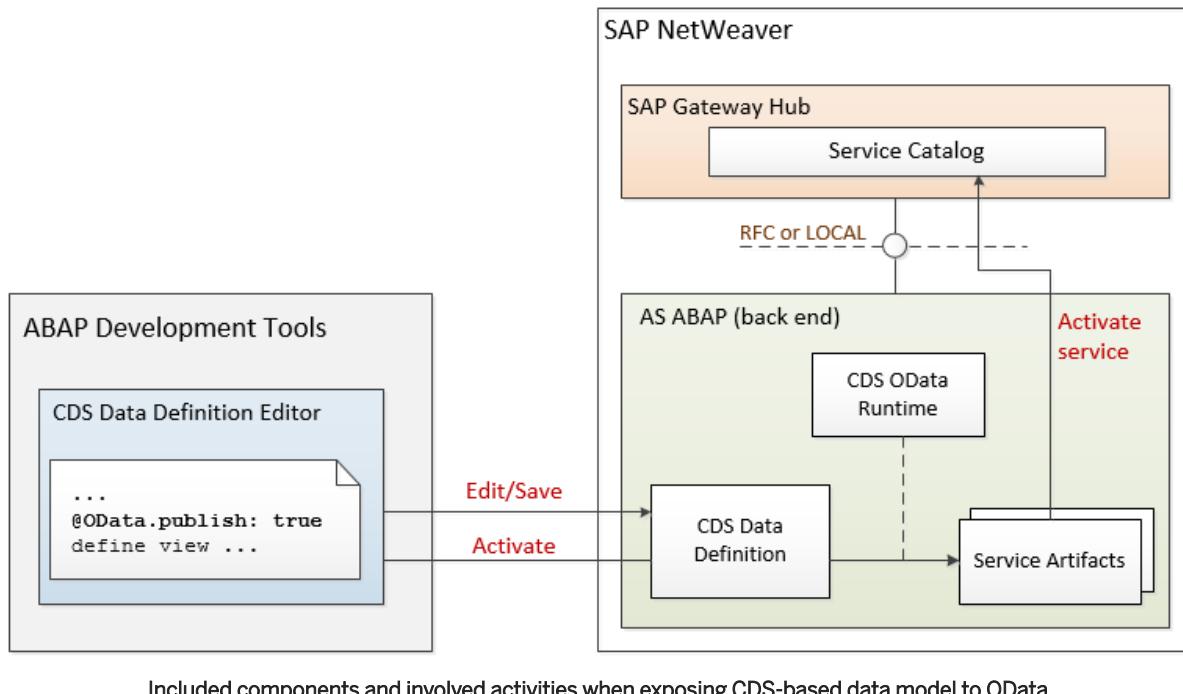
The Auto-Exposure Process

The following figure depicts the main components of the auto-exposure process and refers to the most important activities that are involved:

Starting in *ABAP Development Tools*, you open the relevant CDS data definition object where the CDS view in question is implemented. After you have added the OData annotation to the CDS view, you can trigger the activation of the entire CDS data definition (that serves as transportable development object). *ABAP Development Tools* delegates the activation request to *SADL*. *SADL* framework generates several SAP Gateway artifacts that are stored in the back end of the application server AS ABAP and are required for OData service activation in the SAP Gateway hub system.

In the next step, you need to launch the transaction `/IWFND/MAINT_SERVICE` to add the OData service to the service catalog of the SAP Gateway hub. In this way, the actual OData service is created in the SAP Gateway and the back end system is assigned as a data source using the system alias.

As soon as the OData service is activated in the SAP Gateway hub, it is ready for consumption through an OData client, such as an SAP Fiori app.



Developer-Relevant Tasks

Task 1: Generate Service Artifacts From a CDS View [page 21]

As a result of this task, several service artifacts are generated in the back end of the application server.

Task 2: Activate OData Service in the SAP Gateway Hub [page 22]

As a result of this task, the OData service is added to service catalog of the SAP Gateway hub and is ready for consumption.

Task 3: Test the Activated OData Service [page 26]

8.13.2 Creating an OData Service based on a Referenced Data Source (RDS)

Use Case

You want to generate an OData services with a CDS entity (CDS view) as a referenced data source. The CDS model and the CDS annotations fully specify the OData Model and the runtime behavior, without any additional

or redundant metadata or OData-specific implementation. This means that the CDS View, including all extensions, and all applicable annotations are mapped to a corresponding OData service.

This tool supported approach allows you more flexibility concerning the creation of an OData Service: In contrast to the auto-exposure, you can add additional entities or make structure changes or manipulate data, if necessary, to adapt the OData Service to your specific requirements. This approach is advisable if you want to use custom code to change the OData Service instead of having the OData Service and the runtime artifacts autogenerated.

Prerequisites

Before you start, take note of the following prerequisites:

- Knowledge
 - You require advanced knowledge of the SAP Gateway Service Builder and OData.
 - You require basic knowledge of the Enterprise Procurement Model (EPM).
- Systems and Releases
 - This function is provided as of Netweaver Release 7.5 SP00.
- Authorizations
 - The user needs access and the appropriate authorization in order to run the SAP Gateway Service Builder.

8.13.2.1 Creating a Project

Projects are used to store the artifacts that developers need to create a service and its underlying data model in a single place in the [SAP Gateway Service Builder](#).

Process Steps

1. Enter transaction code [SEGW](#) to access the [SAP Gateway Service Builder](#).
2. Click  [Create Project](#) to open the [Create Project](#) dialog.
3. Enter the following information in the dialog box:
 1. [Project](#): Choose a technical name for your project.
 2. [Description](#): Choose a project description.
 3. [Project Type](#): Choose [Service with SAP Annotations](#) for an OData service that uses standard annotations and SAP annotations.

i Note

The other options are not relevant if you want to create an Odata Service based on a referenced data source.

4. Confirm to create the project.

Result

- The project is added automatically in the *SAP Service Builder*.
- You can now continue to define a data model for your project.

Related Information

[Importing a Referenced Data Source \[page 344\]](#)

[Generating Runtime Artifacts \[page 345\]](#)

[Registering an OData Service \[page 348\]](#)

8.13.2.2 Importing a Referenced Data Source

You can use the data model defined in an CDS entity created earlier, where the CDS model and the CDS annotations fully specify the OData model and the runtime behavior as a referenced data source. You can adapt and change the imported data model, for example by adding associations or further CDS entities.

Process Steps

1. Enter transaction code *SEGW* to access the *SAP Gateway Service Builder*.
2. Create a project as described in [Creating a Project \[page 343\]](#).
3. Right-click on the *Data Model* folder and choose .
4. Enter the name of the CDS entity you want to use as referenced data source in the *CDS Entity* field and choose *Next*.
5. The second page of the wizard is opened and the data model of the selected CDS entity is displayed. You can choose which entities and associations you want to expose with your OData Service.

i Note

When you select an association, the target entity is automatically selected as well. If you deselect the association, you can select the target entities separately.

6. After you choose *Finish*, the selected data model is displayed below in the *Data Source References* node under Exposure via SSDL (CDS Entity Exposures) as depicted below:

CDS-Entity Exposures	Selected	Value Help	Analytical	Parameter	Status
SEPML_Employee	<input checked="" type="checkbox"/>				
SEPML_Company	<input checked="" type="checkbox"/>				
SEPML_EmployeeOrgAssignment	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_EmployeeOrgAssignment	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_GenderCode	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_GenderCode	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_Address	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_LeaveRequest	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_LeaveRequest	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_PreferredLanguage	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_Language	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_SalaryCurrency	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_Currency	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_SalesOrder	<input checked="" type="checkbox"/>				
SEPML_BillingStatus	<input checked="" type="checkbox"/>				
SEPML_SalesOrderBillingStatus	<input checked="" type="checkbox"/>				
SEPML_BillToPartyAddress	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_PartyAddress	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_CreatedByUser	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_Employee	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Customer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SEPML_BusinessPartner	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CustomerContact	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

7. Here you can adapt your data model or add additional CDS entities as data sources. You can display the source entity, the association, and the target entity of each listed entity to get more details.

i Note

By default, only the first level of association is listed in the *CDS Entity Exposure* column. If you want to add more levels to the model, right-click the required entity and select *Add CDS Entity*.

8. You can now generate your runtime artifacts.

Result

You have imported the *Referenced Data Source* and selected the entities you want to expose. You can now generate the *Runtime Artifacts*.

Related Information

[Generating Runtime Artifacts \[page 345\]](#)

8.13.2.3 Generating Runtime Artifacts

You must generate the *Runtime Artifacts* after you have defined the metadata for them. When you generate the *Runtime Artifacts* in the SAP Service Builder, it automatically generates the code for making your service OData-compliant and ready for use in SAP Gateway. In addition, the Service Builder automatically configures the project by assigning the data model to the generated service for use at runtime.

The following occurs when you generate the service:

- The implementation classes for the data model and the service are automatically created.
- The OData service is automatically registered in SAP Gateway.

Process Steps

1. Enter transaction code **SEGW** to access the *SAP Gateway Service Builder*.
2. Create a project as described in [Creating a Project \[page 343\]](#)
3. Define your data model.
4. Click  to generate the *Runtime Artifacts* for your project. The OData service is registered in the back end and the *Model Provider Classes* and the *Data Provider Classes* are automatically generated in the *Runtime Artifacts* folder of your project.

i Note

If a project is generated for the first time, you have to specify the *Technical Model Name* and the *Technical Service Name*. The technical service name is used when the OData service is published and cannot be changed after that.

Result

As a result, the following runtime artifacts are generated:

Runtime Artifact	Description
<i>Project_Name_ANNO_MDL</i>	This class contains the annotation model.
<i>Project_Name_MDL</i>	The technical model name that is used to register the service in the SAP Business Suite system and is automatically generated.
<i>Project_Name_SRV</i>	The technical service name that is used to register the service in the SAP Business Suite system and is automatically created.
	The technical service name becomes the external service name that is later used for publishing the service on the SAP NetWeaver Gateway.

Runtime Artifact	Description
Z_CL_<Project Name>_RDS_DPC	Generating the <i>Runtime Artifacts</i> redefines and implements the following operations in the data model provider base class: Create, Read, Update, Delete (CRUD), and Query. The class is generated only when the Service Builder successfully generates the code for the classes of the Model Provider Class (MPC).
Z_CL_<Project Name>_RDS_DPC_EXT	The data provider implementation class inherits its properties from the data provider base class. You can redefine each method in the implementation class for the data provider class.
Z_CL_<Project Name>_RDS_MPC	The model provider base class contains redefinitions for the DEFINE() and GET_LAST_MODIFIED() methods.
Z_CL_<Project Name>_RDS_MPC_EXT	The model provider implementation class inherits its properties from the model provider base class.

Related Information

[Model Provider Classes \[page 347\]](#)

[Data Provider Classes \[page 348\]](#)

[Registering an OData Service \[page 348\]](#)

8.13.2.3.1 Model Provider Classes

The model provider contains the ABAP code that defines the data model of the OData Service at runtime.

Model Provider Base Class

- ZCL_<Project Name>_MPC: The model provider base class is derived from the superclass */IWBEPE/CL_MGW_PUSH_ABS_MODEL*. and contains redefinitions for the DEFINE() and GET_LAST_MODIFIED() methods.
For more information about the model provider base class and the methods it contains, see [Base Class: Model Provider Class](#)

Model Provider Implementation Class

- ZCL_*Project_Name*_MPC_EXT:

This extension class is inherited from the model provider base class and inherits its properties. The model provider extension class is registered in the back end using the technical service name. In this class, you can choose which methods of the base class you want to keep and which methods you want to redefine to suit your requirements. For more information, about the model provider implementation class, see [Implementation Class: Model Provider Class](#)

8.13.2.3.2 Data Provider Classes

The *data provider classes* contain the OData service operation implementation.

Data Provider Base Class

- ZCL_*Project_Name*_DPC: Generating the *runtime artifacts* redefines and implements the following operations: Create, Read, Update, Delete (CRUD), and Query. The class is generated only when the Service Builder successfully generates the code for the classes of the model provider class (MPC). The DPC base class inherits from the generic runtime class `/IWBEPC/CL_MGW_PUSH_ABS_DATA`. For more details about the *data provider base class*, see [Base Class: Data Provider Class](#).

Data Provider Implementation Class

- ZCL_*Project_Name*_DPC_EXT: This implementation class inherits its properties from *data provider base class*. You can redefine each method in the implementation class for the *data provider class*. For more details about how to redefine methods, see [Redefining Methods of the Operations](#)

8.13.2.4 Registering an OData Service

You need to register your OData Service for use in each system.

Prerequisites

The SAP Gateway systems in which you want to register the OData service must be configured under the following path: [SPRO](#) [SAP NetWeaver](#) [SAP Gateway Service Enablement](#) [Backend OData Channel](#) [Connection Settings to SAP Gateway](#) [SAP Gateway Settings](#). This is done to assign the RFC destination to the specific SAP Gateway system you want to use with the SAP Service Builder..

Process Steps

1. Enter transaction code [SEGW](#) to access the *SAP Gateway Service Builder*.
2. Create a project as described in [Creating a Project \[page 343\]](#).
3. Define your data model.
4. Generate your *Runtime Artifacts* as described in [Generating Runtime Artifacts \[page 345\]](#).
5. Expand the *Service Maintenance* folder in your project. This list contains all configured systems for which you can register your OData service.

→ Tip

You can double-click to view the configured systems in the mass maintenance view to check whether the Odata Service has already been registered for your target system.

6. Right-click on a system for which you want to register the OData service and select *Register*. The *Select System Alias* window is displayed if there is more than one system that points to the system from which you are registering the OData service. Select your target system by using the input help available for this field.
7. Click *Continue* to display the *Add Service* window. The details about the selected target system are displayed. Choose a package and click *Continue*.

Result

The OData service is now registered for the system in question and the registration status turns green.

8.13.2.5 Testing an OData Service in the SAP Gateway Client

You can use the *SAP Gateway Client* to test your OData Service and see if it works as expected.

Process Steps

1. Enter transaction code [SEGW](#) to access the *SAP Gateway Service Builder*.
2. Create a project as described in [Creating a Project \[page 343\]](#).
3. Define your data model.
4. Generate the *Runtime Artifacts* as described in [Generating Runtime Artifacts \[page 345\]](#).
5. Register the OData Service as described in [Registering an OData Service \[page 348\]](#).
6. Click [SAP Gateway Client](#) to launch the SAP Gateway Client to test your OData service.
7. The *SAP Gateway Client* is opened with a prefilled request URL.

→ Tip

If you want to know more about the SAP Gateway Client, see [SAP Gateway Client](#).

8.13.3 Generating an OData Service Using the Mapping Editor

SADL offers a mapping editor in the SAP NetWeaver Gateway Service Builder to bind one or several SADL models to OData entity sets. This enables you to use SADL models as an additional data source besides RFC and others. Once the OData properties are mapped, the system provides a standard, optimized implementation of the OData service to retrieve the data.

Prerequisites

- You are familiar with working with SAP NetWeaver Gateway Service Builder. For information about the Gateway Service Builder, see SAP Help Portal at help.sap.com at ► *Technology* ► *SAP NetWeaver Platform* ► <Release> ► *Application Help* ► *Function-Oriented View* ► *SAP NetWeaver Gateway Foundation (SAP_GWFND)* ► *SAP NetWeaver Gateway Foundation Developer Guide* ► *SAP NetWeaver Gateway Service Builder* ▶.
- You have access to the back-end system and are authorized to log on to this system and to generate and register OData Services. For information about required authorizations, see SAP Help Portal at help.sap.com at ► *Technology* ► *SAP NetWeaver Platform* ► <Release> ► *Application Help* ► *Function-Oriented View* ► *SAP NetWeaver Gateway Foundation (SAP_GWFND)* ► *SAP NetWeaver Gateway Configuration Guide* ► *OData Channel Configuration* ► *User, Developer, and Administrator Authorizations* ▶.

Procedure

1. In SAP NetWeaver Gateway Service Builder, create a project and define the OData model.
2. Open the *Service Implementation* node, right-click on an entity set and choose *Map to Data Source*.
When you implement a SADL-based OData service, you map the complete entity set. This differs from the usual procedure, where you map each operation (Create, Read, and so on) separately.
3. Choose the data source type and open the value help for the *Name* field.
4. Choose the SADL model by selecting a *SADL Model Source* and *SADL Model Name*.
The options you see here, depend on what is available in the system. Usually you should see DDIC and CDS.
5. To map the entities, associations and actions, you have the following options:
 - You can switch between the *Property Editor*, *Association Editor*, and the *Action Editor*. From the editor you are currently working in, you can switch to the other two possible editors.
 - Generate a mapping proposal by choosing *Generate Mapping*. This compares the ABAP field names of the properties with the names in the SADL entity tree. Upper and lower case as well as underscores are ignored. Only the highest level of the entity tree is compared. If both values are the same, you will see that the SADL entity name has been entered in the *Element* column of the mapping table. You have to generate the mapping for entities and associations separately.
 - Create the mapping by dragging the elements from the SADL tree and dropping them on the corresponding cell in the *Elements* column of the mapping table.

6. Generate the runtime artifacts and register the service.

Related Information

[Model a Gateway Service based on Business Entities through SADL](#) 

8.13.3.1 Creating a Project

Projects are used to store the artifacts that developers need to create a service and its underlying data model in a single place in the [SAP Gateway Service Builder](#).

Process Steps

1. Enter transaction code [SEGW](#) to access the [SAP Gateway Service Builder](#).
2. Click  [Create Project](#) to open the [Create Project](#) dialog.
3. Enter the following information in the dialog box:
 1. [Project](#): Choose a technical name for your project.
 2. [Description](#): Choose a project description.
 3. [Project Type](#): Choose [Service with SAP Annotations](#) for an OData service that uses standard annotations and SAP annotations.

i Note

The other options are not relevant if you want to create an Odata Service based on a referenced data source.

4. Confirm to create the project.

Result

- The project is added automatically in the [SAP Service Builder](#).
- You can now continue to define a data model for your project.

Related Information

[Importing a Referenced Data Source \[page 344\]](#)

[Generating Runtime Artifacts \[page 345\]](#)

8.13.3.2 Mapping a Data Source

Mapping is a manual relation that you establish between the parameters of a data source object (in an SAP backend system) and the properties of an entity set in the Service Builder. When you implement a SADL-based OData service, you map the complete entity set in contrast to the usual procedure, where you map each operation (Create, Read, and so on) separately.

Process Steps

1. Enter transaction code [SEGW](#) to access the *SAP Gateway Service Builder*.
2. Create a project as described in [Creating a Project \[page 343\]](#)
3. Define your Odata model.
4. Open the *Service Implementation* node, right-click on an entity set and choose [Map to Data Source](#).

i Note

When you implement a SADL-based OData service, you map the complete entity set. This differs from the usual procedure, where you map each operation (Create, Read, and so on) separately.

5. Choose the data source type and open the value help for the *Name* field.
6. Choose the SADL model by selecting a [SADL Model Source](#) and [SADL Model Name](#).

i Note

The options you see here, depend on what is available in the system. Usually you should see DDIC and CDS

7. To map the entities, associations and actions, you have the following options:
 - You can switch between the *Property Editor*, *Association Editor*, and the *Action Editor*. From the editor you are currently working in, you can switch to the other two possible editors.
 - Generate a mapping proposal by choosing [Generate Mapping](#). This compares the ABAP field names of the properties with the names in the SADL entity tree. Upper and lower case as well as underscores are ignored. Only the highest level of the entity tree is compared. If both values are the same, you will see that the SADL entity name has been entered in the *Element* column of the mapping table. You have to generate the mapping for entities and associations separately.
 - Create the mapping by dragging the elements from the SADL tree and dropping them on the corresponding cell in the *Elements* column of the mapping table.
8. Generate your [Runtime Artifacts](#) as described in [Generating Runtime Artifacts \[page 345\]](#).

Result

You have mapped your entity set and thus bound one or several SSDL models to OData entity sets. You can now generate your *Runtime Artifacts* and register your OData service.

Related Information

[Generating Runtime Artifacts \[page 345\]](#)

[Registering an OData Service \[page 348\]](#)

[Testing an OData Service in the SAP Gateway Client \[page 349\]](#)

8.13.3.3 Generating Runtime Artifacts

You must generate the *Runtime Artifacts* after you have defined the metadata for them. When you generate the *Runtime Artifacts* in the SAP Service Builder, it automatically generates the code for making your service OData-compliant and ready for use in SAP Gateway. In addition, the Service Builder automatically configures the project by assigning the data model to the generated service for use at runtime.

The following occurs when you generate the service:

- The implementation classes for the data model and the service are automatically created.
- The OData service is automatically registered in SAP Gateway.

Process Steps

1. Enter transaction code **SEGW** to access the *SAP Gateway Service Builder*.
2. Create a project as described in [Creating a Project \[page 343\]](#)
3. Define your data model.

4. Click  to generate the *Runtime Artifacts* for your project. The OData service is registered in the backend and the *Model Provider Classes* and the *Data Provider Classes* are automatically generated in the *Runtime Artifacts* folder of your project.

i Note

If a project is generated for the first time, you have to specify the *Technical Model Name* and the *Technical Service Name*. The technical service name is used when the OData service is published and cannot be changed after that.

Result

As a result, the following runtime artifacts are generated:

Runtime Artifact	Description
<code>Project_Name_ANNO_MDL</code>	This class contains the annotation model.
<code>Project_Name_MDL</code>	The technical model name that is used to register the service in the SAP Business Suite system and is automatically generated.
<code>Project_Name_SRV</code>	The technical service name that is used to register the service in the SAP Business Suite system and is automatically created.
	The technical service name becomes the external service name that is later used for publishing the service on the SAP NetWeaver Gateway.
<code>Z_CL_Project_Name_RDS_DPC</code>	Generating the <i>Runtime Artifacts</i> redefines and implements the following operations in the data model provider base class: Create, Read, Update, Delete (CRUD), and Query. The class is generated only when the Service Builder successfully generates the code for the classes of the Model Provider Class (MPC).
<code>Z_CL_Project_Name_RDS_DPC_EXT</code>	The data provider implementation class inherits its properties from the data provider base class. You can redefine each method in the implementation class for the data provider class.
<code>Z_CL_Project_Name_RDS_MPC</code>	The model provider base class contains redefinitions for the DEFINE() and GET_LAST_MODIFIED() methods.
<code>Z_CL_Project_Name_RDS_MPC_EXT</code>	The model provider implementation class inherits its properties from the model provider base class.

Related Information

[Model Provider Classes \[page 347\]](#)

[Data Provider Classes \[page 348\]](#)

[Registering an OData Service \[page 348\]](#)

8.13.3.3.1 Model Provider Classes

The model provider contains the ABAP code that defines the data model of the OData Service at runtime.

Model Provider Base Class

- ZCL_*Project_Name*_MPC: The model provider base class is derived from the superclass `/IWBEPC/CL_MGW_PUSH_ABS_MODEL`. and contains redefinitions for the DEFINE() and GET_LAST_MODIFIED() methods.

For more information about the model provider base class and the methods it contains, see [Base Class: Model Provider Class](#)

Model Provider Implementation Class

- ZCL_*Project_Name*_MPC_EXT:
This extension class is inherited from the model provider base class and inherits its properties. The model provider extension class is registered in the back end using the technical service name. In this class, you can choose which methods of the base class you want to keep and which methods you want to redefine to suit your requirements. For more information, about the model provider implementation class, see [Implementation Class: Model Provider Class](#)

8.13.3.3.2 Data Provider Classes

The *data provider classes* contain the OData service operation implementation.

Data Provider Base Class

- ZCL_*Project_Name*_DPC: Generating the *runtime artifacts* redefines and implements the following operations: Create, Read, Update, Delete (CRUD), and Query. The class is generated only when the Service Builder successfully generates the code for the classes of the model provider class (MPC). The DPC base class inherits from the generic runtime class `/IWBEPC/CL_MGW_PUSH_ABS_DATA`. For more details about the *data provider base class*, see [Base Class: Data Provider Class](#).

Data Provider Implementation Class

- ZCL_*Project_Name*_DPC_EXT: This implementation class inherits its properties from *data provider base class*. You can redefine each method in the implementation class for the *data provider class*. For more details about how to redefine methods, see [Redefining Methods of the Operations](#)

8.13.3.4 Registering an OData Service

You need to register your OData Service for use in each system.

Prerequisites

The SAP Gateway systems in which you want to register the OData service must be configured under the following path: ► *SPRO* ► *SAP NetWeaver* ► *SAP Gateway Service Enablement* ► *Backend OData Channel* ► *Connection Settings to SAP Gateway* ► *SAP Gateway Settings* ▶. This is done to assign the RFC destination to the specific SAP Gateway system you want to use with the SAP Service Builder..

Process Steps

1. Enter transaction code *SEGW* to access the *SAP Gateway Service Builder*.
2. Create a project as described in [Creating a Project \[page 343\]](#).
3. Define your data model.
4. Generate your *Runtime Artifacts* as described in [Generating Runtime Artifacts \[page 345\]](#).
5. Expand the *Service Maintenance* folder in your project. This list contains all configured systems for which you can register your OData service.

→ Tip

You can double-click to view the configured systems in the mass maintenance view to check whether the Odata Service has already been registered for your target system.

6. Right-click on a system for which you want to register the OData service and select *Register*. The *Select System Alias* window is displayed if there is more than one system that points to the system from which you are registering the OData service. Select your target system by using the input help available for this field.
7. Click *Continue* to display the *Add Service* window. The details about the selected target system are displayed. Choose a package and click *Continue*.

Result

The OData service is now registered for the system in question and the registration status turns green.

8.13.3.5 Testing an OData Service in the SAP Gateway Client

You can use the [SAP Gateway Client](#) to test your OData Service and see if it works as expected.

Process Steps

1. Enter transaction code [SEGW](#) to access the [SAP Gateway Service Builder](#).
2. Create a project as described in [Creating a Project \[page 343\]](#).
3. Define your data model.
4. Generate the [Runtime Artifacts](#) as described in [Generating Runtime Artifacts \[page 345\]](#).
5. Register the OData Service as described in [Registering an OData Service \[page 348\]](#).
6. Click [SAP Gateway Client](#) to launch the SAP Gateway Client to test your OData service.
7. The [SAP Gateway Client](#) is opened with a prefilled request URL.

→ Tip

If you want to know more about the SAP Gateway Client, see [SAP Gateway Client](#).

8.13.4 Exposing CDS Entities with Parameters

It is possible to expose CDS views with **parameters**. Parameters can be used for calculations and data selections within a CDS view. Parameter values must be supplied for each DB query. Using specific annotations, you can decide whether values for parameters should be provided by OData consumers (**visible parameter**) or filled internally by the ABAP server, and thus cannot be influenced by OData consumers (**hidden parameters**). Visible and hidden parameters can also be combined in a single CDS view.

Visible Parameters

The following is an example of a CDS view with a Currency parameter, which has to be supplied by the OData consumer to receive amounts converted to a required currency:

```
define view SalesOrdersWithConvAmounts
  with parameters
    currency : snwd_curr_code
    as select from snwd_so as SalesOrder
{
```

```
...  
}
```

In the case of views with visible parameters, the mapping of CDS artifacts to the OData metamodel is different, since the parameter values need to be supplied before executing any OData queries. As a consequence, a CDS view with parameters has to be mapped to two OData entity sets - parameter and result entity set. Let's suppose the name of the CDS view is '**CDSView**'. The OData artifact names would be as follows:

OData	Name
Result Entity Name	CDSViewType
Result Entity Set Name	CDSViewSet
Parameter Entity Name	CDSViewParameters
Parameter Entity Set Name	CDSView
OData Association	<Generated ID>
OData Association Set	<Generated ID>
Navigation Property	Set

i Note

A DDIC data element must be used to declare the type of visible parameters.

i Note

CDS views with visible parameters must not expose any associations and must not be a target view of an exposed association.

i Note

If the CDS view is analytical, a different naming convention is used for the generated OData artifacts.

Hidden Parameters

Parameter annotations can be used to specify that a parameter should be hidden and thus not visible for OData consumers. The system fills the parameters from system variables. The annotation looks as follows:

```
define view Sadl_V_Parameter_Syst  
with parameters  
  @Consumption.hidden: true  
  @Environment.systemField: #Application_User  
  p_user: syuname  
as select from ...
```

Here, the view parameter `p_user` is not exposed. The parameter is bound to the system variable `sy-uname` at runtime.

Options for the `@Environment.systemField` annotation values are as follows:

Value for <code>@Environment.systemField</code>	Binding to system variable
#CLIENT	SY-MANDT
#SYSTEM_LANGUAGE	SY-LANGU
#APPLICATION_USER	SY-UNAME
#SYSTEM_DATE	SY-DATUM
#SYSTEM_TIME	SY-UZEIT

i Note

If one of the parameter annotations `@Environment.systemField` or `@Consumption.hidden:true` is missing, the parameter becomes visible. A parameter for a system client (`systemField:#CLIENT`) must however only be used in combination with `hidden:true`.

Analytical Scenarios

CDS also covers **analytical scenarios**, where measures and dimensions can be defined. The runtime of these CDS views is provided by the Analytical Engine or by SADL, depending on the complexity of the view.

i Note

A CDS view is considered to be an **analytical view** if it contains at least one occurrence of the `@DefaultAggregation` annotation.

For more information about analytical entities, see [Using Aggregate Data in SAP Fiori Apps \[page 263\]](#).

The following is an example of a CDS view with a Currency parameter, which has to be supplied by the OData consumer to receive amounts converted to a required currency. This view also has additional analytical annotations.

```
define view SalesOrdersWithConvAmounts
  with parameters
    currency : snwd_curr_code
    as select from snwd_so as SalesOrder
{
  ...
  @DefaultAggregation: #SUM
  price
  ...
}
```

In the case of views with visible parameters, the mapping of CDS artifacts to the OData metamodel is different, since the parameter values do not need to be supplied before executing an OData query. As a consequence, a CDS view with parameters and analytical annotations has to be mapped to two OData entity sets - parameter

and result entity set. Let's suppose the name of the CDS view is '**CDSView**'. The OData artifact names would be as follows:

OData	Name
Result Entity Name	CDSViewResult
Result Entity Set Name	CDSViewResults
Parameter Entity Name	CDSViewParameters
Parameter Entity Set Name	CDSView
OData Association	
OData Association Set	
Navigation Property	Result

9 Reference

9.1 CDS Annotations

The following list summarizes all SAP annotations of the Data Definition Language (DDL) of ABAP CDS.

SAP CDS annotations are evaluated by SAP frameworks and can be either ABAP annotations or framework-specific annotations.

ABAP CDS - ABAP Annotations

CDS annotations that are evaluated by **ABAP runtime**:

- AbapCatalog Annotations
- AccessControl Annotations
- ClientDependent Annotations (obsolete as of SAP NetWeaver AS for ABAP 7.51 innovation package SP00)
- ClientHandling Annotations
- DataAging Annotations
- EndUserText Annotations
- Environment Annotations
- MappingRole Annotations
- Metadata Annotations
- Semantics Annotations

→ Tip

To access help for an ABAP annotation, position the cursor on the relevant annotation in the DDL editor and choose **F1**.

Framework-Specific Annotations

CDS annotations that (as a rule) are evaluated during runtime by specific frameworks such as **SADL**, **BOPF**, **Analytics**, or **Enterprise Search**:

- [AccessControl Annotations \[page 362\]](#)
- [Aggregation Annotations \[page 414\]](#)
- [Analytics Annotations \[page 367\]](#)
- [AnalyticsDetails Annotations \[page 379\]](#)
- [Consumption Annotations \[page 390\]](#)

- EnterpriseSearch Annotations [page 420]
- Hierarchy Annotations [page 422]
- ObjectModel Annotations [page 428]
- OData Annotations [page 441]
- Search Annotations [page 442]
- Semantics Annotations [page 445]
- UI Annotations [page 465]
- VDM Annotations [page 587]

Related Information

[ABAP CDS - Anotations \(ABAP Keyword Documentation\)](#)

[ABAP CDS - ABAP Annotations \(ABAP Keyword Documentation\)](#)

[ABAP CDS - Framework-Specific Annotations \(ABAP Keyword Documentation\)](#)

9.1.1 AccessControl Annotations

Enable application developers to define how the authorization check for a CDS entity is executed

Scope and Definition

```
@Scope:[#VIEW, #TABLE_FUNCTION]
annotation AccessControl
{
    authorizationCheck : String(20) enum { NOT_REQUIRED; NOT_ALLOWED; CHECK;
PRIVILEGED_ONLY; } default #CHECK;
    privilegedAssociations: array of AssociationRef;
    personalData
    {
        blocking : String(30) enum { NOT_REQUIRED; REQUIRED;
BLOCKED_DATA_EXCLUDED; BLOCKED_DATA_INCLUDED; };
    };
}
```

Usage

Annotation	Meaning								
AccessControl.authorizationCheck	<p>This element defines the behavior of the authorization check.</p> <p>Scope: [#VIEW]</p> <p>Engine Behavior: The runtime and design-time engines handle the authorization check based on the value of the element.</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>#NOT_REQUIRED</td><td> <p>During the authorization runtime, an authorization check is executed if a DCL role exists for the entity. If no role exists, there's no check and no protection. This behavior is the same behavior at runtime as for value #CHECK. However, in this case it's intended by the developer that no role exists.</p> <p>During development, no warning occurs when activating the entity.</p> </td></tr> <tr> <td>#NOT_ALLOWED</td><td> <p>During the authorization runtime, no authorization check is executed.</p> <p>During development, a warning occurs if a developer activates a role for an entity, which has this annotation value.</p> </td></tr> <tr> <td>#CHECK</td><td> <p>During the authorization runtime, an authorization check is executed if a DCL role exists for the entity. If no role exists, there's no check and no protection.</p> <p>During development, a warning occurs if a developer activates the entity and no DCL role exists for the entity. This value is the default value.</p> </td></tr> </tbody> </table>	Value	Description	#NOT_REQUIRED	<p>During the authorization runtime, an authorization check is executed if a DCL role exists for the entity. If no role exists, there's no check and no protection. This behavior is the same behavior at runtime as for value #CHECK. However, in this case it's intended by the developer that no role exists.</p> <p>During development, no warning occurs when activating the entity.</p>	#NOT_ALLOWED	<p>During the authorization runtime, no authorization check is executed.</p> <p>During development, a warning occurs if a developer activates a role for an entity, which has this annotation value.</p>	#CHECK	<p>During the authorization runtime, an authorization check is executed if a DCL role exists for the entity. If no role exists, there's no check and no protection.</p> <p>During development, a warning occurs if a developer activates the entity and no DCL role exists for the entity. This value is the default value.</p>
Value	Description								
#NOT_REQUIRED	<p>During the authorization runtime, an authorization check is executed if a DCL role exists for the entity. If no role exists, there's no check and no protection. This behavior is the same behavior at runtime as for value #CHECK. However, in this case it's intended by the developer that no role exists.</p> <p>During development, no warning occurs when activating the entity.</p>								
#NOT_ALLOWED	<p>During the authorization runtime, no authorization check is executed.</p> <p>During development, a warning occurs if a developer activates a role for an entity, which has this annotation value.</p>								
#CHECK	<p>During the authorization runtime, an authorization check is executed if a DCL role exists for the entity. If no role exists, there's no check and no protection.</p> <p>During development, a warning occurs if a developer activates the entity and no DCL role exists for the entity. This value is the default value.</p>								
AccessControl.privilegedAssociations	<p>This element defines</p> <p>Scope: [#TABLE_FUNCTION], [#VIEW]</p> <p>Engine Behavior: The runtime and design-time engines handle the authorization check based on the value of the element.</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>associationRef</td><td>All associations that are exposed in the projection list and that are mentioned in the list defined in this annotation should allow privileged access to the association target. No DCL check happens at runtime</td></tr> </tbody> </table>	Value	Description	associationRef	All associations that are exposed in the projection list and that are mentioned in the list defined in this annotation should allow privileged access to the association target. No DCL check happens at runtime				
Value	Description								
associationRef	All associations that are exposed in the projection list and that are mentioned in the list defined in this annotation should allow privileged access to the association target. No DCL check happens at runtime								

Annotation	Meaning								
AccessControl.perso nalData.blocking	<p>This element defines the scope of the annotation.</p> <p>Scope: [#TABLE_FUNCTION], [#VIEW]</p> <p>Engine Behavior: The design-time engines handle the authorization check based on the value of the element.</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>REQUIRED</td><td> <p>Target CDS views, which use the annotation value REQUIRED, are use cases, where person-related data, including the related business partner master data, is valid for the primary business purpose as defined and used by the responsible data controller and should be processed. Blocked personal data can be included into the selection result, if the blocking status of the related business partner master data is set and when the user specifically has display authorization for blocked personal data.</p> </td></tr> <tr> <td>NOT_REQUIRED</td><td> <p>Target CDS views, which use the annotation value NOT_REQUIRED, are use cases, where no person-related data is accessed. Therefore, the users of the CDS views don't need to have an additional authorization condition for how to execute the data retrieval regarding the protection of personal data.</p> </td></tr> <tr> <td>BLOCKED_DATA_EXCLUDED</td><td> <p>The annotation value BLOCKED_DATA_EXCLUDED reassures the consumer that blocked personal data is never exposed and thus no authorization checks for blocked personal data are required. The reason for not exposing blocked personal data is purely related to the underlying business logic. For instance, a modeled filter condition for the specific business purpose of the view, such as active data, guarantees in combination with the EoP (End of Purpose) check of the application that blocked personal data can never be retrieved with the view. The modeled filter condition shouldn't be based on a specific personal data blocking status of the data or a specific authorization condition. For these cases, we recommend that the CDS entity use the annotations REQUIRED or #('TRANSACTIONAL_DATA').</p> </td></tr> </tbody> </table>	Value	Description	REQUIRED	<p>Target CDS views, which use the annotation value REQUIRED, are use cases, where person-related data, including the related business partner master data, is valid for the primary business purpose as defined and used by the responsible data controller and should be processed. Blocked personal data can be included into the selection result, if the blocking status of the related business partner master data is set and when the user specifically has display authorization for blocked personal data.</p>	NOT_REQUIRED	<p>Target CDS views, which use the annotation value NOT_REQUIRED, are use cases, where no person-related data is accessed. Therefore, the users of the CDS views don't need to have an additional authorization condition for how to execute the data retrieval regarding the protection of personal data.</p>	BLOCKED_DATA_EXCLUDED	<p>The annotation value BLOCKED_DATA_EXCLUDED reassures the consumer that blocked personal data is never exposed and thus no authorization checks for blocked personal data are required. The reason for not exposing blocked personal data is purely related to the underlying business logic. For instance, a modeled filter condition for the specific business purpose of the view, such as active data, guarantees in combination with the EoP (End of Purpose) check of the application that blocked personal data can never be retrieved with the view. The modeled filter condition shouldn't be based on a specific personal data blocking status of the data or a specific authorization condition. For these cases, we recommend that the CDS entity use the annotations REQUIRED or #('TRANSACTIONAL_DATA').</p>
Value	Description								
REQUIRED	<p>Target CDS views, which use the annotation value REQUIRED, are use cases, where person-related data, including the related business partner master data, is valid for the primary business purpose as defined and used by the responsible data controller and should be processed. Blocked personal data can be included into the selection result, if the blocking status of the related business partner master data is set and when the user specifically has display authorization for blocked personal data.</p>								
NOT_REQUIRED	<p>Target CDS views, which use the annotation value NOT_REQUIRED, are use cases, where no person-related data is accessed. Therefore, the users of the CDS views don't need to have an additional authorization condition for how to execute the data retrieval regarding the protection of personal data.</p>								
BLOCKED_DATA_EXCLUDED	<p>The annotation value BLOCKED_DATA_EXCLUDED reassures the consumer that blocked personal data is never exposed and thus no authorization checks for blocked personal data are required. The reason for not exposing blocked personal data is purely related to the underlying business logic. For instance, a modeled filter condition for the specific business purpose of the view, such as active data, guarantees in combination with the EoP (End of Purpose) check of the application that blocked personal data can never be retrieved with the view. The modeled filter condition shouldn't be based on a specific personal data blocking status of the data or a specific authorization condition. For these cases, we recommend that the CDS entity use the annotations REQUIRED or #('TRANSACTIONAL_DATA').</p>								

Annotation	Meaning
BLOCKED_DATA_INCLUDED	<p>Target CDS views, which use the annotation value BLOCKED_DATA_INCLUDED are exceptional use cases, where reporting for public authorities, for example, for auditing purposes, should be provided. This use case is a valid reason to process personal data beyond the end of the primary business purpose as defined and used by the responsible data controller. Companies are obliged to include all required data into these reports, which makes a differentiation between blocked and unblocked personal data obsolete. The users of the CDS views need a specific authorization for the application data or CDS view to execute the data retrieval for this specific use case, for example, based on a corresponding legal entity. It would contradict the protection of personal data, if these users would also always need a display authorization for blocked personal data, which they could use to see blocked personal data in other use cases.</p>
AccessControl.readAccess.logging.logdomain	<p>This element defines</p> <p>Scope: [#VIEW], [#ELEMENT], [#PARAMETER]</p> <p>Engine Behavior: The channel that implements the specific area sends the log-relevant field values to the read access logging framework.</p> <p>Values: Array of area and domain as String</p>
Value	Description
area as String	<p>Name of the business area of the log domain</p> <p>For a log domain, you specify a business area that the data element is related to. The business area is necessary because different applications might use the same log domain. For example, a log domain "account" might be something different in the Human Resources application than it is in the Banking application.</p>
domain as String	<p>Name of the log domain</p> <p>The log domain is a semantic description of semantically identical or related fields that have different technical representations.</p>
AccessControl.readAccess.logging.output	<p>This element defines</p> <p>Scope: [#VIEW]</p> <p>Engine Behavior: The value enables the logging of the log domain.</p> <p>Values:</p>
Value	Description

Annotation	Meaning
true	The defined log domains are logged.
false	The defined log domains are excluded from logging.
	This value is the default value.

NOTE

The value `#NOT_REQUIRED` is recommended for entities for which no authorization checks are planned yet, but might be needed by the developer or customer later. To prohibit roles for the entity, use the value `#NOT_ALLOWED`.

Examples

Example 1

When the developer activates the following DDL document, since an authorization check isn't required, ABAP development tools don't produce a warning. It doesn't matter whether a role exists for the entity or not.

At runtime, if there's a role for entity, then ABAP performs an authorization check with the role. If there's no role, there's no check and no protection for the entity.

↳ Sample Code

```
@AbapCatalog.sqlViewName: 'DEMO_CDS_PRJCTN'
@AccessControl.authorizationCheck: #NOT_REQUIRED
define view demo_cds_spfli
  as select from spfli
    { key spfli.carrid,
      key spfli.connid,
      spfli.cityfrom,
      spfli.cityto }
```

Example 2

↳ Sample Code

```
@AbapCatalog.sqlViewName: 'Z_MBB_SQL_SOI'
@AccessControl.authorizationCheck: #NOT_REQUIRED
@AccessControl.personalDataBlocking: #NOT_REQUIRED
@AccessControl.privilegedAssociation: ["tosalesorder"]
define view Z_MBB_CDS_SOI as select from snwd_so_i
  association [1] to sacm_cds_snwd_pd as topproduct on product_guid =
prod-uct_id
  association [1] to sacm_cds_snwd_so as tosalesorder on parent_key =
sales_id
{
  key node_key as salesitem_id,
  parent_key,
  product_guid,
  gross_amount,
  tax_amount,
  topproduct, --Association
  tosalesorder --Association
```

```
}
```

Example 3

This sample code shows how the annotations for read access logging can be used.

Sample Code

```
@AbapCatalog.sqlViewName: 'ZBANK_POSTINGS'
@Analytics.dataCategory: #CUBE
@AccessControl.readAccess.logging.output : true
define view Z_BankAccount_Postings
  as select from I_GLAccountLineItem
{
  AccountingDocument,
  PostingDate,
  @AccessControl.readAccess.logging.logdomain: { area : 'FINANCE' , domain :
'BANK' }
  HouseBankAccount,
  @AccessControl.readAccess.logging.logdomain: { area : 'FINANCE' , domain :
'BANK' }
  HouseBank,
  AmountInTransactionCurrency,
  TransactionCurrency
}
```

9.1.2 Analytics Annotations

Enable the Analytic Manager for multidimensional data consumption, performing data aggregation, and slicing and dicing data. BI front ends like Design Studio and Analysis Office can consume the data via the Analytic Manager.

Scope and Definition

```
@Scope:[#VIEW]
Annotation Analytics
{
  dataCategory : String enum { DIMENSION; FACT; CUBE; AGGREGATIONLEVEL; };
  dataExtraction
  {
    enabled : Boolean default true;
    delta :
    {
      byElement : elementRef;
      {
        name : RefToElement;
        maxDelayInSeconds : Integer default 1800;
        detectDeletedRecords: boolean default true;
        ignoreDeletionAfterDays : Integer;
      };
      changeDataCapture
      {
        automatic : Boolean default true;
        mapping
```

```

    {
        role : String(30) enum {MAIN; LEFT_OUTER_TO_ONE_JOIN;};
        table : String(30);
        // only used if association is not specified
        viewElement : array of ElementRef;
        // only used if association is not specified
        tableElement : array of ElementRef;
        filter : array of
        {
            tableElement : ElementRef;
            operator : String(11) enum
{EQ;NOT_EQ;GT;GE;LT;LE;BETWEEN;NOT_BETWEEN;} default #EQ;
            value : String(45);
            highValue : String(45);
        };
    };
    filter : array of
    {
        viewElement : ElementRef;
        operator : String(11) enum {EQ;NOT_EQ;GT;GE;LT;LE;BETWEEN;NOT_BETWEEN;}
    default #EQ;
        value : String(45);
        highValue : String(45);
    };
};
hidden : Boolean default true;
internalName : String(30) enum { DEFAULT; LOCAL; GLOBAL; };
planning
{
    enabled : Boolean default true;
};
query : Boolean default true;
settings.maxProcessingEffort : { LOW; MEDIUM; HIGH; UNLIMITED; } default
#HIGH;
settings.zeroValues.handling: String(20) enum { SHOW; HIDE; HIDE_IF_ALL; }
default #SHOW;
settings.zeroValues.hideOnAxis: String(20) enum { ROWS; COLUMNS;
ROWS_COLUMNS; } default #ROWS_COLUMNS;
viewModelReplication.enabled : Boolean default true;
writeBack
{
    className : String;
};
};

@Scope:[#ELEMENT]
Annotation Analytics
{
    internalName : String(30) enum { DEFAULT; LOCAL; GLOBAL; };
};

```

Usage

The Analytic Manager needs a star schema (multidimensional) and a query to consume the data. Most annotations to define the star schema in different CDS views are specified in `ObjectModel` annotations. The `Analytics` annotations also specify the facts (center of the star schema), extraction capabilities for replicating data into further systems, and analytic query properties. A semantic distinction can be made in the `Analytics` annotations between annotations that are relevant for the InfoProvider (CUBE) level and annotations that are only relevant for analytic queries.

Annotation	Meaning
Analytics.dataCategory	<p>Analytic queries can be defined on top of CDS views using the <code>Analytics.dataCategory</code> annotation.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): By specifying the data category, the developer can provide directives and hints, telling the analytic manager how to interpret individual entities for example.</p>
Value	Description
#DIMENSION	<p>This value indicates that the entity represents master data. This kind of view can be used for replication and for queries.</p>
#FACT	<p>This value indicates that the entity represents transactional data (center of star schema). Usually it contains the measures. Typically, these views are necessary for replication. They therefore should not be joined with master data views.</p>
#CUBE	<p>The #CUBE value (like #FACT) also represents factual data, but #CUBE does not have to be without redundancy. This means joins with master data are possible. Queries are usually built on top of #CUBE, where data is replicated from facts.</p>
#AGGREGATIONLEVEL	<p>This value indicates a projection. For this kind of view, the analytic manager offers write-back functionality (planning functionality). Views in this category have to select from a view with <code>dataCategory = #CUBE</code>, which supports the annotation <code>Analytics.writeBack.className</code>. No associations are allowed, and elements cannot be renamed.</p>
Analytics.dataExtraction.enabled	<p>Application developers can use this annotation to mark views that are suitable for data replication (for example, delta capabilities must be provided for mass data).</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine):</p>
<p>i Note</p> <p>This view must be annotated with <code>Analytics.dataCategory</code> (except the value AGGREGATIONLEVEL) or with <code>ObjectModel.dataCategory</code> with the value #TEXT or #HIERARCHY .</p>	
Value	Description
true	<p>The view is suitable for data replication. The default is true if this annotation is used.</p>
false	<p>The view is not suitable for data replication.</p>

Annotation	Meaning				
Analytics.dataExtraction.delta.byElement.name	<p>Application developers can enable the generic delta extraction with this annotation. This is the element that should be used for filtering during generic delta extraction. This element can be either a date (ABAP type DATS) or a UTC time stamp.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine):</p> <div style="border-left: 2px solid #0070C0; padding-left: 10px;"> i Note If the field is a time stamp, the system also checks the annotation Semantics.systemDate.lastChangedAt: true.. </div>				
Analytics.dataExtraction.delta.byElement.maxDelayInSeconds	<p>There is always a time delay between taking a UTC time stamp and the database commit. This annotation specifies the maximum possible delay in seconds.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine):</p>				
Analytics.dataExtraction.delta.byElement.detectDeletedRecords	<p>By using this annotation, the system will remember all key combinations of the view that were extracted in delta mode. If a key combination does not occur in the view anymore, this will automatically generate a delete image in the extracted data.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Detects deleted records.</p>				
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Boolean</td><td>The default is true if this annotation is used.</td></tr> </tbody> </table>	Value	Description	Boolean	The default is true if this annotation is used.
Value	Description				
Boolean	The default is true if this annotation is used.				

Annotation	Meaning						
<code>Analytics.dataExtraction.delta.byElement.ignoreDeletionAfterDays</code>	<p>This annotation only makes sense together with <code>Analytics.dataExtraction.delta.byElement.detectDeletedRecords</code>. The extraction will ignore deleted records if they are older than the specified number of days. The main purpose is archiving.</p> <p>Example</p> <p>If records are archived after two years, this value should be less than 700. In this case, the deletion in the database tables will be ignored if the record is only older than 700 days.</p>						
	<p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine):</p>						
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Integer</td><td>Specified number of days.</td></tr> </tbody> </table>	Value	Description	Integer	Specified number of days.		
Value	Description						
Integer	Specified number of days.						
<code>Analytics.dataExtraction.delta.changeDataCapture</code>	<p>Use the CDC (change data capture) delta mechanism based on data base triggers.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine):</p>						
<code>Analytics.dataExtraction.delta.changeDataCapture.automation</code>	<p>In case of simple CDS views like projections no mapping information needs to be provided, but the relevant information about key fields etc. is inferred by the system.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine):</p>						
<code>Analytics.dataExtraction.delta.changeDataCapture.mapping</code>	<p>In case of more complex CDS views, e.g. using joins, mapping information needs to be provided. With this annotation you define the mapping of exposed elements in the CDS view and the underlying table key fields; see also example 3.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine):</p>						
<code>Analytics.dataExtraction.delta.changeDataCapture.mapping.role</code>	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>MAIN</td><td>The key of the extraction view corresponds exactly to the key of the underlying main table to be logged.</td></tr> <tr> <td>LEFT_OUTER_TO_ONE_JOIN</td><td>There is a left outer join but not all requirements of role #COMPOSITION are met.</td></tr> </tbody> </table>	Value	Description	MAIN	The key of the extraction view corresponds exactly to the key of the underlying main table to be logged.	LEFT_OUTER_TO_ONE_JOIN	There is a left outer join but not all requirements of role #COMPOSITION are met.
Value	Description						
MAIN	The key of the extraction view corresponds exactly to the key of the underlying main table to be logged.						
LEFT_OUTER_TO_ONE_JOIN	There is a left outer join but not all requirements of role #COMPOSITION are met.						
<code>Analytics.dataExtraction.delta.changeDataCapture.mapping.table</code>	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>String (30)</td><td>Name of the table to be logged.</td></tr> </tbody> </table>	Value	Description	String (30)	Name of the table to be logged.		
Value	Description						
String (30)	Name of the table to be logged.						

Annotation	Meaning	
	Value	Description
Analytics.dataExtra ction.delta.changeD ataCapture.mapping. viewElement	array of ElementRef	<p>List of CDS view elements that map to key fields of the underlying tables.</p> <p>If mapping.role LEFT_OUTER_TO_ONE_JOIN, the list enumerates all exposed CDS view element names that correspond to key fields in this underlying outer table; in other words that correspond to the foreign key fields used in the on-conditions of the joins.</p>
Analytics.dataExtra ction.delta.changeD ataCapture.mapping. tableElement	array of ElementRef	<p>List of table key fields that map to CDS view elements.</p> <p>i Note</p> <p>Both arrays, viewElement and tableElement, must contain the same number of elements and must list them in the same order so that corresponding fields match. Additionally, the array tableElement must contain exactly all key elements of the logging table without the client element.</p>
Analytics.dataExtra ction.delta.changeD ataCapture.mapping. filter	<p>Value</p> <pre>«, Code Syntax array of { tableElement : ElementRef; operator : String(11) enum; value : String(45); highValue : String(45); }</pre>	<p>This annotation allows the definition of filters on the table to be logged for a CDS view. If the filter is one single value (no operator or operator #EQ) it can replace a missing mapping between viewElement and table Element.</p> <p>Filter on logging table:</p> <ul style="list-style-type: none"> .tableElement: Name of the element in the logging table or view that is to be restricted. .operator: Restriction operator, possible values are: #EQ, #NOT_EQ, #GT, #GE, #LT, #LE, #BETWEEN and #NOT_BETWEEN; default is #EQ. .value: The filter value. .highValue: That specifies the upper value for ranges defined in combination with operators #BETWEEN or #NOT_BETWEEN.
Analytics.dataExtra ction.filter	<p>Filters to be applied after the extraction of data.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine):</p>	
	Value	Description

Annotation	Meaning
	<p>Code Syntax</p> <pre>array of { viewElement : ElementRef; operator : String(11) enum; value : String(45); highValue : String(45);}</pre>
	<p>Filter on logging table:</p> <ul style="list-style-type: none"> .viewElement: Name of the element in the current view. .operator: Possible values are #EQ, #NOT_EQ, #GT, #GE, #LT, #LE, #BETWEEN and #NOT_BETWEEN; default is #EQ. .value: The filter value. .highValue: Only used in combination with operators #BETWEEN or #NOT_BETWEEN.
Analytics.hidden	<p>You can use this flag to decide whether the entity should be visible to analytic clients.</p> <p>Scope: #ELEMENT, #VIEW</p> <p>Evaluation Runtime (Engine): Views with Analytics.query are not exposed in value help. Views with Analytics.dataCategory are not exposed in value help for the CDS query designer.</p>
Value	Description
true	<ul style="list-style-type: none"> • The element will not be added to the InfoProvider model of Analytics. You can use this flag if there are elements in the view, which shouldn't be part of the InfoProvider model of Analytics for technical reasons, but the flag can't be deleted because the view is not consumed by Analytics. Using this flag, you can avoid error messages from Analytics for these elements. • The view cannot be consumed by analytic clients. The default is true if this annotation is used.
false	The view can be consumed by analytic clients.

Annotation	Meaning
Analytics.internalName	<p>With this annotation, the identifier which is used between BI Tools and the analytic manager can be influenced, meaning that adding <code>@ObjectModel.foreignKey.association</code> or <code>@ObjectModel.text.association</code> will not change the identifier.</p> <p>Scope: #ELEMENT, #VIEW</p> <p>Evaluation Runtime (Engine): It will be interpreted by the analytic manager.</p> <div style="border: 1px solid #ccc; padding: 10px;"> <p>i Note</p> <p>The CDS model is transformed to an InfoProvider or InfoObject model. The analytic protocols like INA or BICS are based on InfoProvider and InfoObject names. The elements of an analytic CDS view (CUBE, FACT, DIMENSION) are mapped to InfoObject names. If there is an <code>@ObjectModel.foreignKey.association</code> or <code>@ObjectModel.text.association</code>, and the target of the association is only used once, the InfoObject name is derived by default from the SQL view name of the target view. These names are called <code>global</code>. The InfoObject name for the representative key element of a dimension is always derived from the SQL view name of the dimension view. In all other cases, the InfoObject name is derived from the SQL view name of the view and the element name. These names are called <code>local</code>.</p> <p>If multiple elements are used in the ON condition of the foreign key association of an element, and the InfoObject name should be global, all other elements of the ON condition need to be global InfoObject names. This default behavior can be overruled with this annotation.</p> </div>
Value	Description
local	<ul style="list-style-type: none"> At Element level: The element is assigned to a local InfoObject name. If not possible, an error is raised. This is useful if the annotation <code>@ObjectModel.foreignKey.association</code> should be added to an element of an existing view. The InfoObject name will then not change. At View level: All elements for which it is possible will be assigned to a local InfoObject name. If it is used, <code>@ObjectModel.foreignKey.association / @ObjectModel.text.association</code> can be added, changed or removed without changing the InfoObject names.

Annotation	Meaning
global	<ul style="list-style-type: none"> At Element level: The element is assigned to an InfoObject with a global name. If not possible, an error is raised. It can be used if there are multiple elements referencing to the same dimension (same target of the foreign key association). In this case, one element can be specified. This is then given the global name, while all others are given local names. If not specified, the first element is given the global name. It is especially useful for an existing element if a new element should be added, which should reference to the same dimension. At View level: The default is <code>global</code>. All elements for which it is possible will be assigned to a global InfoObject name.
<code>Analytics.planning.enabled</code>	<p>An input-enabled query provides writeback capabilities and can be used in planning scenarios.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Specifies which view will be interpreted as input-enabled analytic query by the analytic manager.</p> <div style="background-color: #f0f8ff; padding: 10px;"> <p>i Note</p> <p>This view must be annotated with <code>Analytics.query: true</code>.</p> </div>
Value	Description
<code>true</code>	<p>The view is enabled for planning. The default is <code>true</code> if this annotation is used.</p> <div style="background-color: #f0f8ff; padding: 10px;"> <p>i Note</p> <p>The view has to select data from a view, which is annotated with <code>Analytics.dataCategory: #AGGREGATIONLEVEL</code>.</p> </div>
<code>false</code>	The view is not enabled for planning.
<code>Analytics.query</code>	<p>Query view classification.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): By tagging the CDS view, the developer can specify which views will be exposed to the analytic manager. This type of view will be interpreted as an analytic query by the analytic manager.</p> <div style="background-color: #f0f8ff; padding: 10px;"> <p>i Note</p> <p>This view must not be annotated with <code>Analytics.dataCategory = #NONE</code>.</p> </div>
Value	Description

Annotation	Meaning										
	<p>true</p> <p>The query view will be exposed to the analytic manager. The default is true if this annotation is used.</p>										
	<p>i Note</p> <p>Data will be selected from the view specified in the <code>from</code> clause. The view of the <code>from</code> clause has to be annotated with the <code>Analytics.dataCategory</code> as <code>#DIMENSION</code>, <code>#CUBE</code> or <code>#AGGREGATIONLEVEL</code>, and the DCL has to be assigned to the view of the <code>from</code> clause.</p>										
	<p>false</p> <p>The query view will not be exposed to the analytic manager.</p>										
<code>Analytics.settings.maxProcessingEffort</code>	<p>If InfoProviders store large amounts of data, certain queries can retrieve large result sets. The <code>maxProcessingEffort</code> is the maximum effort expected by analytic manager for processing the data it retrieves.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): This annotation will be interpreted by the analytic manager for query views (<code>Analytics.query: true</code>).</p>										
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>low</td><td></td></tr> <tr> <td>medium</td><td></td></tr> <tr> <td>high</td><td>The default is high.</td></tr> <tr> <td>unlimited</td><td></td></tr> </tbody> </table>	Value	Description	low		medium		high	The default is high.	unlimited	
Value	Description										
low											
medium											
high	The default is high.										
unlimited											
<code>Analytics.settings.zeroValues.handling</code>	<p>You can specify whether cells that contain zeros as values are to be displayed.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): This annotation will be interpreted by the analytic manager for query views (<code>Analytics.query: true</code>).</p>										
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>show</td><td>You can specify that zeros are to be displayed (No zero suppression takes place). The default is <code>show</code>.</td></tr> <tr> <td>hide</td><td>If you want to suppress zeros, there are two options available: <code>hide</code> and <code>hide_if_all</code>. For queries that have characteristics in the rows and key figures in the columns (or the other way around, with key figures in the rows and characteristics in the columns), the effect of the settings is identical. The two settings only differ in how they affect queries that have characteristics in the rows and columns (cross-classified table). If you use zero suppression with the <code>hide</code> value, the system checks whether there are zero values in the results area. If there are zero values in the results area, the corresponding row or column is hidden.</td></tr> </tbody> </table>	Value	Description	show	You can specify that zeros are to be displayed (No zero suppression takes place). The default is <code>show</code> .	hide	If you want to suppress zeros, there are two options available: <code>hide</code> and <code>hide_if_all</code> . For queries that have characteristics in the rows and key figures in the columns (or the other way around, with key figures in the rows and characteristics in the columns), the effect of the settings is identical. The two settings only differ in how they affect queries that have characteristics in the rows and columns (cross-classified table). If you use zero suppression with the <code>hide</code> value, the system checks whether there are zero values in the results area. If there are zero values in the results area, the corresponding row or column is hidden.				
Value	Description										
show	You can specify that zeros are to be displayed (No zero suppression takes place). The default is <code>show</code> .										
hide	If you want to suppress zeros, there are two options available: <code>hide</code> and <code>hide_if_all</code> . For queries that have characteristics in the rows and key figures in the columns (or the other way around, with key figures in the rows and characteristics in the columns), the effect of the settings is identical. The two settings only differ in how they affect queries that have characteristics in the rows and columns (cross-classified table). If you use zero suppression with the <code>hide</code> value, the system checks whether there are zero values in the results area. If there are zero values in the results area, the corresponding row or column is hidden.										

Annotation	Meaning								
	<p><code>hide_if_all</code></p> <p>For queries that have characteristics in the rows and key figures in the columns (or the other way around, with key figures in the rows and characteristics in the columns), the effect of the value is identical to <code>hide</code>. If your query has characteristics in the rows and columns however (cross-classified table), and you use zero suppression with the <code>hide_if_all</code> value, the system hides all rows or columns that contain zero values.</p>								
	<p>i Note</p> <p>If a detail row is not equal to zero (in a hierarchy for example), all superordinate rows are displayed, even if they contain zeros. As a result, the system ensures that the business context of the query is retained for the end user, even if the value <code>hide_if_all</code> is set.</p>								
<code>Analytics.settings.</code> <code>zeroValues.hideOnAx</code> <code>is</code>	<p>You can specify whether rows or columns that contain zeros as values are to be hidden.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): This annotation will be interpreted by the analytic manager for query views (<code>Analytics.query: true</code>).</p>								
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>rows</code></td><td>Only rows that contain zeros as values are to be hidden.</td></tr> <tr> <td><code>columns</code></td><td>Only columns that contain zeros as values are to be hidden.</td></tr> <tr> <td><code>rows_columns</code></td><td>The default is <code>rows_columns</code>. Rows and columns that contain zeros as values are to be hidden.</td></tr> </tbody> </table>	Value	Description	<code>rows</code>	Only rows that contain zeros as values are to be hidden.	<code>columns</code>	Only columns that contain zeros as values are to be hidden.	<code>rows_columns</code>	The default is <code>rows_columns</code> . Rows and columns that contain zeros as values are to be hidden.
Value	Description								
<code>rows</code>	Only rows that contain zeros as values are to be hidden.								
<code>columns</code>	Only columns that contain zeros as values are to be hidden.								
<code>rows_columns</code>	The default is <code>rows_columns</code> . Rows and columns that contain zeros as values are to be hidden.								
<code>Analytics.viewModel</code>	<p>The annotation is used to enable a view for view replication.</p>								
<code>Replication.enabled</code>	<p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): View replication is mainly used for analytical use cases.</p>								
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>true</code></td><td>The default is <code>true</code>.</td></tr> <tr> <td><code>false</code></td><td>The view isn't enabled for view replication.</td></tr> </tbody> </table>	Value	Description	<code>true</code>	The default is <code>true</code> .	<code>false</code>	The view isn't enabled for view replication.		
Value	Description								
<code>true</code>	The default is <code>true</code> .								
<code>false</code>	The view isn't enabled for view replication.								

Annotation	Meaning
Analytics.writeBack .className	<p>Views with Analytics.dataCategory: #AGGREGATIONLEVEL on top of this type of view can be used for planning scenarios. The ABAP class is used for saving the data, authorization checks and enqueueing.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Specifies which class enables the analytic manager to write data to the view.</p>
Value	Description
className as String	Name of an ABAP class, which implements the interface IF_RODPS_ODP_WRITEBACK.

Examples

Example 1

Example for replication-enabled master data.

↳ Sample Code

```
@EndUserText.label: 'EPM Demo: Employee'
@Analytics:{ dataCategory: #DIMENSION , dataExtraction.enabled: true }
//@VDM.viewType: #BASIC
@AccessControl.authorizationCheck: #CHECK
@ObjectModel.representativeKey: 'EmployeeUUID'
@AbapCatalog.sqlViewName: 'SEPM_IEMPLOYEE'
define view SEPM_I_Employee as
  select from snwd_employees
    association [0..1] to SEPM_I_Company           as _Company
    on $projection.CompanyUUID          = _Company.CompanyUUID
  ...
{
  key snwd_employees.node_key      as EmployeeUUID,
  @ObjectModel.foreignKey.association: '_Company'
  snwd_employees.parent_key       as CompanyUUID,
  ...
}
```

Example 2

Example for transactional data (fact).

↳ Sample Code

```
@EndUserText.label: 'EPM Demo: Sales Order Item with Addtl. Data (private
view)'
@Analytics.dataCategory: #CUBE
```

```

@AccessControl.authorizationCheck: #CHECK
@AbapCatalog.sqlViewName: 'SEPM_PSOIC'
define view SEPM_P_SalesOrderItemCube
  with parameters
    P_DisplayCurrency : snwd_curr_code           //for currency conversion, TODO:
  data element with better description
    as select from SEPM_I_SalesOrderItem
    association [0..1] to SEPM_I_SalesOrder_E      as _SalesOrder_E
    on $projection.salesorder          = _SalesOrder_E.SalesOrder
    ...
  {
    @ObjectModel.foreignKey.association: '_SalesOrder_E'
    key _SalesOrder.SalesOrder,
        _SalesOrder_E,
    ...
}

```

Example 3

Example for defining a list of tables that should be logged for delta data extraction.

↳ Sample Code

```

@Analytics: {
  dataCategory: #FACT,
  dataExtraction: {
    enabled: true,
    delta.changeDataCapture.mapping:
      [ { table: 'rsodpcdcunittab2', role: #MAIN, viewElement:
        ['view_key1', 'view_key2'], tableElement: ['tab_key1', 'tab_key2'] },
        { table: 'rsodpcdcunittab1', role: #LEFT_OUTER_TO_ONE_JOIN,
          viewElement: ['view_key3'], tableElement: ['tab_key1'] } ]
    }
}

```

Related Information

[ObjectModel Annotations \[page 428\]](#)

[Semantics Annotations \[page 445\]](#)

9.1.3 AnalyticsDetails Annotations

Enable application developers to specify the default multidimensional layout of the query, the sequence of variables in UI consumption, and the specific aggregation and planning behavior of the data. All these annotations can only be used in views with `@Analytics.query : true`.

Scope and Definition

`@Scope: [#ELEMENT]`

```

Annotation AnalyticsDetails
{
    query
    {
        formula : String;
        axis : String enum { FREE; ROWS; COLUMNS; };
        totals: String enum { HIDE; SHOW; };
        scaling : Integer;
        decimals : Integer;
        displayHierarchy : String enum { OFF; ON; FILTER; };
        hierarchyBinding : array of
        {
            type : String enum { ELEMENT; PARAMETER; CONSTANT; USER_INPUT; };
            value : String;
            variableSequence : Integer;
        };
        hierarchyInitialLevel : Integer;
        hierarchySettings: {
            hidePostedNodesValues: Boolean default true
        };
        elementHierarchy: {
            parent: RefToElement;
            initiallyCollapsed: Boolean default true
        };
    };
    exceptionAggregationSteps : array of
    {
        exceptionAggregationBehavior : String enum { SUM; MIN; MAX; NHA; COUNT;
COUNT_DISTINCT; AVG; STD; FIRST; LAST};
        exceptionAggregationElements : array of elementRef;
    };
    planning
    {
        enabled : Boolean default true;
        disaggregation : String enum { NONE; TOTAL; DIFFERENCE; };
        distribution : String enum { EQUAL; PROPORTIONAL; PROPORTIONAL_REF; };
        distributionReference : elementRef;
    };
    resultValueSource : String enum { CUBE; DIMENSION; };
};

@Scope:[#PARAMETER, #ELEMENT]
Annotation AnalyticsDetails
{
    query
    {
        display { KEY; TEXT; TEXT_KEY; KEY_TEXT; }
        hidden
        sortDirection
        variableSequence : Integer;
    };
}

```

Usage

Annotation	Meaning
<code>AnalyticsDetails.exceptionAggregationSteps.exceptionAggregationBehavior</code>	<p>Usually, the default aggregation determines how measures are aggregated in analytics.</p> <div style="border-left: 2px solid #f08030; padding-left: 10px;"><p>⚠ Caution</p><p>The default aggregation behavior cannot be defined in the query. It needs to be defined on the cube layer.</p></div> <p>In some cases different aggregation behavior is needed for a special element of the entity (dimension of a cube).</p> <div style="border-left: 2px solid #0070C0; padding-left: 10px;"><p>i Note</p><p>Example: A measure "Inventory" can be summed up for the different plants and other dimensions, but not for time – according to time the last or average value might be relevant.</p></div> <p>Exception aggregation is optional and is used to define different (to the default aggregation) aggregation behavior for specified elements. In general there can be multiple elements in which a measure has to be aggregated differently. Therefore a list of <code>ExceptionAggregationSteps</code> can be assigned.</p> <p><code>ExceptionAggregationBehavior</code> defines the aggregation behavior.</p> <div style="border-left: 2px solid #0070C0; padding-left: 10px;"><p>i Note</p><p>Example: In the query there is a measure, which should show the number of customers with positive sales - where sales is a measure of the underlying CUBE view with default aggregation SUM. When the sales measure is now used in a query with <code>exceptionAggregationBehavior: COUNT</code> and <code>exceptionAggregationElements : Customer</code>, the sales must first be aggregated (SUM) at customer level, and then COUNT has to be performed. If the sales for a customer is positive, this means that the sales is replaced by 1 (otherwise it is set to 0). This number can be summed up again.</p></div> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): The (logical) order in which the Analytic manager performs the aggregation is as follows: Firstly the DefaultAggregation is performed. This intermediate result is still grouped by all elements in the list of <code>ExceptionAggregationSteps</code>. The result is then aggregated by the exception aggregation in the order of StepNumber.</p> <p>The first remark holds even if the DefaultAggregation is FORMULA. This means that the calculation is performed when the result is still grouped by the exception aggregation elements. After the formula has been calculated, the result is aggregated according to the list of <code>ExceptionAggregationSteps</code>. This means that the aggregation level to be used for calculation can be defined precisely.</p>
Value	Description

Annotation	Meaning
SUM	sum
MIN	minimum
MAX	maximum
NHA	NHA
COUNT	counter
COUNT_DISTINCT	counter_distinct
AVG	average
STD	standard deviation
FIRST	FIRST
LAST	LAST
AnalyticsDetails.exceptionAggregationSteps.exceptionAggregationElements : ['']	<p>The elements which should be aggregated in this step. These elements must represent characteristics.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): Analytic manager: For more information see AnalyticsDetails.exceptionAggregationSteps.exceptionAggregationBehavior.</p>
Value	Description
array of elementRef	list of elements representing characteristics
AnalyticsDetails.planning.disaggregation.on	<p>This annotations allows you to define the disaggregation behavior. This annotation is only available for elements with AnalyticsDetails.planning.enabled.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): If a measure is specified with this annotation, it is input-ready in the analytic query. It enables the user to make manual entries for aggregated values. These values must be disaggregated on all the data records that contribute to the aggregated value of the cell.</p>
Value	Description
NONE	No Disaggregation (default)
TOTAL	Disaggregates the value entered.
DIFFERENCE	Disaggregates the difference to value entered.
AnalyticsDetails.planning.distribution	If disaggregation is chosen, you can choose how the value is distributed during disaggregation with this annotation.
	Scope: #ELEMENT
	Evaluation Runtime (Engine): This annotation will be ignored if CDS view or the corresponding element is not enabled for planning.

Annotation	Meaning								
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>EQUAL</td><td>Equal distribution (default)</td></tr> <tr> <td>PROPORTIONAL</td><td>Proportional to the current value</td></tr> <tr> <td>PROPORTIONAL_REF</td><td>Proportional to the current value of a sibling member in the selection list, which needs to be specified via the <code>AnalyticsDetails.planning.distributionReference</code> annotation.</td></tr> </tbody> </table>	Value	Description	EQUAL	Equal distribution (default)	PROPORTIONAL	Proportional to the current value	PROPORTIONAL_REF	Proportional to the current value of a sibling member in the selection list, which needs to be specified via the <code>AnalyticsDetails.planning.distributionReference</code> annotation.
Value	Description								
EQUAL	Equal distribution (default)								
PROPORTIONAL	Proportional to the current value								
PROPORTIONAL_REF	Proportional to the current value of a sibling member in the selection list, which needs to be specified via the <code>AnalyticsDetails.planning.distributionReference</code> annotation.								
<code>AnalyticsDetails.planning.distributionReference</code>	<p>If disaggregation is chosen and <code>AnalyticsDetails.planning.distribution : #PROPORTIONAL_REF</code>, you can specify a sibling member in the selection list for reference with this annotation.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): This annotation will be ignored if CDS view or the corresponding element is not enabled for planning.</p>								
	<table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>elementRef</td> <td>Name of sibling member for reference</td> </tr> </tbody> </table>	Value	Description	elementRef	Name of sibling member for reference				
Value	Description								
elementRef	Name of sibling member for reference								
<code>AnalyticsDetails.planning.enabled</code>	<p>Individual members in the selection list (no calculated elements) need to be annotated for enabling planning. The list can only be used in input-enabled analytic queries. This means that the views have to be annotated with <code>Analytics.query: true</code> and <code>Analytics.planning.enabled: true</code>.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): If a measure is specified with this annotation, it is input-ready in the analytic query.</p>								
	<table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>true</td> <td>The member in the selection list is enabled for planning. This is the default setting.</td> </tr> <tr> <td>false</td> <td>The member in the selection list is not enabled for planning.</td> </tr> </tbody> </table>	Value	Description	true	The member in the selection list is enabled for planning. This is the default setting.	false	The member in the selection list is not enabled for planning.		
Value	Description								
true	The member in the selection list is enabled for planning. This is the default setting.								
false	The member in the selection list is not enabled for planning.								
<code>AnalyticsDetails.query.axis</code>	<p>The elements of the view can be positioned on multiple axes. The elements can be directly annotated with their axis. Measures (elements which can be aggregated (<code>defaultAggregation</code>)) all need to be on the same axis. The annotation of the first measure will therefore be used for all measures of the query. If no <code>AnalyticsDetails.query.axis</code> is found, the system positions the measures on the columns.</p> <p>The default value for elements which are not measures is the free axis. Note that elements in the projection list, which belong to the same field in the query, will be grouped together.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): If the BI client does not specify which elements are on which axis, the layout is derived by this annotation.</p>								
	<table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> </table>	Value	Description						
Value	Description								

Annotation	Meaning
	FREE Default value for elements other than measures.
	ROWS
	COLUMNS Default value for measures.
AnalyticsDetails.query.decimals	For measures, restricted measures and calculated elements, you can set the number of decimals to be used. Scope: #ELEMENT Evaluation Runtime (Engine): Analytic manager: This annotation is only relevant for measures. For other elements it will be ignored.
Value	Description
Integer number from 0 to 9.	Numbers will displayed with n decimals.
AnalyticsDetails.query.display	Scope: #ELEMENT Evaluation Runtime (Engine): Analytic manager
Value	Description
KEY	The key will be displayed.
TEXT	The text will be displayed.
TEXT_KEY	Text and key will be displayed.
KEY_TEXT	Key and text will be displayed.
AnalyticsDetails.query.displayHierarchy	This annotation allows you to specify the display hierarchy attribute for the element. It is not possible for measures.
Y	Scope: #ELEMENT Evaluation Runtime (Engine): Analytic manager The query result is shown in a hierarchy for the specified element.
Value	Description
OFF	No display hierarchy
ON	With display hierarchy. The hierarchy used can be specified by AnalyticsDetails.query.hierarchyBinding
FILTER	The display hierarchy is the same one defined on the filter for this element. In this case, a hierarchy filter needs to be defined on the same element. The hierarchy binding is taken from the filter.
AnalyticsDetails.query.elementHierarchy.initiallyCollapse	If true, the hierarchy node represented by the annotated entry will be initially collapsed (only applicable if the annotated entry has children, i.e. is parent of another entry). Scope: #ELEMENT Evaluation Runtime (Engine):

Annotation	Meaning				
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Boolean</td><td>true is the default setting.</td></tr> </tbody> </table>	Value	Description	Boolean	true is the default setting.
Value	Description				
Boolean	true is the default setting.				
AnalyticsDetails.query.elementHierarchy.parent	<p>Measures, restricted measures and calculated elements will be shown as a flat list in the keyfigure structure of the Analytic Query. To achieve a hierarchical display, select list entries can be annotated with @AnalyticsDetails.query.elementHierarchy.parent and will appear hierarchically below the specified parent entry.</p> <p>Scope:#ELEMENT</p> <p>Evaluation Runtime (Engine):</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>elementRef</td><td>Alias of the select list entry serving as parent for the annotated entry.</td></tr> </tbody> </table>	Value	Description	elementRef	Alias of the select list entry serving as parent for the annotated entry.
Value	Description				
elementRef	Alias of the select list entry serving as parent for the annotated entry.				
AnalyticsDetails.query.formula	<p>This annotation allows you to specify the formula expression, which cannot be expressed as an SQL formula (operands required from the element list of the view). Only numerical values (measures) can be used as operands.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): Analytic manager: This annotation will be interpreted as a formula.</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>String</td><td> <p>This expression can contain the following:</p> <ul style="list-style-type: none"> • cube measures • arithmetic expressions • functions NDIVO and NODIM • CASE expressions with a maximum of one THEN clause (will be translated into BW IF operator) <ul style="list-style-type: none"> ◦ WHEN clause can contain conditional or Boolean expressions of measures ◦ ELSE clause is optional (default to ELSE 0) </td></tr> </tbody> </table>	Value	Description	String	<p>This expression can contain the following:</p> <ul style="list-style-type: none"> • cube measures • arithmetic expressions • functions NDIVO and NODIM • CASE expressions with a maximum of one THEN clause (will be translated into BW IF operator) <ul style="list-style-type: none"> ◦ WHEN clause can contain conditional or Boolean expressions of measures ◦ ELSE clause is optional (default to ELSE 0)
Value	Description				
String	<p>This expression can contain the following:</p> <ul style="list-style-type: none"> • cube measures • arithmetic expressions • functions NDIVO and NODIM • CASE expressions with a maximum of one THEN clause (will be translated into BW IF operator) <ul style="list-style-type: none"> ◦ WHEN clause can contain conditional or Boolean expressions of measures ◦ ELSE clause is optional (default to ELSE 0) 				
AnalyticsDetails.query.hidden	<p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): Analytic manager:</p>				

Annotation	Meaning										
AnalyticsDetails.query.hierarchyBinding.type	<p>The <code>AnalyticsDetails.query.hierarchyBinding</code> annotations allow you to specify a special hierarchy for an element with a display hierarchy. One tuple has to be specified for each key field of the hierarchy directory view. The first entry is supplied to the first key field of the hierarchy directory, the second entry to the second key field, and so on. If only one hierarchy exists, the hierarchy binding can be omitted.</p> <p><code>AnalyticsDetails.query.hierarchyBinding.type</code> determines how the key element is filled (by a constant, a parameter, a filtered element or by a user input field).</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): Analytic manager</p>										
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>#ELEMENT</td><td>Name of an element, which has a unique filter. At runtime the filter value is used for this hierarchy component.</td></tr> <tr> <td>#PARAMETER</td><td>Parameter name</td></tr> <tr> <td>#CONSTANT</td><td>Constant</td></tr> <tr> <td>#USER_INPUT</td><td> <p><code>USER_INPUT</code> is optional. It will be requested at runtime of the analytic query. It can contain a name. <code>USER_INPUT</code> with the same name will be provided with the same user input at runtime. The variable is placed in the list of all values in accordance with <code>AnalyticsDetails.query.variableSequence</code>.</p> </td></tr> </tbody> </table>	Value	Description	#ELEMENT	Name of an element, which has a unique filter. At runtime the filter value is used for this hierarchy component.	#PARAMETER	Parameter name	#CONSTANT	Constant	#USER_INPUT	<p><code>USER_INPUT</code> is optional. It will be requested at runtime of the analytic query. It can contain a name. <code>USER_INPUT</code> with the same name will be provided with the same user input at runtime. The variable is placed in the list of all values in accordance with <code>AnalyticsDetails.query.variableSequence</code>.</p>
Value	Description										
#ELEMENT	Name of an element, which has a unique filter. At runtime the filter value is used for this hierarchy component.										
#PARAMETER	Parameter name										
#CONSTANT	Constant										
#USER_INPUT	<p><code>USER_INPUT</code> is optional. It will be requested at runtime of the analytic query. It can contain a name. <code>USER_INPUT</code> with the same name will be provided with the same user input at runtime. The variable is placed in the list of all values in accordance with <code>AnalyticsDetails.query.variableSequence</code>.</p>										
AnalyticsDetails.query.hierarchyBinding.value	<p>This annotation contains, depending on the type, a literal value, the parameter name (without <code>:</code> or <code>\$parameter</code>), the element name and an identifier for the user input field, respectively.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): Analytic manager</p>										
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>String (512)</td><td></td></tr> </tbody> </table>	Value	Description	String (512)							
Value	Description										
String (512)											
AnalyticsDetails.query.hierarchyInitialLevel	<p>This annotation defines which hierarchy level will be displayed initially.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): Analytic manager</p>										
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Integer: Integer number</td><td>Hierarchy level with that number will be displayed initially.</td></tr> </tbody> </table>	Value	Description	Integer: Integer number	Hierarchy level with that number will be displayed initially.						
Value	Description										
Integer: Integer number	Hierarchy level with that number will be displayed initially.										

Annotation	Meaning								
AnalyticsDetails.query.hierarchyBinding.g.variableSequence	<p>This annotation allows you to specify the order of parameters and variables on the variable input UIs.</p> <p>i Note</p> <p>You can also use the annotation Consumption.filter.hierarchyBinding.variableSequence.</p>								
	<p>In case filters or parameters are not annotated they are displayed after the annotated ones in the order they appear in the CDS document.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): Analytic manager</p>								
AnalyticsDetails.query.hierarchySettings.hidePostedNodesValues	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Integer</td><td>This number defines the position of the variable generated for the user input field.</td></tr> </tbody> </table> <p>For some hierarchies (typical example: CostCenter hierarchy), the hierarchy nodes can have posted values on their own (and not just represent the aggregation over all its child nodes) which are displayed as separate rows in the report. Using this annotation the display of the posted values can be suppressed.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine):</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Boolean</td><td>true is the default setting.</td></tr> </tbody> </table>	Value	Description	Integer	This number defines the position of the variable generated for the user input field.	Value	Description	Boolean	true is the default setting.
Value	Description								
Integer	This number defines the position of the variable generated for the user input field.								
Value	Description								
Boolean	true is the default setting.								
AnalyticsDetails.query.scaling	<p>For measures, restricted measures and calculated elements, you can set the scaling factor to be used.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): This annotation is only relevant for measures. For other elements it will be ignored.</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Integer: Integer number from 0 to 9.</td><td>Numbers will be scaled by a factor of 10^{**n}.</td></tr> </tbody> </table>	Value	Description	Integer: Integer number from 0 to 9.	Numbers will be scaled by a factor of 10^{**n} .				
Value	Description								
Integer: Integer number from 0 to 9.	Numbers will be scaled by a factor of 10^{**n} .								
AnalyticsDetails.query.sortDirection	<p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): Analytic manager:</p>								
AnalyticsDetails.query.totals	<p>For attributes you can set the behavior for totals.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): These annotations will be ignored for measures.</p>								

Annotation	Meaning	
	Value	Description
	HIDE	Totals or subtotals are not added to the result set for this element.
	SHOW	In addition to the details, the subtotals are added to the result set for this element.
AnalyticsDetails.query.variableSequence	<p>If user input is necessary for the parameter, the sequence of all fields for user input at runtime can be specified with this annotation.</p> <p>Scope: #PARAMETER, #ELEMENT</p> <p>Evaluation Runtime (Engine): UIs will create user prompts for parameters which require a user input or elements with filters (consumption.filter). This annotation can be used to specify the sequence of user prompts at runtime. If filters or parameters are not annotated, they are displayed after the annotated ones - in the order they appear in the CDS document.</p>	
	Value	Description
	Integer	Order of parameters and variables. The integer values may contain gaps.
AnalyticsDetails.resultValueSource	<p>This annotation influences the list of values, which should be taken into account for a specific characteristic.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): When the query is executed, a row of data is only displayed for the characteristic if there are posted values for this characteristic.</p>	
	Value	Description
	CUBE	All values available in the view of the <code>from</code> clause.
	DIMENSION	All values available in dimension.

Examples

Example 1

Calculated Elements

↳ Sample Code

```
@Analytics.query : true
define view financial as select from sales
{
  @AnalyticsDetails.query.axis : #ROWS
  product,
  @AnalyticsDetails.query.axis : #COLUMNS
  @AnalyticsDetails.query.formula : 'revenue - cost'
  1 as absolute_margin,
  @AnalyticsDetails.query.formula : 'NDIV0($projection.absolute_margin /
revenue) * 100'
  1 as relative_margin,
```

```

@AnalyticsDetails.query.formula : 'CASE WHEN $projection.relative_margin >
20 THEN revenue ELSE 0 END'
  1 as revenue_for_margin_gt_20
}

```

Example 2

Display hierarchy selection: Specify the hierarchy directly.

↳ Sample Code

```

@Analytics.query : true
define view costcenter_reporting
  with parameters
    cost_center_hier_param : String
  as select from costcenters
{
  ...
  @AnalyticsDetails.query : {
    displayHierarchy: #ON,

    hierarchyInitialLevel: 3, // three levels of the hierarchy will be
    opened at start
    hierarchySettings{hidePostedNodesValues: true }, // don't show posted
    node values
    hierarchyBinding :
      [ { type : #CONSTANT, value : 'CONTR_AREA_10' },
        { type : #PARAMETER, value : 'cost_center_hier_param' }
      ]
  }
  costcenter, // hierarchy node filter
  costs,
  ...
}

```

Example 3

Use of \$session variables, especially \$session.system_date.

The system supports variables like \$session.system_date for different use cases:

- in cube parameters

↳ Sample Code

```

@Analytics.query : true
define view ...
  as select from zCostCenter_Flt( P_KeyDate: $session.system_date )

```

- in queries on time-dependent master data in where clauses

↳ Sample Code

```

} where
  DateTo    >= $session.system_date and
  DateFrom <= $session.system_date

```

- in paths to time-dependent attributes or texts

« Sample Code

```
_CostCenter[1: DateTo >= $session.system_date and DateFrom <=
$session.system_date]._Text[1: Language = $parameters.P_Language].Text as
CostCenterText,
```

- in restricted key figures

« Sample Code

```
case when PostingDate = $session.system_date
```

Related Information

[Aggregation Annotations \[page 414\]](#)

9.1.4 Consumption Annotations

Define a specific behavior that relates to the consumption of CDS content through domain-specific frameworks.

Scope and Definition

```
@Scope:[#ELEMENT, #PARAMETER]
Annotation Consumption
{
    labelElement : elementRef;
    quickInfoElement : ElementRef;
    hidden : Boolean default true;
    derivation
    {
        lookupEntity : entityRef;
        pfcgMapping : String(30);
        resultElement : elementRef;
        resultElementHigh : elementRef;
        resultHierarchyNode
        {
            nodeTypeElement : ElementRef;
            mapping : array of
            {
                hierarchyElement : ElementRef;
                lookupElement : ElementRef;
            };
        };
        binding : array of
        {
            targetParameter : parameterRef;
            targetElement : elementRef;
            type : String(10) enum { ELEMENT; PARAMETER; CONSTANT; SYSTEM_FIELD };
        };
    };
}
```

```

        value : String(512);
    };
};

};

@Scope:[#ENTITY, #PARAMETER, #ELEMENT]
Annotation Consumption
{
    semanticObject : String;
};

@Scope:[#ELEMENT]
Annotation.Consumption
{
    filter
    {
        mandatory : Boolean default true;
        defaultValue : Expression;
        defaultValueHigh : Expression;
        defaultHierarchyNode: { nodeType : elementRef ,
                               node : array of { element : elementRef,
                                                 value : expression } };
        hidden : Boolean default true;
        selectionType      : String(20) enum {SINGLE; INTERVAL; RANGE};
    HIERARCHY_NODE }
        multipleSelections : Boolean default true;
        hierarchyBinding : array of
        {
            type : String(10) enum { ELEMENT; PARAMETER; CONSTANT; USER_INPUT; };
            value : String(512);
            variableSequence : Integer;
        };
        businessDate :
        {
            at : Boolean default true;
        };
    };
    groupWithElement: elementRef;
    ranking
    {
        functionParameterBinding : array of
        {
            functionId  : String(120);
            parameterId : String(120);
        };
    }
};

@Scope: [#PARAMETER,#ELEMENT]
Annotation.Consumption
{
    defaultValue : String(1024);
    valueHelp      : ElementRef;
    valueHelpDefinition: array of
    {
        qualifier: String(120);
        entity
        {
            @Scope:[#VIEW, #ELEMENT, #PARAMETER]
            name   : String(40);
            element : String(40);
        };
        association      : AssociationRef;
        distinctValues    : Boolean default true;
        additionalBinding : array of
        {
            localParameter : ParameterRef;
            localElement   : ElementRef;
            parameter      : String(40);
            element        :
        String(40);
        usage          : String(30) enum

```

```

    {
        FILTER;
        RESULT;
        FILTER_AND_RESULT;
    };

};

label : String(60);
presentationVariantQualifier : String(120);
};

@Scope: [#VIEW]
Annotation.Consumption
{
    dbHints : array of String(1298);
    dbHintsCalculatedBy : String(255);
}

```

Usage

Via these annotations, the specific behavior is defined which is related to the consumption of CDS content. This metadata makes no assumptions about the concrete consumption technology/infrastructure, but it is applicable across multiple consumption technologies (e.g. Analytics or OData).

Annotation	Meaning
Consumption.defaultValue	<p>This annotation enables specification of a default value for a View parameter or a View element.</p> <p>Scope: #PARAMETER, #ELEMENT</p> <p>Evaluation Runtime (Engine): This annotation will be interpreted by the analytic manager or .</p> <p>Values:</p> <p>This value is either proposed to the end user or implicitly set by the consumer (framework) of the view whenever the end user does not explicitly specify a different value. This default value is transparent for the CDS runtime, which means that the View parameter is still mandatory at this level. Therefore, the consumption framework is responsible for passing the specified default value as a parameter binding when the view is invoked.</p>

i Note

The `defaultValue` annotation is only allowed for View parameters. Elements and default values can only be specified within filters. Hence, the annotation `@Consumption.filter.defaultValue` has to be used.

Annotation	Meaning						
Consumption.derivation	<p>This annotation enables derivation of the value for a parameter or a filter automatically at runtime by selecting a row from a given entity (table).</p> <p>For more information see Annotations for Derivations [page 248].</p> <p>An element can be annotated with Consumption.derivation only if for this element Consumption.filter is present. The derivation is then used to determine the filter value. An empty derivation of a hidden and mandatory parameter/filter will cause a runtime error.</p>						
	<p>i Note</p> <p>Analytic manager: Annotation of a non-filtered element of the projection list with a derivation is not allowed.</p>						
	<p>Scope: #ELEMENT, #PARAMETER</p> <p>Evaluation Runtime (Engine): This annotation will be interpreted by the analytic manager and SADL.</p>						
Consumption.derivation.binding.targetElement	<p>This sub-annotation enables provision of a parametrization for the (scalar) input parameters of lookupEntity or procedure.</p>						
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>elementRef</td><td></td></tr> </tbody> </table>	Value	Description	elementRef			
Value	Description						
elementRef							
Consumption.derivation.binding.targetParameter	<p>This annotation enables provision of a parametrization for the (scalar) input parameters of the lookupEntity procedure. This sub-annotation specifies a parameter name of lookupEntity or procedure.</p>						
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>parameterRef</td><td></td></tr> </tbody> </table>	Value	Description	parameterRef			
Value	Description						
parameterRef							
Consumption.derivation.binding.type	<p>This sub-annotation specifies how the target is filled (by a constant, a parameter or a (filtered) element).</p> <p>The following enumerations (String (10)) are provided:</p>						
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>#ELEMENT</td><td></td></tr> <tr> <td>#PARAMETER</td><td></td></tr> </tbody> </table>	Value	Description	#ELEMENT		#PARAMETER	
Value	Description						
#ELEMENT							
#PARAMETER							

Annotation	Meaning				
	#CONSTANT				
Consumption.derivation.binding.value	<p>This sub-annotation contains, depending on the type, the constant value, the parameter name (without : or \$parameter) or the element name.</p> <p>i Note</p> <p>Analytic manager: If parameter values are used in this sub-annotation, the corresponding parameter must be declared before the parameter that is to be derived. For SADL the order does not matter.</p>				
	<p>i Note</p> <p>Analytic manager: All provided annotation values are treated case sensitive.</p>				
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>String (512)</td><td>constant value, parameter name (without : or \$parameter) or element name</td></tr> </tbody> </table>	Value	Description	String (512)	constant value, parameter name (without : or \$parameter) or element name
Value	Description				
String (512)	constant value, parameter name (without : or \$parameter) or element name				
Consumption.derivation.lookupEntity	<p>Analytic manager: Reads the result to fill the parameter. SADL: Reads the result to fill the parameter and filter.</p> <table border="1"> <thead> <tr> <th>Values:</th><th>Description</th></tr> </thead> <tbody> <tr> <td>entityRef</td><td></td></tr> </tbody> </table>	Values:	Description	entityRef	
Values:	Description				
entityRef					

Annotation	Meaning				
Consumption.derivation.pfcgMapping	<p>Usecase: The query should be restricted by the hierarchy nodes, the user is authorized for.</p> <p>The PFCG-mapping defines a mapping between a PFCG-object and the access policy of an DCL. At runtime it contains all values from the PFCG-object which are assigned to an user. This annotation can be used in a similar way as <code>Consumption.derivation.lookupEntity</code>.</p> <p>i Note</p> <p>If this annotation is used for a parameter, then with <code>Consumption.derivation.resultElement</code> the field has to be specified, from which the distinct values should be selected. Only fields can be used, which belong to the key of the hierarchy. It can be used, if the hierarchy should also be derived from the PFCG-mapping. The parameter has to be used in the <code>Consumption.filter.hierarchyBinding</code> annotation.</p> <p>If this annotation is used for an element in combination with <code>Consumption.filter.selectionType : #HIERARCHY_NODE</code>, then no mapping is needed, since this mapping can be derived from the PFCG-mapping. Only the nodes are selected which fit to the specified hierarchy.</p>				
Consumption.derivation.resultElement	<table border="1"> <thead> <tr> <th>Values:</th><th>Description</th></tr> </thead> <tbody> <tr> <td>String (30)</td><td>name of an existing PFCG-Mapping</td></tr> </tbody> </table> <p>Element of the entity/scalar export parameter of the procedure providing the result.</p>	Values:	Description	String (30)	name of an existing PFCG-Mapping
Values:	Description				
String (30)	name of an existing PFCG-Mapping				
Consumption.derivation.resultElementHigh	<table border="1"> <thead> <tr> <th>Values:</th><th>Description</th></tr> </thead> <tbody> <tr> <td>elementRef</td><td></td></tr> </tbody> </table> <p>Element of the interval export parameter of the procedure providing the result.</p>	Values:	Description	elementRef	
Values:	Description				
elementRef					
	<table border="1"> <thead> <tr> <th>Values:</th><th>Description</th></tr> </thead> <tbody> <tr> <td>elementRef</td><td></td></tr> </tbody> </table>	Values:	Description	elementRef	
Values:	Description				
elementRef					

Annotation	Meaning
<code>Consumption.derivation.resultHierarchyNode.no deTypeElement</code>	This annotation has to reference a field in the lookup entity, which returns the node types. In <code>Consumption.derivation</code> either <code>resultElement</code> or <code>resultHierarchyNode</code> can be used.
	A chargeable node is a node with node type equal to the element.
	<p>Example</p> <p>1000/4711 is a costcenter which has further costcenters as children.</p>
	<p>There is the need to distinguish between the node (a set of all its leaves) and the pure leaf.</p> <ul style="list-style-type: none"> • If the selected value for <code>Consumption.derivation.resultHierarchyNode.nodeTypeElement</code> is initial, then the leaf will be addressed. • If the value is equal to the element, for which it is used, then the node will be addressed. If there is no node with such a name, then the leaf is addressed.
	<p>Example</p> <p>If the selected record from the lookup entity is equal to <code>(nodeTypeElement, controllingArea, costCenter) = (space, 1000, 4711)</code> means always the leaf. While <code>(costCenter, 1000, 4711)</code> means the node. If this does not exist, then it is interpreted as a leaf.</p>
Values:	Description

Annotation	Meaning						
	<p><code>elementRef</code></p> <p>This is a field in the lookup view which returns the element which describes the node type. The values which the lookup returns for the <code>nodeType</code>, have to be element names from the associated hierarchy view (<code>@ObjectModel.dataCategory: #HIERARCHY</code>).</p> <p>⚠ Caution</p> <p><code>Consumption.filter.defaultHierarchyNode.nodeType</code> is an element name which describes the node type.</p>						
<code>Consumption.derivation.resultHierarchyNode.mapping</code>	<p>The view defined in <code>lookupEntity</code> contains fields which can be mapped to the semantic fields of the hierarchy node view. The mapping of the fields is described in <code>Consumption.derivation.resultHierarchyNode.mapping</code>. This annotation is optional and can be omitted, if the fieldnames in the lookup entity correspond to these in the hierarchy node view.</p> <table border="1"> <thead> <tr> <th>Array of:</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>hierarchyElement : elementRef</code></td><td></td></tr> <tr> <td><code>lookupElement : elementRef</code></td><td></td></tr> </tbody> </table>	Array of:	Description	<code>hierarchyElement : elementRef</code>		<code>lookupElement : elementRef</code>	
Array of:	Description						
<code>hierarchyElement : elementRef</code>							
<code>lookupElement : elementRef</code>							

Annotation	Meaning
Consumption.filter	<p>This annotation enables filtering elements of the underlying view. A filter should be specified before executing a query on the view.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL - Translates the following CDS annotations into the corresponding OData annotations: <ul style="list-style-type: none"> ◦ Consumption.filter.defaultValue ◦ Consumption.filter.hidden ◦ Consumption.filter.mandatory ◦ Consumption.filter.multipleSelections ◦ Consumption.filter.selectionType • Analytic Manager supports additionally to the under SADL listed annotations also the following annotations: <ul style="list-style-type: none"> ◦ Consumption.filter.defaultValueHigh ◦ Consumption.filter.defaultHierarchyNode ◦ Consumption.filter.hierarchyBinding.type ◦ Consumption.filter.hierarchyBinding.value ◦ Consumption.filter.hierarchyBinding.variableSequence <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>i Note</p> <p>This element should be exposed by UIs. UIs based on Analytic Manager will offer this element as variables.</p> </div>

Annotation	Meaning
Consumption.filter.defaultValue	<p>This annotation can be used to provide a fixed default value for a filter. This value is completely transparent for the CDS runtime, which means that the consumer is responsible for explicitly extending the - This annotation will be interpreted by the analytic manager:</p> <p>WHERE condition.</p> <p>The default value is either proposed to the end user or implicitly set by the consumer (framework) of the view whenever the end user doesn't explicitly specify a different value.</p>
Value	Description
Expression	Characteristic members need to be provided in an internal noncompounded format.
Consumption.filter.defaultValueHigh	<p>This annotation, together with Consumption.filter.defaultValue, allows to specify a default interval.</p>
Value	Description
Expression	Characteristic members need to be provided in an internal noncompounded format.
Consumption.filter.defaultHierarchyNode.nodeType	<p>Consumption.filter.defaultHierarchyNode can only be used in combination with Consumption.filter.selectionType = #HIERARCHY_NODE.</p> <p>In Consumption.filter.hierarchyBinding, the hierarchy is specified</p> <p>The nodeType-annotation references to elements in the hierarchy node view.</p>
Value	Description
elementRef	
Consumption.filter.defaultHierarchyNode.node	<p>The node - This-annotation references to elements in the hierarchy node view.</p>
Array of:	Description

Annotation	Meaning	
	element : elementRef	
	value : expression	Elements with value initial can be omitted in the node array.
Consumption.filter.hidden	The filter will not be shown in the UIs. In combination with a derivation, this leads to a dynamic filter at runtime without user interaction.	
	Value	Description
	Boolean (true, false)	Defines whether the filter is hidden for the client or not.
	Default: true	
Consumption.filter.hierarchyBinding.type	This annotation determines how the key element is filled (by a constant, a parameter, a filtered element or by a user input field).	
	Value	Description
	#ELEMENT	Name of an element, which has a unique filter. At runtime, the filter value is used for this hierarchy component.
	#PARAMETER	Parameter name
	#CONSTANT	Constant

Annotation	Meaning				
#USER_INPUT	USER_INPUT is optional. It will be requested at runtime of the analytic query. It can contain a name. USER_INPUT with the same name will be provided with the same user input at runtime. The variable is placed in the list of all values in accordance with Consumption.filter.hierarchyBinding.variableSequence.				
Consumption.filter.hierarchyBinding.value	This annotation contains, depending on the type, a literal value, the parameter name (without : or \$parameter), the element name and an identifier for the user input field, respectively.				
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>String (512)</td><td></td></tr> </tbody> </table>	Value	Description	String (512)	
Value	Description				
String (512)					
Consumption.filter.hierarchyBinding.variableSequence	This annotation allows you to specify the order of parameters and variables on the variable input UIs.				
<p>i Note</p> <p>You can also use the annotation AnalyticsDetails.query.hierarchyBinding.variableSequence.</p>					
<p>In case filters or parameters are not annotated, they are displayed after the annotated ones in the order of appearance in the CDS document.</p>					
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Integer</td><td>This number defines the position of the variable generated for the user input field.</td></tr> </tbody> </table>	Value	Description	Integer	This number defines the position of the variable generated for the user input field.
Value	Description				
Integer	This number defines the position of the variable generated for the user input field.				

Annotation	Meaning												
Consumption.filter.mandatory	To prompt the "Filter UI" to enforce a user entry for a (semantically) mandatory element, even if a default value exists (for example, defined through a CASE expression).												
<p>• Example</p> <p>The user deletes the proposal for a mandatory filter in the Filter UI. The UI then displays an error message "Please enter a value for filter ...". After the user has entered a value , the entered value is then sent to the engine. Therefore, the Filter UI ensures that automatic replacement by the engine is never performed for mandatory filters.</p>													
Value	Description												
Boolean (true, false)													
Consumption.filter.multipleSelections	This annotation indicates the lines that can be entered on filter input (selection) UIs combined with the following selectionType values:												
<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>selectionType SINGLE and multipleSelections = true</td><td>IN list</td></tr> <tr> <td>selectionType INTERVAL and multipleSelections = false</td><td>Single interval</td></tr> <tr> <td>selectionType INTERVAL and multipleSelections = true</td><td>Several intervals</td></tr> <tr> <td>selectionType RANGE and multipleSelections = true</td><td>Several ranges (complete (ABAP-like) SELECT options).</td></tr> <tr> <td>selectionType HIERARCHY_NODE and multipleSelections = true/false</td><td>Single node, several nodes</td></tr> </tbody> </table>		Value	Description	selectionType SINGLE and multipleSelections = true	IN list	selectionType INTERVAL and multipleSelections = false	Single interval	selectionType INTERVAL and multipleSelections = true	Several intervals	selectionType RANGE and multipleSelections = true	Several ranges (complete (ABAP-like) SELECT options).	selectionType HIERARCHY_NODE and multipleSelections = true/false	Single node, several nodes
Value	Description												
selectionType SINGLE and multipleSelections = true	IN list												
selectionType INTERVAL and multipleSelections = false	Single interval												
selectionType INTERVAL and multipleSelections = true	Several intervals												
selectionType RANGE and multipleSelections = true	Several ranges (complete (ABAP-like) SELECT options).												
selectionType HIERARCHY_NODE and multipleSelections = true/false	Single node, several nodes												
Consumption.filter.selectionType	This annotation determines how values can be entered.												
Value	Description												

Annotation	Meaning
SINGLE	Single value
INTERVAL	Special case of a range with sign = including and operator = BT.
RANGE	A range is a complete (ABAP like) SELECT option including sign (including/excluding) and operator (e.g. EQ, CP, BT).
HIERARCHY_NODE	Hierarchy node (means everything under the node)
Consumption.groupWithElement	<p>This annotation enables recognition of View/Entity elements whose values semantically depend on the values of other elements and can only be understood in their context.</p> <p>Scope: #ELEMENT</p> <p>Evaluation Runtime (Engine): This annotation will be interpreted by the SADL.</p> <div style="background-color: #f0f0f0; padding: 10px;"> <p>i Note</p> <p>Whenever such elements are used for querying grouped data, the context-providing elements should also be included in the query's "group by" expression.</p> </div>
Value	Description
elementRef	

Annotation	Meaning				
Consumption.hidden	<p>This annotation prevents fields from being exposed by OData. The annotation, therefore, prevents fields from being available to the client. This is necessary for system parameters because these parameters need to be filled by the runtime engine, but must not be available to the client.</p> <p>Scope: #ELEMENT, #PARAMETER</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL - Prevents fields from being exposed by OData. • Analytic Manager - For #ELEMENT: In the Analytic Query Layer, this annotation is only evaluated if the element represents a measure. Typical usecase: An element is a calculated measure, which serves as an intermediate result and shall not be shown in the Analytic Query result. <p>i Note</p> <p>The field will not be exposed to UIs.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Boolean (true, false)</td> <td>Defines whether fields are available to the client or not. Default: true</td> </tr> </tbody> </table>	Value	Description	Boolean (true, false)	Defines whether fields are available to the client or not. Default: true
Value	Description				
Boolean (true, false)	Defines whether fields are available to the client or not. Default: true				

Annotation	Meaning
Consumption.labelXElement	<p>This annotation enables consumer frameworks (such as UI or analytical clients) to identify through which elements the explanatory texts for the "identifier" elements can be retrieved.</p> <p>In cases where both the <code>identifier</code> elements and the <code>text</code> elements are exposed, the <code>identifier</code> elements are annotated with the <code>labelElement</code> annotation. They then point to the element that contains the label text. A <code>labelElement</code> annotation silently implies a <code>groupWithElement</code> annotation of the "text" element pointing to the "identifier" element.</p> <p>Scope: #ELEMENT, #PARAMETER</p> <p>Evaluation Runtime (Engine): SADL - The referenced element will be handled as descriptive text of the annotated field in OData exposure scenarios.</p>
	Note
	<p>We recommend to use <code>ObjectModel.text.element[]</code> instead of the <code>Consumption.labelXElement</code> annotation.</p>
Value	Description
elementRef	

Annotation	Meaning
Consumption.semanticObject	<p>This annotation enables annotation of SAP-specific business semantics, thus, extending the standardized business semantics covered by the <code>@Semantics</code> domain.</p> <p>Consumers may leverage this enrichment for enhanced interoperability across applications.</p>
• Example	
<p>SAP Fiori has introduced the concept of intent-based navigation, whereby an intent is a combination of <code><semanticObject> <action></code>. A <code>semanticObject</code> annotation is used in SAP Fiori UIs to dynamically derive navigation targets for the annotated view as a source.</p>	
<p>Scope: #ELEMENT, #PARAMETER, #TABLE FUNCTION, #VIEW</p> <p>Evaluation Runtime (Engine): SADL - Translates CDS annotations into the corresponding OData annotations.</p>	
Value	Description
String	For more information Based on Intent [page 329] .

Annotation	Meaning
Consumption.valueHelp	<p>This annotation enables referencing of the association to a CDS view or CDS entity that provides the value help object for the annotated element.</p> <p>The ON condition of the referenced association specifies how the value help content needs to be retrieved for the element. Generic UI consumers have to identify these ValueHelp annotations and use the referenced association to retrieve the values.</p> <p>Instead of defining the "Value element" of the value help view using an equal ON condition of the association, you can also specify the "value element" in the annotation and then have no or non-equal ON conditions for the "value element".</p> <p>Scope: #ELEMENT, #PARAMETER</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL - Derives a default value help support from the annotated association. • Analytic Manager - valid only for scope #PARAMETER.
Value	Description
elementRef	
Consumption.ValueHelpDefintition	<p>Annotations belonging to <i>Consumption.valueHelpDefinition</i> directly establish a relationship to an entity that acts as a value help provider.</p> <p>The value help can be consumed without an association to the target value help provider.</p> <p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine): Interpreted by ABAP Runtime Environment</p>
Consumption.valueHelpDefinition.additionalBinding[]	Defines an additional binding condition for the value help on the same target value help provider entity for filtering the value help result list and/or returning values from the selected value help record.
Value	Description
element	Specifies the element in the target value help provider entity that is linked to the local element or parameter for the additional binding.

Annotation	Meaning	
localElement	Specifies the local element that is linked to the element or parameter in the target value help provider entity for the additional binding.	
localParameter	Specifies the local parameter that is linked to the element or parameter in the target value help provider entity for the additional binding.	
parameter	Specifies the parameter in the target value help provider entity that is linked to the local element or parameter for the additional binding.	
usage	The binding may either specify an additional filter-criterion on the value help list (#FILTER), or an additional result mapping for the selected value help record (#RESULT) or a combination thereof (#FILTER_AND_RESULT). If not specified explicitly the usage is #FILTER_AND_RESULT . If <i>distinctValues</i> is set to true, additional bindings must specify the usage as #FILTER .	
Consumption.valueHelpDefinition.association	Value	Description

Annotation	Meaning						
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>AssociationRef</td><td>Defines the value help to the value help providing entity via an association. The binding condition is then given by the on-condition of the association.</td></tr> </tbody> </table>	Value	Description	AssociationRef	Defines the value help to the value help providing entity via an association. The binding condition is then given by the on-condition of the association.		
Value	Description						
AssociationRef	Defines the value help to the value help providing entity via an association. The binding condition is then given by the on-condition of the association.						
Consumption.valueHelpDefinition.distinctValues	Specifies whether the value help result list shall only contain distinct values for the annotated field or parameter. If set to true all mappings will be used for filtering, but only the value for the field/parameter which the value help was requested for will be returned by the value help.						
Consumption.valueHelpDefinition.entity[]	Defines the binding for the value help to the value help providing entity. It requires specification of the entity and the element providing the value help for the annotated element. <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>element</td><td>Specifies the element in the entity referenced in name that provides the value help for the annotated element.</td></tr> <tr> <td>name</td><td>Specifies the entity which contains the element that provides the value help.</td></tr> </tbody> </table>	Value	Description	element	Specifies the element in the entity referenced in name that provides the value help for the annotated element.	name	Specifies the entity which contains the element that provides the value help.
Value	Description						
element	Specifies the element in the entity referenced in name that provides the value help for the annotated element.						
name	Specifies the entity which contains the element that provides the value help.						
Consumption.valueHelpDefinition.label	This annotation contains a language-dependent text that is used to label the value list. If not specified the label of the value help defining entity is used.						
Consumption.valueHelpDefinition.presentationVariantQualifier	The Presentation Variant indicates how the value help result should be displayed.						
Consumption.valueHelpDefinition.qualifier	Uniquely identifies alternative values for an annotation. Omission means the OData term is applied without explicit qualifier. If more than one value help is defined for one element, a qualifier must be used.						

Examples

Example 1

The annotations `Consumption.labelXElement` and `Consumption.quickInfo` are used to link the corresponding text and ID fields:

↳ Sample Code

```
DEFINE VIEW SalesOrder
AS SELECT FROM sepm_cds_sales_order AS so
{
    key so.sales_order_id AS SalesOrderId,
    @Consumption.labelXElement: 'CompanyName'
    @Consumption.quickInfoElement: 'CompanyDescription'
    so.buyer_guid AS BuyerGuid,
    so.company_name AS CompanyName,
    so.description AS CompanyDescription
}
```

Example 2

The annotation `Consumption.groupWithElement` is used to define that the element `CompanyName` depends semantically on the element `BuyerGuid`. In the case of an aggregated table, the fields can be displayed combined:

↳ Sample Code

```
DEFINE VIEW SalesOrder with parameters
AS SELECT FROM sepm_cds_sales_order AS so
{
    key so.sales_order_key AS NodeKey,
    so.buyer_guid AS BuyerGuid,
    @Consumption.groupWithElement: 'BuyerGuid'
    so.company_name AS CompanyName,
    so.currency_code AS CurrencyCode,
    @DefaultAggregation: #SUM
    so.gross_amount AS GrossAmount
}
```

Example 3

The annotation `Consumption.hidden` is used to prevent fields from being exposed by OData. The annotation, therefore, prevents fields from being available to the client. This is necessary for system parameters because these parameters need to be filled by the runtime-engine, but must not be available to the client:

↳ Sample Code

```
DEFINE VIEW SalesOrder
AS SELECT FROM sepm_cds_sales_order AS so
    ASSOCIATION [0..1] TO sepm_cds_business_partner AS _Customer
    on $projection.BuyerGuid = _Customer.business_partner_key
{
```

```

key so.sales_order_id AS SalesOrderId,
@Consumption.hidden: true
so.buyer_guid AS BuyerGuid,
Customer.company_name AS CompanyName
}

```

Example 4

The annotation `Consumption.defaultValue` is used to specify default values for parameters. In this example, the currency EUR is used:

↳ Sample Code

```

DEFINE VIEW SalesOrder with parameters
  @Consumption.defaultValue: 'EUR'
  p_TargetCurrency : snwd_curr_code
AS SELECT FROM sepm_cds_sales_order AS so
{
  key so.sales_order_key AS NodeKey,

  $parameters.p_TargetCurrency AS CurrencyCode,
  currency_conversion(
    amount          => so.gross_amount,
    source_currency => so.currency_code,
    target_currency  => $parameters.p_TargetCurrency,
    exchange_rate_date => CAST( '20150101' AS abap.dats )
  ) AS GrossAmount
}

```

Example 5

The annotation `Consumption.valueHelp` is used to expose an association as a value help. In this example, the CDS view `CurrencyCodeValueHelp` is used for the field `CurrencyCode`:

The annotation `Consumption.valueHelpDefinition` is used to define a value help for the annotated element. The value help provider can be a different CDS entity without association. To consume the value help, the value help provider entity must be added to the respective OData service.

You can filter the available value help options by defining an additional binding. In the following example case, only the business partners are displayed that use the same currency code.

↳ Sample Code

```

DEFINE VIEW BuPaView AS SELECT FROM db_bp
{
  key bp.bp_id                               as BusinessPartnerID,
  ...
  bp.currency_code                          as CurrencyCode,
  bp.company_name                           as CompanyName,
}
DEFINE VIEW SOview AS SELECT FROM db_so as so
{
  so.sales_order_id as SalesOrderID,
  ...
  so.CurrencyCode as CurrCode,
  @Consumption.valueHelpDefinition: [{ entity : { name      :
'I_AIVS_BusinessPartner',
                                              element :
'BusinessPartnerID'}, ...
}

```

```

        additionalBinding : [
            { localElement : 'CurrCode',
              element : 'CurrencyCode'
            }
        ]
    _BusinessPartner.BusinessPartnerID
}

```

Example 6

The association `Consumption.semanticObject` is used to assign semantic objects to CDS views or elements that can be used in the UI for intent-based navigation.

↳ Sample Code

```

@Consumption.semanticObject: 'SalesOrder'
DEFINE VIEW SalesOrder
AS SELECT FROM sepm_cds_sales_order AS so
{
    so.sales_order_id AS SalesOrderId,
    @Consumption.semanticObject: 'BusinessPartner'
    so.buyer_guid AS BuyerGuid,
}

```

Example 7

The annotation `Consumption.filter` is used to enable filters for values. In this example, `product_hierarchy` is filtered for several single values of `calendar_day`, and for one single interval of `product`.

↳ Sample Code

```

@Analytics.query : true
DEFINE VIEW product_hierarchy
AS SELECT FROM sales
{
    ...
    @Consumption.filter : { selectionType : #SINGLE, multipleSelections : true }
        calendar_day,      //several single values are allowed
        @Consumption.filter : { selectionType : #INTERVAL, multipleSelections : false }
            product,          // single interval allowed
            ...
}

```

Example 8

The annotation `Consumption.filter` is used to enable filters for values. In this example, `costcenter_reporting` is filtered hierarchically for the constant `CONTR_AREA_10` and for the parameter `cost_center_hier_param`.

↳ Sample Code

```

@Analytics.query : true
DEFINE VIEW costcenter_reporting
    with parameters cost_center_hier_param : String
AS SELECT FROM costcenters
{
    ...
}

```

```

@Consumption.filter : {
    selectionType : #HIERARCHY_NODE,
    multipleSelections : false,
    hierarchyBinding :
        [ { type : #CONSTANT, value : 'CONTR_AREA_10' },
          { type : #PARAMETER, value : 'cost_center_hier_param' } ]
}
costcenter,      // hierarchy node filter

costs,
...
}

```

Example 9

This example shows how to specify hierarchy nodes as **default value** in the Consumption.filter annotation. A consumption view should provide a prompt for a hierarchy node of a costcenter hierarchy with default 1000/ALL (HierarchyNode).

↳ Sample Code

```

@Consumption.filter{ selectionType: #HIERARCHY_NODE ;
                     defaultHierarchyNode: { nodeType :
                     'HierarchyNode' ;
                     'ControllingArea' ; value: '1000' } ;
                     node: [ { element:
                     'HierarchyNode' ; value: 'ALL' } ] };
                     hierarchyBinding: [ ... ] );
costCenter;

```

Example 10

This example shows how to enable the Consumption.derivation annotation to return hierarchy nodes. The annotation Consumption.derivation.resultHierarchyNode is used to return hierarchy nodes. Given a view CostcenterResponsible with fields field1, field2, field3, field4, and responsible. The query view should filter by the costcenter nodes the given responsible (parameter) is responsible for.

↳ Sample Code

```

@Consumption.filter: { selectionType: #HIERARCHY_NODE ;
                      hierarchyBinding: [ ... ] };
@Consumption.derivation.resultHierarchyNode:
    { lookupEntity: 'CostcenterResponsible';
      nodeTypeElement: 'field1';
      mapping: [ { hierarchyElement : 'ControllingArea' ;
      lookupElement: 'field2' } ;
                  { hierarchy Element: 'CostCenter' ; lookupElement:
      'field3' } ;
                  { hierarchyElement: 'HierarchyNode' ; lookupElement:
      'field4' } ] };
@Consumption.hidden: true;
costCenter;

```

The hierarchy binding is derived from Consumption.filter.hierarchyBinding. Consumption.derivation can only be used in combination with Consumption.filter. Furthermore Consumption.filter.selectionType: #HIERARCHY_NODE has to be true.

Example 11

This example shows how to define and use access policy with PFCG-mapping:

↳ Sample Code

```
// Access policy with PFCG-mapping
define accesspolicy I_ProfitCenterHierAuth {
    pfcg_mapping I_ProfitCenterHierAuth between I_ProfitCenterHierarchyNode and
    K_PCA_HIER using I_ProfitCenterHierSubTree
    {
        ProfitCenterHierarchy      = PCTRHIER,
        ControllingArea           = KOKRS,
        HierarchyNode              = PCTRHIERND
    }
}

// Analytic Query
@Analytics.query: true
define view ...
with parameters
    @Consumption.derivation: { pfcgMapping: 'I_PROFITCENTERHIERAUTH' ,
                                resultElement: 'ProfitCenterHierarchy' }
    P_ProfitCenterHierarchy : fis_hryid_prctr,
    @Consumption.derivation: { pfcgMapping: 'I_PROFITCENTERHIERAUTH' ,
                                resultElement: 'ControllingArea' }
    P_ControllingArea : fis_kokrs,
as select from ...
{
    ...
    @AnalyticsDetails.query.axis: #ROWS
    @Consumption.filter:
    { selectionType : #HIERARCHY_NODE,
        multipleSelections: true,
        hierarchyBinding: [ { type: #PARAMETER, value: 'P_ControllingArea' }
                            { type: #PARAMETER, value:
                                'P_ProfitCenterHierarchy' } ]
    }
    @Consumption.derivation.pfcgMapping: 'I_PROFITCENTERHIERAUTH'
    @AnalyticsDetails.query.displayHierarchy: #FILTER
    ProfitCenter,
    ...
}
```

9.1.5 Aggregation Annotations

Specifies aggregation behavior at element level

Scope and Definition

```
@Scope:[#ELEMENT]
annotation Aggregation
{
    default: String(30) enum
    {
```

```

        NONE;
        SUM;
        MIN;
        MAX;
        AVG;
        COUNT_DISTINCT;
        NOP;
        FORMULA;
    };
exception : String( 30) enum
{
    SUM;
    MIN;
    MAX;
    AVG;
    COUNT;
    COUNT_DISTINCT;
    FIRST;
    LAST;
    STANDARD_DEVIATION;
    VARIANCE;
    MEDIAN;
    NHA;
};
referenceElement : array of ElementRef;
};

```

Usage

With this annotation, you can specify the aggregation behaviour of elements in generic usage like analytic manager or ODATA. Elements without default aggregation or with `Aggregation.default: #NONE` will not be aggregated and will be used in `GROUP BY` for aggregating access. Elements that can be aggregated are known as [measures](#).

Annotation	Meaning								
<code>Aggregation.default</code>	Scope: [ELEMENT] Evaluation Runtime (Engine): This annotation will be interpreted by the analytic manager . Use all values for this annotation only for fields of numeric type. Exception is <code>NONE</code> : It has to be used for all strings like elements. Values: <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>SUM</td><td>The total (sum) of all values in this column is shown in the result line.</td></tr> <tr> <td>MAX</td><td>The highest value of all calculated values in this column is shown in the result line.</td></tr> <tr> <td>MIN</td><td>The lowest value of all calculated values in this column is shown in the result line.</td></tr> </tbody> </table>	Value	Description	SUM	The total (sum) of all values in this column is shown in the result line.	MAX	The highest value of all calculated values in this column is shown in the result line.	MIN	The lowest value of all calculated values in this column is shown in the result line.
Value	Description								
SUM	The total (sum) of all values in this column is shown in the result line.								
MAX	The highest value of all calculated values in this column is shown in the result line.								
MIN	The lowest value of all calculated values in this column is shown in the result line.								

Annotation	Meaning
AVG	<p>The average of all values is calculated.</p> <p>i Note</p> <p>The analytic manager allows AVG only together with Aggregation.referenceElement and exactly one element in this list. Aggregation: { default: #AVG , referenceElement: <element> } behaves like Aggregation: { default: #SUM, exception: #AVG , referenceElement: <element> }.</p>
COUNT	<p>The number of rows is calculated.</p> <p>i Note</p> <p>The analytic manager allows COUNT only together with Aggregation.referenceElement and exactly one element in this list. Aggregation: { default: #COUNT , referenceElement: <element> } behaves like Aggregation: { default: #SUM, exception: #COUNT_DISTINCT , referenceElement: <element> }.</p>
COUNT_DISTINCT	<p>The number of distinct values is calculated.</p> <p>i Note</p> <p>The analytic manager allows COUNT_DISTINCT only together with Aggregation.referenceElement and exactly one element in this list. Aggregation: { default: #COUNT_DISTINCT , referenceElement: <element> } behaves like Aggregation: { default: #SUM, exception: #COUNT_DISTINCT , referenceElement: <element> }.</p>

Annotation	Meaning										
	<p>FORMULA</p> <p>The value FORMULA indicates that the element is a formula. This has to be calculated after the operands have been determined by aggregation or calculation. It should never be aggregated.</p> <p>i Note</p> <p>This can only be used in views with <code>Analytics.query: true</code>.</p> <p>• Example</p> <p>Margin := Revenue / Cost. If Margin should be shown per OrgUnit in a report, the aggregates of Revenue and Cost have to be determined per OrgUnit first, and then the Margin has to be calculated per OrgUnit.</p>										
	<p>NONE</p> <p>This value indicates that the element is not a measure. Usually these elements are used in filters and GROUP BY statements.</p>										
<code>Aggregation.exception</code>	<p>Exception aggregation is always performed in addition to default aggregation, which must not be equal to #NONE. This is not an alternative to default aggregation. This means that data is first aggregated through the default aggregation grouped by the reference elements, and then aggregated with the exception aggregation.</p> <p>i Note</p> <p>If you want to use <code>Aggregation.exception</code>, you have to define <code>Aggregation.default</code> (not equal to NONE) and <code>Aggregation.referenceElement</code> in order to define the granularity with which the aggregation rule is applied.</p>										
	<p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine): This annotation will be interpreted by the analytic manager.</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>SUM, MAX, MIN, AVG, COUNT, COUNT_DISTINCT</td><td>All these values determine the aggregation of the measure in the same way as SUM, MAX, MIN, AVG, COUNT, COUNT_DISTINCT in <code>Aggregation.default</code>.</td></tr> <tr> <td>FIRST, LAST</td><td>These values indicate that the first or last value in relation to the reference characteristic is shown in the result line.</td></tr> <tr> <td>STANDARD_DEVIATION</td><td>This value indicates that, after aggregating with the reference characteristic, the standard deviation of the calculated values is shown in the results row.</td></tr> <tr> <td>VARIANCE</td><td>This value indicates that, after aggregating with the reference characteristic, the variance of the calculated values is shown in the results row.</td></tr> </tbody> </table>	Value	Description	SUM, MAX, MIN, AVG, COUNT, COUNT_DISTINCT	All these values determine the aggregation of the measure in the same way as SUM, MAX, MIN, AVG, COUNT, COUNT_DISTINCT in <code>Aggregation.default</code> .	FIRST, LAST	These values indicate that the first or last value in relation to the reference characteristic is shown in the result line.	STANDARD_DEVIATION	This value indicates that, after aggregating with the reference characteristic, the standard deviation of the calculated values is shown in the results row.	VARIANCE	This value indicates that, after aggregating with the reference characteristic, the variance of the calculated values is shown in the results row.
Value	Description										
SUM, MAX, MIN, AVG, COUNT, COUNT_DISTINCT	All these values determine the aggregation of the measure in the same way as SUM, MAX, MIN, AVG, COUNT, COUNT_DISTINCT in <code>Aggregation.default</code> .										
FIRST, LAST	These values indicate that the first or last value in relation to the reference characteristic is shown in the result line.										
STANDARD_DEVIATION	This value indicates that, after aggregating with the reference characteristic, the standard deviation of the calculated values is shown in the results row.										
VARIANCE	This value indicates that, after aggregating with the reference characteristic, the variance of the calculated values is shown in the results row.										

Annotation	Meaning
	MEDIAN
	This value indicates that the middle value (central value) of a set sorted in ascending order is calculated: <ul style="list-style-type: none"> For a set with an uneven number of elements, the result is the middle value.
	<p>Example</p> <p>For a set of five elements, the result is the value of the third element. Suppose you have the following values: 1, 12, 2, 4, 24. When the values are sorted in ascending order (1, 2, 4, 12, 24), the median is the third middle value, which is 4.</p> <ul style="list-style-type: none"> For a set with an even number of elements, the result is the average of the two middle values.
	<p>Example</p> <p>For a set of six elements, the result is the average of the third and fourth elements. Suppose you have the following values: 1, 12, 2, 4, 24, 6. When the values are sorted in ascending order (1, 2, 4, 6, 12, 24), the median is the average of the values 4 and 6, which is 5.</p>
NHA	No aggregation along the hierarchy. This value ensures that nodes with a postable value part, and which only aggregate the values of their children, are calculated as NULL. This means they have no value if they are not postable nodes.
Aggregation.referenceElement	In views of dataCategory: #CUBE or #DIMENSION you can use exactly one element. In analytic queries (Analytics.query: true) you can use up to five elements. The elements in the list cannot be measures.
	Scope: [ELEMENT]
	Evaluation Runtime (Engine): This annotation will be interpreted by the analytic manager .
Value	Description
array of ElementRef	You can specify a reference characteristic.

Examples

Example 1: default Aggregation

Sample Code

```
@Semantics.currencyCode:true
GlobalCurrency,
@Aggregation.default: #SUM
@Semantics: { amount : {currencyCode: 'GlobalCurrency'} }
FixedAmountInGlobalCrcy,
```

When selecting FixedAmountInGlobalCrcy the SUM of this element is calculated.

Example 2: exception aggregation

↳ Sample Code

```
calendarDay,  
@Semantics.unitOfMeasure: true  
unit,  
@Aggregation.default: #SUM  
@Aggregation.exception: #LAST  
@Aggregation.referenceElement: 'calenderDay'  
@Semantics.quantity : 'unit'  
Quantity
```

This view describes the quantity of certain goods in a stock per calendar day. The measure quantity can be aggregated with SUM for all dimensions except the calendar day.

Non-aggregated data

calendarDay	Plant	Material	Quantity
01/01/2018	Plant1	Material1	10 PC
02/01/2018	Plant1	Material1	20 PC
01/01/2018	Plant2	Material1	30 PC
02/01/2018	Plant2	Material1	10 PC
01/01/2018	Plant1	Material2	50 PC
02/01/2018	Plant1	Material2	40 PC
01/01/2018	Plant2	Material2	40 PC
02/01/2018	Plant2	Material2	70PC

Aggregate Quantity and group by Material and CzalendarDay

Material	calendarDay	Quantity
Material1	01/01/2018	40 PC
Material1	02/01/2018	30 PC
Material1	Result	30 PC
Material2	01/01/2018	90 PC
Material2	02/01/2018	110 PC
Material2	Result	110 PC
Result	Result	140PC

Example 3: Counter-optimized

For CDS views representing master data reports (the FROM clause refers to a CDS view categorized as #DIMENSION), a counter of the different dimension values can be defined by combining the annotation Aggregation: #SUM with a select list entry containing constant value 1:

↳ Sample Code

```
@EndUserText.label: 'CostCenters per ControllingArea'
```

```

@Analytics.query: true
define view ControllingAreaView
as SELECT from CostCenterDimension {
controllingArea,
@Aggregation: #SUM
1 as costCenterCount
}

```

With costCenter in the key of the dimension, no exception aggregation is involved, thus improving the performance of the report.

i Note

This pattern can be combined with CAST statements (for reusing texts of DDIC data types) and/or CASE statements (for defining restricted measures). You must not use other constants than 1 however, or add annotation AnalyticsDetails.query.formula.

Related Information

[AnalyticsDetails Annotations \[page 379\]](#)

9.1.6 EnterpriseSearch Annotations

i Note

These annotations are currently **only available for SAP-internal projects** and not released for customer projects.

Note that SAP might change the behavior of this annotation in future. Consequently, functionality might change. Therefore, usage is on your own responsibility for customer projects. SAP recommends not to use these CDS annotations in customer projects.

Scope and Definition

```

@Scope:[#VIEW, #TABLE_FUNCTION, #ENTITY]
Annotation EnterpriseSearch
{
    enabled : Boolean default true;
};

@Scope:[#ELEMENT]
Annotation EnterpriseSearch
{
    expand : Boolean default true;
    filteringFacet : { default };
    defaultValueSuggestElement : true;
}

```

```

usageMode : array of String(20) enum
{
    ADVANCED_SEARCH = 'AdvancedSearch';
    AUTO_FACET = 'AutoFacet';
    SUGGESTION = 'Suggestion';
};
presentationMode : array of String(20) enum
{
    DETAIL = 'Detail';
    HIDDEN = 'Hidden';
    IMAGE = 'Image';
    SUMMARY = 'Summary';
    THUMBNAIL = 'Thumbnail';
    TITLE = 'Title';
    NONE = 'None';
};
commonAttributes : array of String(100);
snippets
{
    enabled : Boolean default true;
    beginTag : String(128) default '<b>';
    endTag : String(128) default '</b>';
};
highlighted
{
    enabled : Boolean default true;
    beginTag : String(128) default '<b>';
    endTag : String(128) default '</b>';
};

```

Usage

Annotation	Meaning
EnterpriseSearch.enabled	Defines if a CDS view is generally relevant for search scenarios based on SAP HANA Enterprise Search.
i Note	
EnterpriseSearch annotations require the annotation @Search.searchable for the same view.	
Scope: #View	
Evaluation Runtime (Engine): Interpreted by Enterprise Search	
Values:	
Value	Description
<i>Boolean (true, false)</i>	Defines whether a CDS view is relevant for Enterprise Search or not. If it is set to true a search connector is created in Enterprise Search automatically. Default: true

Annotation	Meaning	
EnterpriseSearch.filteringFacet	<p>Specifies that the element is to be considered as a request field which is used for faceted search (also named interactive navigation or guided navigation).</p> <p>Scope: #Element</p> <p>Evaluation Runtime (Engine): Interpreted by Enterprise Search</p> <p>Values:</p>	
	Value	Description
{ default }		Defines whether the element is to be considered as a request field in faceted search.
		Default: { default }
EnterpriseSearch.defaultValueSuggestElement	<p>Specifies that the element is to be considered for suggestions (also named type-ahead or auto-completion function).</p> <p>Scope: #Element</p> <p>Evaluation Runtime (Engine): Interpreted by Enterprise Search</p> <p>Values:</p>	
	Value	Description
Boolean (true, false)		Defines whether the element is to be considered for suggestions.
		Default: true

Related Information

[Search Annotations \[page 442\]](#)

9.1.7 Hierarchy Annotations

Enables an application developer to specify a parent-child hierarchy that s/he wants to make explicitly accessible in a data model, together with the structure that defines this hierarchy.

Scope and Definition

```
@Scope: [#VIEW]
Annotation Hierarchy
{
  parentChild : array of
```

```

{
  name : String(127);
  label : String;
  multipleParents : Boolean default true;
  recurseBy : elementRef;
  recurse
  {
    parent : array of elementRef;
    child : array of elementRef;
  };
  siblingsOrder : array of
  {
    by : elementRef;
    direction : String(4) enum { ASC = 'ASC'; DESC = 'DESC'; } default #ASC;
  };
  rootNode
  {
    visibility : String(25) enum { ADD_ROOT_NODE_IF_DEFINED; ADD_ROOT_NODE;
DO_NOT_ADD_ROOT_NODE; } default #ADD_ROOT_NODE_IF_DEFINED;
  };
  orphanedNode
  {
    handling : String(20) enum { ROOT_NODES; ERROR; IGNORE;
STEPPARENT_NODE; } default #ROOT_NODES;
    stepParentNodeId : array of String;
  };
  directory : associationRef;
};
};

```

Usage

The parent-child hierarchy is based directly on the master data entities. The hierarchy is time-dependent if the master data entity is as well. The hierarchy is defined either by one parent element (with `Hierarchy.parentChild.recurseBy`) or by multiple parents (with `Hierarchy.parentChild.recurse`). A parent element describes a self-referencing relationship within the master data entity and will usually be defined via an association. Only one level needs to be assigned to a parent-child hierarchy, because the levels in the hierarchy are taken from the parent-child relationships between members associated with the parent element. One or more parent-child hierarchies can be defined within the same master data entity.

i Note

A simple example of a parent-child hierarchy is the “Employee” master data. A “Manager” is an “Employee” and almost every “Employee” is assigned to a “Manager”.

On entity level the following metadata can be defined:

Annotation	Meaning
Hierarchy.parentChild.directory	<p>For external hierarchies, the view of hierarchy nodes often contains the nodes for multiple alternative hierarchies. A user is supposed to choose a single hierarchy for display, and his/her selection is used as a filter when selecting from the hierarchy node view. The directory annotation identifies an association, the hierarchy directory association, from the hierarchy node view to a view (the so-called hierarchy directory), providing all available alternative hierarchies for this hierarchy node view.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Analytic manager: The hierarchy directory is used as user input help, and a chosen hierarchy directory entry is used to filter the nodes via the hierarchy directory association.</p>
Value	Description
associationRef	Name of the hierarchy directory that is used to filter the nodes
Hierarchy.parentChild.label	<p>User-friendly name of the hierarchy.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Analytic manager: This name can be exposed in input help for <code>Hierarchy.parentChild.name</code>.</p> <p>Values (optional):</p>
Value	Description
String	Description of the hierarchy.
Hierarchy.parentChild.multipleParents	<p>Indicates that multiple parents might occur in the hierarchy.</p> <p>i Note</p> <p>In a geographic hierarchy for example, you might want to assign the country Turkey to the continents Europe and Asia.</p> <p>△ Caution</p> <p>We need to distinguish the above use case from the following one: In a time hierarchy YEAR, QUARTER, MONTH January under 2011 and January under 2012 are not the same member with multiple parents. They are different Januaries.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): The analytic manager can handle hierarchies with multiple roots. This annotation is only relevant for engines that cannot handle multiple roots.</p>
Value	Description

Annotation	Meaning								
	<table> <tr> <td>Boolean (true, false)</td><td>Defines whether multiple parents occur in the hierarchy or not.</td></tr> <tr> <td></td><td>Default: true</td></tr> </table>	Boolean (true, false)	Defines whether multiple parents occur in the hierarchy or not.		Default: true				
Boolean (true, false)	Defines whether multiple parents occur in the hierarchy or not.								
	Default: true								
Hierarchy.parentChild.name	<p>Technical name of the hierarchy. Only relevant if the view defines one hierarchy only, and <code>Hierarchy.parentChild.directory</code> is not used.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Analytic manager: The hierarchy is accessed in queries or in program logic later on via this name.</p>								
	Value Description								
	<table> <tr> <td>String</td><td>Technical name of the hierarchy.</td></tr> </table>	String	Technical name of the hierarchy.						
String	Technical name of the hierarchy.								
Hierarchy.parentChild.orphanedNode.handling	<p>Defines how nodes without a parent (more precisely with a parent that does not occur as a child) are processed.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Depending on the annotation value, the analytic manager will ignore orphaned nodes, treat them as root nodes, put them under a step parent node or raise an error.</p>								
	Value Description								
	<table> <tr> <td>ROOT_NODES</td><td>Treat nodes as root nodes (default).</td></tr> <tr> <td>ERROR</td><td>Stop processing and show an error.</td></tr> <tr> <td>IGNORE</td><td>Ignore nodes and remove them from the hierarchy.</td></tr> <tr> <td>STEPPARENT_NODE</td><td>Put nodes under a step parent node</td></tr> </table>	ROOT_NODES	Treat nodes as root nodes (default).	ERROR	Stop processing and show an error.	IGNORE	Ignore nodes and remove them from the hierarchy.	STEPPARENT_NODE	Put nodes under a step parent node
ROOT_NODES	Treat nodes as root nodes (default).								
ERROR	Stop processing and show an error.								
IGNORE	Ignore nodes and remove them from the hierarchy.								
STEPPARENT_NODE	Put nodes under a step parent node								
Hierarchy.parentChild.orphanedNode.stepParentNodeId	<p>Defines how nodes without a parent (more precisely with a parent that does not occur as a child) are processed.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Analytic manager:</p> <ul style="list-style-type: none"> With the annotation <code>Hierarchy.parentChild.orphanedNode.handling : STEPPARENT_NODE</code> this annotation contains the node ID(s) of the step parent node(s). With parent-child hierarchies, you need to define a step parent node ID for each component (each parent-child combination). 								
	Value Description								
	<table> <tr> <td>stepParentNo deId</td><td>node ID(s) of the step parent node(s)</td></tr> </table>	stepParentNo deId	node ID(s) of the step parent node(s)						
stepParentNo deId	node ID(s) of the step parent node(s)								

Annotation	Meaning						
Hierarchy.parentChild.recurse. child	If the underlying view definition does not contain an association defining the parent-child relationship but only “normal” elements, this annotation has to be used to define the children.						
	<p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Analytic manager</p>						
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>array of elementRef;</td><td>The element names define the key elements of the “child”.</td></tr> </tbody> </table>	Value	Description	array of elementRef;	The element names define the key elements of the “child”.		
Value	Description						
array of elementRef;	The element names define the key elements of the “child”.						
Hierarchy.parentChild.recurse. parent	If the underlying view definition does not contain an association defining the parent-child relationship but only “normal” elements, this annotation has to be used to define the parents.						
	<p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Analytic manager</p>						
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>array of elementRef;</td><td>The element names define the key elements of the “parent”.</td></tr> </tbody> </table>	Value	Description	array of elementRef;	The element names define the key elements of the “parent”.		
Value	Description						
array of elementRef;	The element names define the key elements of the “parent”.						
Hierarchy.parentChild.recurseB y	Defines the parent-child relationship in a hierarchy based on an existing association from hierarchy node to its parent node in the same view.						
	<p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Analytic manager</p>						
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>elementRef</td><td>The name of the parent-child association needs to be specified here.</td></tr> </tbody> </table>	Value	Description	elementRef	The name of the parent-child association needs to be specified here.		
Value	Description						
elementRef	The name of the parent-child association needs to be specified here.						
Hierarchy.parentChild.rootNode .visibility	Using this annotation, you can define dedicated metadata for how to handle root node(s) in the hierarchy.						
	<p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): Depending on the annotation value, the analytic manager might add an extra root node to the hierarchy. The label of this node is the value of <code>Hierarchy.parentChild.label</code>.</p>						
	<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>ADD_ROOT_NOD E_IF_DEFINED</td><td>The system will add the root node of the hierarchy if it is explicitly defined, but the system will not add an extra artificial root node. This is the default.</td></tr> <tr> <td>ADD_ROOT_NOD E</td><td>The system will always add an artificial single root node to the hierarchy. All other nodes are descendants of this node.</td></tr> </tbody> </table>	Value	Description	ADD_ROOT_NOD E_IF_DEFINED	The system will add the root node of the hierarchy if it is explicitly defined, but the system will not add an extra artificial root node. This is the default.	ADD_ROOT_NOD E	The system will always add an artificial single root node to the hierarchy. All other nodes are descendants of this node.
Value	Description						
ADD_ROOT_NOD E_IF_DEFINED	The system will add the root node of the hierarchy if it is explicitly defined, but the system will not add an extra artificial root node. This is the default.						
ADD_ROOT_NOD E	The system will always add an artificial single root node to the hierarchy. All other nodes are descendants of this node.						

Annotation	Meaning
DO_NOT_ADD_Root_NODE	The system will not add an artificial single root node to the hierarchy.
Hierarchy.parentChild.siblingsOrder.by	<p>Hierarchy.parentChild.siblingsOrder defines the order of values with the same parent.</p> <p>Hierarchy.parentChild.siblingsOrder.by defines the element name, which contains the values to be ordered.</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): The analytic manager sorts all children of a node in the given order.</p>
	Value Description
	elementRef element name
Hierarchy.parentChild.siblingsOrder.direction	<p>Hierarchy.parentChild.siblingsOrder defines the order of values with the same parent.</p> <p>Hierarchy.parentChild.siblingsOrder.direction defines if the sort order of values with the same parent is "ascending" or "descending".</p> <p>Scope: #VIEW</p> <p>Evaluation Runtime (Engine): The analytic manager sorts all children of a node in the given order.</p>
	Value Description
	ASC The sort order is ascending (default).
	DESC The sort order is descending.

Examples

Example 1

Parent Child Hierarchy for Views with Associations:

Sample Code

```
@Analytics : { dataCategory : #DIMENSION, replicationEnabled }
@Hierarchy.parentChild : [ { name : 'Organisation', recurseBy : 'Manager' } ]
entity Employee {
  key ID : String(8);
  Manager : Association to this;
  ...
};
```

Example 2

Parent Child Hierarchy for Views without Associations:

↳ Sample Code

```
@Analytics : { dataCategory : #DIMENSION, replicationEnabled : true }
@Hierarchy.parentChild : [ { name : 'Organisation',
    recurse : { parent : [ 'Manager' ], child : [ 'ID' ] }
  }
]
entity Employee {
  key ID : String(8);
  Manager : String(8);
  ...
};
```

Example 3

Hierarchy Directory Association:

↳ Sample Code

```
@Analytics : { dataCategory : #HIERARCHY }
@Hierarchy.parentChild : [
  recurseBy : 'ParentNode',
  directory: 'CostCenterHierarchyDirectory'
]
entity CostCenterHierarchyNode {
  key CostCenterHierarchyDirectory : Association to
  CostCenterHierarchyDirectory;
  key HierarchyNode : CostCenterHierarchyNode;
  ParentNode : Association to this;
  ...
};
```

9.1.8 ObjectModel Annotations

Provide definitions of structural as well as transactional related aspects of the business data model

Scope and Definition

```
@Scope:[#VIEW]
Annotation ObjectModel
{
  lifecycle {
    processor:
    {
      expiryBehavior : String(30) enum { RELATIVE_TO_PROCESSING_START;
      RELATIVE_TO_LAST_CHANGE; } default #RELATIVE_TO_LAST_CHANGE;
      expiryInterval : String(20) default 'PT15M';
      notificationBeforeExpiryInterval : String(20) default 'PT5M';
    };
    enqueue
  }
}
```

```

        expiryBehavior : String enum { RELATIVE_TO_ENQUEUE_START;
RELATIVE_TO_LAST_CHANGE; } default #RELATIVE_TO_LAST_CHANGE;
        expiryInterval : String;
        notificationBeforeExpiryInterval : String;
    };
    processing
    {
        expiryBehavior : String enum { RELATIVE_TO_PROCESSING_START;
RELATIVE_TO_LAST_CHANGE; } default #RELATIVE_TO_LAST_CHANGE;
        expiryInterval : String;
        notificationBeforeExpiryInterval : String;
    };
    draft:
    {
        expiryBehavior : String enum { RELATIVE_TO_PROCESSING_START;
RELATIVE_TO_LAST_CHANGE; } default #RELATIVE_TO_LAST_CHANGE;
        expiryInterval : String default 'P28D';
        notificationBeforeExpiryInterval : String(20) default 'P10D';
    };
    modelCategory : String enum { BUSINESS_OBJECT; };
    dataCategory : String enum { TEXT; HIERARCHY; };
    representativeKey : keyElementRef;
    semanticKey : array of elementRef;
    compositionRoot : Boolean default true;
    transactionalProcessingEnabled : Boolean default true;
    transactionalProcessingDelegated : Boolean default true;
    createEnabled : Boolean default true;
    updateEnabled : Boolean default true;
    deleteEnabled : Boolean default true;
    writeDraftPersistence : String;
    writeActivePersistence : String;
    entityChangeStateId : elementRef;
};
@Scope:[#ELEMENT]
Annotation ObjectModel
{
    association
    {
        type : array of String enum { TO_COMPOSITION_CHILD; TO_COMPOSITION_PARENT;
TO_COMPOSITION_ROOT; };
    };
    createEnabled : Boolean default true;
    usageType { serviceQuality : String(30) enum {A; B; C; D; X;} default 'X';
sizeCategory : String(3) enum {S; M; L; XL; XXL;} default 'S';
dataClass : String(30) enum {TRANSACTIONAL;
MASTER;
ORGANIZATIONAL;
CUSTOMIZING;
META;
MIXED;} default 'MIXED';
    };
    text
    {
        element : array of elementRef;
        association : associationRef;
    };
    hierarchy
    {
        association : associationRef;
    };
    foreignKey
    {
        association : associationRef;
    };
    readOnly : Boolean default true;
}

```

```

        mandatory : Boolean default true;
        enabled : Boolean default true;
    };

```

Usage

Annotation	Meaning								
ObjectModel. association. type[]	<p>Defines the association type that is used for defining a compositional view hierarchy.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> SADL: Influences the scope of the OData auto-exposure (@OData.publish:true): All views that are included in the view hierarchy are automatically included in the same OData V2-service. BOPF: Influences the scope of the BOPF Business Object generation (@ObjectModel.transactionalProcessingEnabled : true): All views that are included in the view hierarchy are automatically included in the same BOPF business object. <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>#TO_COMPOSIT ION_CHILD</td><td>Within the compositional hierarchy there must be a #TO_COMPOSITION_CHILD association leading to each direct sub view of a given view.</td></tr> <tr> <td>#TO_COMPOSIT ION_PARENT</td><td>Views that do not represent the root of the hierarchy must have an association to their compositional parent view annotated with #TO_COMPOSITION_PARENT.</td></tr> <tr> <td>#TO_COMPOSIT ION_ROOT</td><td>Additionally, views that do not represent the root must have a #TO_COMPOSITION_ROOT association for performance reasons (for example: authorization checks (instance restrictions) based of information from root views).</td></tr> </tbody> </table> <p>NOTE: In such a case, the root view of the hierarchy must be annotated with @ObjectModel.compositionRoot: true.</p>	Value	Description	#TO_COMPOSIT ION_CHILD	Within the compositional hierarchy there must be a #TO_COMPOSITION_CHILD association leading to each direct sub view of a given view.	#TO_COMPOSIT ION_PARENT	Views that do not represent the root of the hierarchy must have an association to their compositional parent view annotated with #TO_COMPOSITION_PARENT.	#TO_COMPOSIT ION_ROOT	Additionally, views that do not represent the root must have a #TO_COMPOSITION_ROOT association for performance reasons (for example: authorization checks (instance restrictions) based of information from root views).
Value	Description								
#TO_COMPOSIT ION_CHILD	Within the compositional hierarchy there must be a #TO_COMPOSITION_CHILD association leading to each direct sub view of a given view.								
#TO_COMPOSIT ION_PARENT	Views that do not represent the root of the hierarchy must have an association to their compositional parent view annotated with #TO_COMPOSITION_PARENT.								
#TO_COMPOSIT ION_ROOT	Additionally, views that do not represent the root must have a #TO_COMPOSITION_ROOT association for performance reasons (for example: authorization checks (instance restrictions) based of information from root views).								
ObjectModel. compositionR oot	Defines the root of a compositional hierarchy								
	Scope: [VIEW]								

Annotation	Meaning								
ObjectModel. createEnable d	<p>If this annotation has value <code>true</code>, it is allowed to create new instances.</p> <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL: Exposes this information • BOPF: Rejects create requests that do not provide a value <code>true</code> for this field (in case of static field control). With the value '<code>EXTERNAL_CALCULATION</code>', the create property is calculated in a BOPF property determination (dynamic field control). 								
ObjectModel. dataCategory	<p>Defines the category of data that is represented by the below-mentioned values.</p> <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine): In ABQL joins between a <i>Data</i>- and a <i>Text</i> entity without explicit language key handling will be interpreted as a 1 : (0,1) association, where the language key is defaulted with the logon language.</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>#TEXT</td><td>Indicates that the annotated entity represents texts. Usually one key element is of type language.</td></tr> <tr> <td colspan="2">NOTE: Within the VDM a text view is always language-dependent.</td></tr> <tr> <td>#HIERARCHY</td><td>Indicates that the entity represents the hierarchy-related data. This could be a header information or structure information.</td></tr> </tbody> </table>	Value	Description	#TEXT	Indicates that the annotated entity represents texts. Usually one key element is of type language.	NOTE: Within the VDM a text view is always language-dependent.		#HIERARCHY	Indicates that the entity represents the hierarchy-related data. This could be a header information or structure information.
Value	Description								
#TEXT	Indicates that the annotated entity represents texts. Usually one key element is of type language.								
NOTE: Within the VDM a text view is always language-dependent.									
#HIERARCHY	Indicates that the entity represents the hierarchy-related data. This could be a header information or structure information.								
ObjectModel. deleteEnable d	<p>If this annotation has value <code>true</code>, it is allowed to delete existing instances.</p> <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL: Exposes this information • BOPF: Rejects delete requests that do not provide a value <code>true</code> for this field (in case of static field control). With the value '<code>EXTERNAL_CALCULATION</code>', the delete property is calculated in a BOPF property determination (dynamic field control). 								
ObjectModel. enabled	<p>If this annotation has the value <code>true</code>, the corresponding element (field or association) is supported at runtime.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine): SADL: Exposes this information</p>								

Annotation	Meaning
ObjectModel.entityChangeStateId	<p>This annotation is related to a single field that contains the change state of an active instance of an business object. The change state is always updated as soon as the instance data is changed. Usually, fields like last changed timestamp, hash values, or version counters are used as EntityChangeStateIds.</p> <p>With the value 'EXTERNAL_CALCULATION', the change state of a business object instance is calculated in BOPF. You can use this value if the underlying table does not provide any field that is applicable as an entity tag (ETag). BOPF calculates the hash value form the entire business object. This calculation can be used as an indicator for changes concerning the state of the related instances. Consider however that this calculation might have a negative effect on the performance.</p> <p>Scope: [VIEW]</p>
ObjectModel.foreignKey.association	<p>Defines association to a view that represents a value list/check table of the annotated filed. The annotated field must be valuated as equal to the annotated representative key field of the target view. The maximum target cardinality of the association has to be 1.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • Analytic Manager: Uses associated view as DIMENSION view for the annotated field. • SADL: Derives a default value help support from the foreign key relationship.
ObjectModel.hierarchy.association	<p>This annotation can be added to the key field that specifies the association to a hierarchy view. The hierarchy view defines an external hierarchy for the instances of the current view and is annotated with @ObjectModel.dataCategory: #HIERARCHY.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine):</p> <p>Analytic Manager: Uses associated view for hierarchical presentation.</p>
ObjectModel.lifecycle.processor.expiryBehavior	<p>The assignment of a processor to a shared draft related causes an exclusive lock for editing the draft document. The lifecycle.processor. annotations allow to overrule the global defaults for the processor expiration handling. After a certain period of time, the exclusive lock on the draft document has to be released. This period of time is defined by the specified expiry behavior.</p> <ul style="list-style-type: none"> • RELATIVE_TO_ENQUEUE_START: The period of time starts when the processor has been initially assigned. • RELATIVE_TO_LAST_CHANGE: The interval RELATIVE_TO_LAST_CHANGE describes the period of inactivity (no modifying roundtrips to the draft document), after that the processor has to be removed. <p>Scope: [VIEW]</p>
ObjectModel.lifecycle.processor.expiryInterval	<p>The duration interval can be specified by the annotation lifecycle.processor.expiryInterval in order to overrule the default duration. The value must be compliant to the dayTimeDuration format.</p> <p>Scope: [VIEW]</p>

Annotation	Meaning
ObjectModel.lifecycle.processor.notificationBeforeExpiryInterval	<p>To notify the draft processor in advance before the expiration takes place, a notification will be send. The default warning interval can be overruled by the help of annotation <code>notificationBeforeExpiryInterval</code>. The value must be compliant to the dayTimeDuration format.</p> <p>Scope: [VIEW]</p>
ObjectModel.lifecycle.enqueue.expiryBehavior	<p>The creation of an exclusive draft related to an existing active document causes an exclusive durable lock of the active document. The <code>lifecycle.enqueue.*</code> annotations allow to overrule the global defaults for the durable lock expiration handling. After a certain period of time, the durable exclusive lock on the active document has to be released. This period of time is defined by the expiration behavior.</p> <p>Enqueue Expiration Behavior:</p> <ul style="list-style-type: none"> • <code>RELATIVE_TO_ENQUEUE_START</code> - The period of time starts from the point in time the exclusive durable lock has been initially acquired. • <code>RELATIVE_TO_LAST_CHANGE</code> - The interval describes the period of inactivity (no modifying roundtrips to the draft document), after that the exclusive durable lock has to be removed <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine): The lifecycle services automatically unlock active documents accordingly to the defined enqueue expiration behavior.</p>
ObjectModel.lifecycle.enqueue.expiryInterval	<p>Using this annotation, you can specify an interval to overrule the default duration. The value must be compliant to the dayTimeDuration format (http://www.w3.org/TR/xmlschema11-2/#dayTimeDuration).</p> <p>Scope: [VIEW]</p>
ObjectModel.lifecycle.enqueue.notificationBeforeExpiryInterval	<p>To notify the draft processor in advance before the expiration takes place, a notification will be send. The default warning interval can be overruled by the help of annotation <code>notificationBeforeExpiryInterval</code> using the valid dayTimeDuration format (http://www.w3.org/TR/xmlschema11-2/#dayTimeDuration).</p> <p>Scope: [VIEW]</p>

Annotation	Meaning
ObjectModel.lifecycle.processing.expiryBehavior	<p>After the durable lock expiration phase has been processed for an exclusive draft, the durable lock of its active document is released. However for draft-aware applications it is still not allowed to create a new draft related to the same active document until the processing of the existing draft is expired.</p> <p>The <code>lifecycle.processing.*</code> annotations allow to overrule the global defaults for the draft processing expiration handling. After a certain period of time, the processor is removed from an exclusive draft and a different user is allowed to create a new draft related to the same active document. In that case, the first draft is deleted. Otherwise, the editing for the first draft can be continued at a later point in time.</p> <p>The default processing expiration settings can be overruled like the <code>lifecycle.enqueue.*</code> annotations by the help of a behavior, duration interval and notification interval.</p> <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine): The lifecycle services automatically set the processing status of a draft accordingly to the defined processing expiration behavior.</p>
ObjectModel.lifecycle.processing.expiryInterval	<p>Using this annotation, you can specify an interval to overrule the default duration interval.</p> <p>Scope: [VIEW]</p>
ObjectModel.lifecycle.processing.notificationBeforeExpiryInterval	<p>Using this annotation, you can specify an interval to overrule the default notification interval.</p> <p>Scope: [VIEW]</p>
ObjectModel.lifecycle.draft.expiryBehavior	<p>To comply to data privacy policies and the deletion of personal data, drafts have to be deleted after a certain period if they are not processed any more. The default expiration timeframe is 28 days. The <code>lifecycle.draft.</code> annotations allow to overrule the global defaults for the draft expiration handling. After a certain period of time, the draft document has to be deleted. This period of time is defined by the expiry behavior.</p> <ul style="list-style-type: none"> • <code>RELATIVE_TO_ENQUEUE_START</code> : The period of time starts from the point in time the draft document has been initially created. • <code>RELATIVE_TO_LAST_CHANGE</code> : The interval describes the period of inactivity (no modifying roundtrips to the draft document), after that the draft document has to be removed. <p>Scope: [VIEW]</p>
ObjectModel.lifecycle.draft.expiryInterval	<p>Using this annotation, you can overrule the default duration. The value must be compliant to the dayTimeDuration format.</p> <p>Scope: [VIEW]</p>
ObjectModel.lifecycle.draft.notificationBeforeExpiryInterval	<p>To notify the draft processor in advance before the expiration takes place, a notification will be send. The default warning interval can be overruled using this annotation. The value must be compliant to the dayTimeDuration format.</p> <p>Scope: [VIEW]</p>

Annotation	Meaning				
ObjectModel.mandatory	<p>Defines that element is mandatory</p> <p>If this annotation has the value <code>true</code>, the field must be filled by the consumer when executing a modification.</p> <p>With the value '<code>EXTERNAL_CALCULATION</code>', you have the option to specify on which conditions the element is defined as mandatory (dynamic field control). The mandatory property is in this case calculated in a property determination of the corresponding business object's node.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL: Exposes this information • BOPF: The evaluation runtime rejects modifications that do not provide a value for the annotated field. However, BOPF does not automatically check for the mandatory property. For this purpose, it is required that you add a validation to the corresponding business object's node and configure it as a <i>consistency</i> validation for checking mandatory properties (usually with the trigger condition for <i>Create</i> and <i>Update</i>). For implementation of the consistency validation, you can use the library class <code>/BOBF/CL_LIB_V_MANDATORY_ATTR</code>, or your own implementation in case this class should not fit to your needs. 				
ObjectModel.modelCategory	<p>Each business object can semantically be categorized using this object model setting</p> <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine): The model category must not have any runtime effect but is used for a semantic grouping, for example in view browsers.</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>#BUSINESS_OBJECT</td><td>View that represents a BOPF business object</td></tr> </tbody> </table>	Value	Description	#BUSINESS_OBJECT	View that represents a BOPF business object
Value	Description				
#BUSINESS_OBJECT	View that represents a BOPF business object				
ObjectModel.readOnly	<p>If this annotation has the value <code>true</code>, the field must not be updated by the consumer.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL: Exposes this information • BOPF: Rejects modifications when updating fields that are defined as <code>@ObjectModel.readOnly: true</code> (in case of static field control). With the value '<code>EXTERNAL_CALCULATION</code>', the readonly property is calculated in a BOPF property determination (dynamic field control). 				

Annotation	Meaning
ObjectModel.representativeKey	<p>Most specific element (field or managed association) of the primary key (indicated by the keyword KEY) that represents the entity which the view is based on. This element shall be used as the anchor for defining foreign key relationships (except for text views): The foreign key field corresponding to the representative key represents the entity. As such it can be called representative foreign key element. The foreign key association is defined on the representative foreign key element. The name of the representative key typically equals the name of the entity represented by the view.</p> <p>For non-text views it is the key element for which the view serves as a value list/check table. For text views (@ObjectModel.dataCategory: #TEXT) it identifies the key element to which the text fields relate to.</p> <p>The representative key element has to be modelled explicitly even if there is only one primary key field (no implicit derivation).</p> <p>A view may only become a target of a foreign key association if it has a representative key element (exception: language dependent text views may not be used as targets of foreign key relationships)</p> <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine): Analytic Manager: In analytics, both the grouping by the entity and the hierarchical representation of the entity are handled using the representative foreign key field.</p>
ObjectModel.semanticKey[]	<p>Identifies an instance of an entity from business perspective using human-readable field values. It does neither contain time-/language-dependent nor other technical components (for example: draft indicator). Thus it may be ambiguous resulting in multiple selected records/instances of a view that may need to be filtered using contextual information (for example: current date, preferred language).</p> <p>For a given entity, only a single semantic key is defined.</p> <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine): UI: Uses semantic key for bookmarking and navigation.</p>
ObjectModel.text.association	<p>Defines the associated view (annotated with @ObjectModel.dataCategory: #TEXT), which provides textual descriptions for the annotated field.</p> <p>NOTE: The usage of this annotation excludes the usage of @ObjectModel.text.element.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL - Enriches the OData entity type of the view with the textual description of the target view applying an automated language filtering. The name of the auto-generated text property will be composed out of the annotated field name and the constant suffix _Text. This OData property is mapped onto the first text field of the associated target CDS view annotated with @Semantics.text:true. • Analytic Manager - Uses the associated view as TEXT view for annotated field.
ObjectModel.text.element[]	<p>Establishes the conjunction of a field with its descriptive language-independent texts.</p> <p>NOTE: The usage of this annotation excludes the usage of @ObjectModel.text.association.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine): SADL - First text field listed in the annotation array will be handled as descriptive text of the annotated field in OData exposure scenarios.</p>

Annotation	Meaning
ObjectModel.transactionalProcessingEnabled	<p>Indicates that transactional accesses to the view are delegated to the transactional runtime of the underlying view (which is annotated with <code>@ObjectModel.transactionalProcessingEnabled:true</code>). It may only be defined on root view level (<code>@ObjectModel.compositionRoot:true</code>).</p> <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL: Uses the information to delegate requests to the transactional runtime
ObjectModel.updateEnabled	<p>If this annotation has value <code>true</code>, it is allowed to update existing instances.</p> <p>Scope: [VIEW]</p> <p>Evaluation Runtime (Engine):</p> <ul style="list-style-type: none"> • SADL: Exposes this information • BOPF: Rejects modification requests that do not provide a value <code>true</code> for this field (in case of static field control). With the value '<code>EXTERNAL_CALCULATION</code>', the update property is calculated in a BOPF property determination (dynamic field control).
ObjectModel.usageType.sizeCategory	<p>The size category enables the consumer of a service to estimate the possible results set. It reflects the set of data that has to be searched through to compute, for example, the number of rows in the results set by using a <code>count (*)</code> function in a CDS view.</p> <p>You can assign one of the following size categories, which specifies the expected number of data sets (rows) in a production customer system:</p> <p>S: < 1000</p> <p>M: < 100.000</p> <p>L: < 10.000.000</p> <p>XL: < 100.000.000</p> <p>XXL: >= 100.000.000</p> <p>Scope: [VIEW]</p>

Annotation	Meaning
ObjectModel.usageType.serviceQuality	<p>This annotation reflects the quality of the service that results from the CDS view. Using this annotation, the consumer is able to decide whether or not the annotated CDS view fits the demanded response time and the throughput requirements.</p> <p>Each CDS view can be assigned to one of the following quality categories:</p> <ul style="list-style-type: none"> A: The annotated CDS view may be consumed within the business logic for high volume transactions or in background processing. B: The annotated CDS view may be consumed within business logic for transactions or in background processing. C: The annotated CDS view may be consumed from the UI in transactions for single object retrieval. D: The annotated CDS view may be consumed in analytical reporting. X: The annotated CDS view is used to push down the application's code to SAP HANA DB. <p>Scope: [VIEW]</p>
ObjectModel.usageType.dataClass	<p>To support the decision on cache strategies for higher layers and to enable client-side statement routing using these caches, each CDS view can be assigned to a data class (@ObjectModel.usageType.dataClass).</p> <p>The different data classes correspond with different life time cycles:</p> <ul style="list-style-type: none"> TRANSACTIONAL data: The CDS view provides data that is written or changed in high volume transactions or in background processing. Examples: Header or items for sales order processing or financial bookings. MASTER data is read or changed but not written in high volume transactions or in background processing. This data typically drives the business process decisions when business logic is executed. It is also displayed to the end user as context information or to enable user decisions when the transactions are executed manually. Examples are material data, business partner data, or account data. ORGANIZATIONAL data describes the organizational structure of a company and its business processes. You can consider it as a special kind of MASTER data. Examples are sales unit, distribution channel and cost center. CUSTOMIZING data describes how a concrete business process is executed in a customer system landscape. This data typically consists of a content that is delivered by SAP and added to by the customer. Examples are countries, units of measures, and currencies. META data classes either specify how the system is configured or describes the technical structure of entities. Typically, this type of data is part of content that is shipped to customers. Examples are DDIC structures, field controls, and authorization objects. MIXED data classes should be used if the annotated CDS view contains tables with data that represents multiple data classes. <p>Scope: [VIEW]</p>
ObjectModel.writeActivePersistence	<p>Defines the DDIC database table name of the active view. This requires @ObjectModel.transactionalProcessingEnabled: true</p> <p>Scope: [VIEW]</p>

Annotation	Meaning
ObjectModel. writeDraftPe rsistence	Defines the DDIC database table name of the draft data. This requires @ObjectModel.transactionalProcessingEnabled: true Scope: [VIEW]
ObjectModel. writeEnabled-	Enables the transactional runtime support. It may only be defined on root view level (@ObjectModel.compositionRoot: true).
Deprecated!	<p>⚠ Caution</p> <p>Do not use this annotation anymore! It is replaced by ObjectModel.transactionalProcessingEnabled.</p>
	Scope: [VIEW]
	Evaluation Runtime (Engine):
	<p>BOPF - Depending on the specified persistence information, we can distinguish between the following scenarios:</p> <ul style="list-style-type: none"> a) <code>writeActivePersistence</code>: Active data (defined by that CDS view) is directly modified by the transactional runtime (without having any kind of draft). b) <code>writeDraftPersistence</code>: Active data (defined by that CDS view) is only indirectly modified by a draft. c) <code>writeActivePersistence & writeDraftPersistence</code>: Both the draft and the active data are written by the transactional runtime.

Examples

Example 1

This example demonstrates how you can define a compositional hierarchy.

↳ Sample Code

```
@ObjectModel.modelCategory: #BUSINESS_OBJECT
@ObjectModel.compositionRoot: true
define view I_SalesOrder
    association [0..*] to I_SalesOrderItem as _Item ...
        @ObjectModel.association.type: #TO_COMPOSITION_CHILD
        _Item, ...
    }
define view I_SalesOrderItem
    association [1..1] to I_SalesOrder as _SalesOrder ...
        @ObjectModel.association.type: [#TO_COMPOSITION_ROOT,
                                         #TO_COMPOSITION_PARENT]
        _SalesOrder, ...
    }
```

Example 2

This example demonstrates how you can define foreign key relationships.

↳ Sample Code

```
define view I_SalesOrderItem
  association [0..1] to I_Material as _Material
    on $projection.Material = _Material.Material ... {
      @ObjectModel.foreignKey.association: '_Material'
      Material,
      _Material, ...
    }
  @ObjectModel.representativeKey: 'Material'
define view I_Material ... {
  key Material, ...
}
```

Example 3

This example demonstrates how you can define language-dependent texts

↳ Sample Code

```
define view I_Material
  association [0..*] to I_MaterialText as _Text ... {
    @ObjectModel.text.association: '_Text'
    key Material,
    _Text, ...
  }
  @ObjectModel.dataCategory: #TEXT
  @ObjectModel.representativeKey: 'Material'
define view I_MaterialText ... {
  key Material,
  @Semantics.language: true
  key Language,
  @Semantics.text: true
  MaterialName,
  @Semantics.text: true
  MaterialDescription, ...
}
```

Example 4

This example demonstrates how you can define language-independent texts

↳ Sample Code

```
define view I_Plant ... {
  @ObjectModel.text.element: ['PlantName']
  key Plant,
  @Semantics.text: true
  PlantName, ...
}
```

Example 5

This example demonstrates how you can define the transactional behavior

↳ Sample Code

```
@ObjectModel.compositionRoot: true  
@ObjectModel.transactionalProcessingEnabled: true  
@ObjectModel.writeDraftPersistence: '<DraftDDICTable>',  
define view I_MaterialWithDraft ... {  
    ...  
}
```

9.1.9 OData Annotations

Capture OData-related aspects to expose data gained from a CDS entity in an OData service.

Scope and Definition

```
@Scope:[#VIEW, #TABLE_FUNCTION]  
Annotation OData  
{  
    publish : Boolean default true;  
};
```

Usage

Annotation	Meaning						
OData.publish	OData.publish is intended for generating an OData service. Scope: [VIEW, TABLE_FUNCTION] Evaluation Runtime (Engine): SSDL - Generates the OData service Values: <table><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td><i>true</i></td><td>Has to be set to generate an OData service</td></tr><tr><td><i>false</i></td><td>Defines that this CDS entity cannot be exposed for an OData service</td></tr></tbody></table>	Value	Description	<i>true</i>	Has to be set to generate an OData service	<i>false</i>	Defines that this CDS entity cannot be exposed for an OData service
Value	Description						
<i>true</i>	Has to be set to generate an OData service						
<i>false</i>	Defines that this CDS entity cannot be exposed for an OData service						

i Note

When the CDS entity is activated, the OData service is generated automatically. Before you can open and use the OData service, it also needs to be activated using transaction /IWFND/MAINT_SERVICE manually. For further information, see the Related Information below.

You can then start the generated OData service through your Eclipse-based IDE. To do this, choose **F2** from the name of the CDS entity that is used in the `define view` statement. The *ABAP Element Information* popup is then opened. In the *Generated Objects* section, the *OData Service* link is provided. After clicking the link, the default Internet browser is opened and displays its XML code.

Examples

This example demonstrates how you can define the `SEPM_I_SalesOrder` CDS entity that exposes data to an OData service.

Here, an OData service is generated for the CDS entity. The data is provided from the database table `snwd_so`.

↳ Sample Code

```
@OData.publish:true
define view SEPM_I_SalesOrder as select from snwd_so {
    ...
}
```

Related Information

[Generating OData Service With Auto-Exposure \[page 19\]](#)

9.1.10 Search Annotations

This annotation marks a view as searchable. You define the fuzziness threshold as well as the specifics of term mappings at element level.

Scope and Definition

```
@Scope:[#VIEW, #TABLE_FUNCTION]
Annotation Search
{
    searchable : Boolean default true;
};

@Scope:[#ELEMENT]
Annotation Search
{
    defaultSearchElement : Boolean default true;
    ranking : String(6) enum { HIGH = 'high'; MEDIUM = 'medium'; LOW = 'low'; }
    default #MEDIUM;
    fuzzinessThreshold : Decimal(3,2);
    termMappingDictionary : String(128);
```

```

    termMappingListID : String(32);
};

```

Usage

Annotation	Meaning				
Search.searchable	<p>Defines if a CDS view or entity is generally relevant for search scenarios. This annotation must be set in case other search-related annotations are being defined for elements of the respective CDS view or entity. The annotation offers a general switch and a means to quickly detect whether a view is relevant or not.</p> <p>Scope: #View</p> <p>Evaluation Runtime (Engine): Interpreted by Enterprise Search and SADL</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>Boolean (true, false)</i></td><td>Defines whether a view is relevant for search or not. Default: true</td></tr> </tbody> </table>	Value	Description	<i>Boolean (true, false)</i>	Defines whether a view is relevant for search or not. Default: true
Value	Description				
<i>Boolean (true, false)</i>	Defines whether a view is relevant for search or not. Default: true				
Search.defaultSearchElement	<p>Specifies that the element is to be considered in a freestyle search (for example a SELECT...) where no columns are specified.</p> <p>Usually, such a search must not operate on all elements – for performance reasons, and because not all elements (e.g. internal keys) do qualify for this kind of access.</p> <p>Scope: #Element</p> <p>Evaluation Runtime (Engine): Interpreted by Enterprise Search and SADL</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>Boolean (true, false)</i></td><td>Defines weather the element is to be considered in a free-style search. Default: true</td></tr> </tbody> </table>	Value	Description	<i>Boolean (true, false)</i>	Defines weather the element is to be considered in a free-style search. Default: true
Value	Description				
<i>Boolean (true, false)</i>	Defines weather the element is to be considered in a free-style search. Default: true				
Search.ranking	<p>Specifies how relevant the values of an element are for ranking, if the freestyle search terms match the element value.</p> <p>Scope: #Element</p> <p>Evaluation Runtime (Engine) : Interpreted by Enterprise Search</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>HIGH</td><td>The element is of high relevancy; this holds usually for ID and their descriptions.</td></tr> </tbody> </table>	Value	Description	HIGH	The element is of high relevancy; this holds usually for ID and their descriptions.
Value	Description				
HIGH	The element is of high relevancy; this holds usually for ID and their descriptions.				

Annotation	Meaning
	MEDIUM The element is of medium relevancy; this holds usually for other, important element. This is the default.
	LOW Although the element is relevant for freestyle search, a hit in this element has no real significance for a result item's ranking.
Search.fuzzinessThreshold	Specifies the least level of fuzziness (with regard to some comparison criteria passed at runtime) the element has to have to be considered in a fuzzy search at all.

i Note

A fuzzy search enables a certain degree of error tolerance and returns records even if the search term contains additional or missing characters or other types of spelling errors.

i Note

To perform a fuzzy search you have to set the *search mode* to *fuzzy* in the customizing settings of your ABAP system. Find the customizing node under ► *SAP NetWeaver Implementation Guide* ► *Search and Operational Analytics* ► *Enterprise Search* ► *Search Configuration* ► *Set Parameters for Federated Search* ►.

If in the customizing a value for *Fuzzy Similarity* is present, the value of the parameter *Search.fuzzinessThreshold* will become void.

Scope: #Element

Evaluation Runtime (Engine): Interpreted by SADL

Values:

Value	Description
Decimal (3,2)	The least level of fuzziness the element has to have to be considered in a fuzzy search at all, e.g. 0 . 7. The value can be between 0 and 1. We recommend using the default value 0 . 7 to start with. Later on, you can fine-tune the search settings based on your experiences with the search. You can also fine-tune the search using feedback collected from your users. A value between 0 . 7 and 0 . 99 would be most useful. Use 1 for exact matches.

Annotation	Meaning				
Search.termMappingDictionary	<p>Specifies the table that holds the term mappings (synonyms) to be considered in the context of a search on this view.</p> <p>Scope: #Element</p> <p>Evaluation Runtime (Engine): No engine usage right now. Reserved for future usage.</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>String(128)</i></td><td>Defines the term mapping dictionary, e.g. a table or entity.</td></tr> </tbody> </table>	Value	Description	<i>String(128)</i>	Defines the term mapping dictionary, e.g. a table or entity.
Value	Description				
<i>String(128)</i>	Defines the term mapping dictionary, e.g. a table or entity.				
Search.termMappingListID	<p>Specifies one or multiple list IDs within the term mapping dictionary mentioned before.</p> <p>The list is implemented as a column of the term mapping table, with the list ID as content of this column. This concept has the aim to enable overarching term mapping dictionaries while being able to separate domain-specific content at the same time.</p> <p>Scope: #Element</p> <p>Evaluation Runtime (Engine): No engine usage right now. Reserved for future usage.</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>Array of String(32)</i></td><td>Defines one or more columns of the term mapping dictionary.</td></tr> </tbody> </table>	Value	Description	<i>Array of String(32)</i>	Defines one or more columns of the term mapping dictionary.
Value	Description				
<i>Array of String(32)</i>	Defines one or more columns of the term mapping dictionary.				

9.1.11 Semantics Annotations

Used by the core engines for data processing, analytics, and data consumption

Scope and Definition

```
@Scope:[#ELEMENT, #PARAMETER]
Annotation Semantics
{
  telephone
  {
    type : array of String(10) enum { HOME; CELL; WORK; FAX; PREF; TEXT;
    VOICE; VIDEO; PAGER; TEXT_PHONE; } default #PREF;
  };
  eMail
  {
    type : array of String(10) enum { HOME; WORK; PREF; OTHER; } default #PREF;
    address : Boolean default true;
    from : Boolean default true;
    sender : Boolean default true;
    to : Boolean default true;
    cc : Boolean default true;
  }
}
```

```

bcc : Boolean default true;
subject : Boolean default true;
body : Boolean default true;
keywords : Boolean default true;
received : Boolean default true;
};

name
{
    fullName : Boolean default true;
    givenName : Boolean default true;
    additionalName : Boolean default true;
    familyName : Boolean default true;
    nickName : Boolean default true;
    suffix : Boolean default true;
    prefix : Boolean default true;
    jobTitle : Boolean default true;
};

address
{
    type : array of String(10) enum { HOME; WORK; PREF; OTHER; } default #PREF;
    city : Boolean default true;
    street : Boolean default true;
    streetNoNumber : Boolean default true;
    number : Boolean default true;
    country : Boolean default true;
    region : Boolean default true;
    subRegion : Boolean default true;
    zipCode : Boolean default true;
    postBox : Boolean default true;
    label : Boolean default true;
};

organization
{
    name : Boolean default true;
    unit : Boolean default true;
    role : Boolean default true;
};

calendarItem
{
    summary : Boolean default true;
    description : Boolean default true;
    categories : Boolean default true;
    dtStart : Boolean default true;
    dtEnd : Boolean default true;
    duration : Boolean default true;
    due : Boolean default true;
    completed : Boolean default true;
    priority : Boolean default true;
    class : Boolean default true;
    status : Boolean default true;
    percentComplete : Boolean default true;
    contact : Boolean default true;
    location : Boolean default true;
    transparent : Boolean default true;
    fbType : Boolean default true;
    wholeDay : Boolean default true;
};

businessDate
{
    at : Boolean default true;
    from : Boolean default true;
    to : Boolean default true;
    createdAt : Boolean default true;
    lastChangedAt : Boolean default true;
};

systemDate
{
    createdAt : Boolean default true;
}

```

```

        lastChangedAt : Boolean default true;
    };
    time : Boolean default true;
    durationInSeconds : Boolean default true;
    calendar
    {
        dayOfMonth : Boolean default true;
        dayOfYear : Boolean default true;
        week : Boolean default true;
        month : Boolean default true;
        quarter : Boolean default true;
        halfyear : Boolean default true;
        year : Boolean default true;
        yearWeek : Boolean default true;
        yearMonth : Boolean default true;
        yearQuarter : Boolean default true;
        yearHalfyear : Boolean default true;
    };
    fiscal
    {
        yearVariant : Boolean default true;
        period : Boolean default true;
        year : Boolean default true;
        yearPeriod : Boolean default true;
    };
    geoLocation
    {
        longitude : Boolean default true;
        latitude : Boolean default true;
        cartoId : Boolean default true;
        normalizedName : Boolean default true;
    };
    url
    {
        mimeType : elementRef;
    };
    imageUrl : Boolean default true;
    contact
    {
        type : String(10) enum {PERSON; ORGANIZATION; };
        note : Boolean default true;
        photo : Boolean default true;
        birthDate : Boolean default true;
    };
    user
    {
        id : Boolean default true;
        createdBy : Boolean default true;
        lastChangedBy : Boolean default true;
        responsible : Boolean default true;
    };
    mimeType : Boolean default true;
    text : Boolean default true;
    language : Boolean default true;
    languageReference : elementRef;
};
@Scope:[#ELEMENT]
Annotation Semantics
{
    amount
    {
        currencyCode : elementRef;
    };
    quantity
    {
        unitOfMeasure : elementRef;
    };
    currencyCode : Boolean default true;
}

```

```

    unitOfMeasure : Boolean default true;
};

```

Usage

Semantic annotations are used to inform the client as to which of the elements contain a phone number, a part of a name or address, or something relating to a calendar event. They must not be bound, for example, to a dedicated consumption channel. They need to be available for consumption through OData, (S)QL, and so on.

Semantic annotations complement the concept of semantic data types, while semantic data types always introduce specific behavior in the provider/core infrastructure (through dedicated operations or conversion functions).

Semantic annotations allow the standardizing of semantics that only have an impact on the consumption side (such as telephone number, mail address, city, and so on).

Annotation	Meaning												
	Annotations belonging to <i>Semantics.address</i> follow the vCard standard (RFC6350)												
Scope: [ELEMENT, PARAMETER]													
Evaluation Runtime (Engine): SADL:	Translates CDS annotations into the corresponding OData annotations (Exception: <i>Semantics.address.number</i> and <i>Semantics.address.streetNoNumber</i>)												
Values:													
<i>Semantics.address.type[]</i>	<p>Description</p> <p>Values:</p> <p>String (10)</p> <p>The following enumerations are provided:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>HOME</i></td><td>Home address</td></tr> <tr> <td><i>WORK</i></td><td>Work address</td></tr> <tr> <td><i>PREF</i></td><td>Preferred address</td></tr> <tr> <td></td><td>Default</td></tr> <tr> <td><i>OTHER</i></td><td>Other address</td></tr> </tbody> </table>	Value	Description	<i>HOME</i>	Home address	<i>WORK</i>	Work address	<i>PREF</i>	Preferred address		Default	<i>OTHER</i>	Other address
Value	Description												
<i>HOME</i>	Home address												
<i>WORK</i>	Work address												
<i>PREF</i>	Preferred address												
	Default												
<i>OTHER</i>	Other address												
<i>Semantics.address.city</i>	Boolean default true												
<i>Semantics.address.country</i>	The annotated field contains a plain-text string that contains the name of a city.												
	The annotated field contains a plain-text string that contains the name of a country.												

Annotation	Meaning
<code>Semantics.address.label</code>	The annotated field contains a plain-text string representing the formatted address for printing.
<code>Semantics.address.number</code>	The annotated field contains a street number separated from a street name.
<code>Semantics.address.postBox</code>	The annotated field contains information about a post office box.
<code>Semantics.address.region</code>	The annotated field contains a plain-text string that contains the name of a region.
<code>Semantics.address.subRegion</code>	The annotated field contains a plain-text string that contains the name of a subregion.
<code>Semantics.address.street</code>	The annotated field contains a street name and a street number.
<code>Semantics.address.streetNoNumber</code>	The annotated field contains a street name separated from a street number.
<code>Semantics.address.zipCode</code>	The annotated field contains a numeric string that contains the ZIP code (type of postal code used within the United States).

Annotations belonging to `Semantics.amount` contain a monetary amount, and the corresponding currency code is contained in the referenced field.

Scope: [ELEMENT]

Evaluation Runtime (Engine): Interpreted by **ABAP** Runtime Environment

Values:

<code>Semantics.amount.currencyCode</code>	elementRef	The annotated field contains a monetary amount, and the corresponding currency code is contained in the referenced field.
--	------------	---

Annotation	Meaning
Annotations belonging to <code>Semantics.businessDate</code> contain information about changes of database table records.	
Scope: [ELEMENT, PARAMETER]	
Evaluation Runtime (Engine): *Business data often carries time validities. When accessing data, one usually intends to get the most current or a specific state according to a defined date. It is mandatory for the engine to know the semantics in order to carry out a generic validity evaluation.	
<p>i Note</p> <p>Many objects used in the Business Suite store data in a way that requires an algorithm to derive the actual data for a validity state. The annotation must not be used if a generic evaluation would yield incorrect results.</p>	
Values:	
<code>Semantics.businessDate.at*</code>	Boolean default true
<code>Semantics.businessDate.createdAt*</code>	The annotated field indicates that the column contains the key dates and not intervals.
<code>Semantics.businessDate.from*</code>	The annotated field is the date (and time) when the database table record was created for the first time.
<code>Semantics.businessDate.to*</code>	The annotated field is the date timestamp or interval that defines the validity of the data of the database table record from a business point of view.
<code>Semantics.businessDate.lastChangedAt</code>	Technical validity: The <code>from</code> date is different from the point in time when the record was created. The dates are not to be confused with dates contained in the data part.
<code>Semantics.businessDate.lastChangedAt</code>	The annotated field is the date/time when the database table record was modified for the last time.
<code>Semantics.businessDate.lastChangedAt</code>	It is usually identical to the creation date/time of the insertion, if the record has not been modified.
<p>Evaluation Runtime (Engine):</p> <p>This is required to answer queries like "all changes since..." or "most recent ...". The latter is very relevant, for example, for the ranking of a search result.</p>	

Annotation	Meaning
	<p>Annotations belonging to <i>Semantics.calendar</i> follow the iCalendar standard (RFC5545)</p> <p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine): SADL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: Boolean default true</p>
<i>Semantics.calendar.dayOfMonth</i>	<p>Boolean default true</p> <p>The value of the annotated field is a day number relative to a calendar month.</p>
	<p>Example</p> <p>1 - 31</p>
<i>Semantics.calendar.dayOfYear</i>	<p>The value of the annotated field is a day number relative to a calendar year.</p>
	<p>Example</p> <p>1 - 366</p>
<i>Semantics.calendar.halfyear</i>	
<i>Semantics.calendar.month</i>	<p>The value of the annotated field encodes a calendar month number as a string following the logical pattern MM consisting of two digits.</p>
	<p>Example</p> <p>The string matches the regex pattern 0 [1-9] 1 [0-2]</p>
<i>Semantics.calendar.quarter</i>	<p>The value of the annotated field encodes a calendar quarter number as a string following the logical pattern Q consisting of a single digit.</p>
	<p>Example</p> <p>The string matches the regex pattern [1-4]</p>

Annotation	Meaning
<code>Semantics.calendar.week</code>	The value of the annotated field encodes a calendar week number as a string following the logical pattern WW consisting of two digits.
	<p>Example</p> <p>The string matches the regex pattern <code>0 [1-9] [1-4] [0-9] 5 [2-3]</code></p>
<code>Semantics.calendar.year</code>	The value of the annotated field encodes a year number as a string following the logical pattern <code>(-?) YYYY (Y*)</code> consisting of an optional minus sign for years B.C., followed by at least four digits.
	<p>Example</p> <p>The string matches the regex pattern <code>-? ([1-9] [0-9] {3,} 0 [0-9] {3})</code></p>
<code>Semantics.calendar.yearHalfyear</code>	
<code>Semantics.calendar.yearMonth</code>	The value of the annotated field encodes a calendar year and month as a string following the logical pattern <code>(-?) YYYY (Y*) MM</code> consisting of an optional minus sign for years B.C., followed by at least six digits, where the last two digits represent the months January to December.
	<p>Example</p> <p>The string matches the regex pattern <code>-? ([1-9] [0-9] {3,} 0 [0-9] {3}) (0[1-9] 1[0-2])</code></p>

Annotation	Meaning
<code>Semantics.calendar.yearQuarter</code>	<p>The value of the annotated field encodes a calendar year and quarter as a string following the logical pattern $(-?) \text{YYYY} (\text{Y}^*) \text{Q}$ consisting of an optional minus sign for years B.C., followed by at least five digits, where the last digit represents the quarter.</p> <p>Example The string matches the regex pattern $-? ([1-9] [0-9] \{3\}) \{0 [0-9] \{3\}) [1-4]$</p>
<code>Semantics.calendar.yearWeek</code>	<p>The value of the annotated field encodes a calendar year and week as a string following the logical pattern $(-?) \text{YYYY} (\text{Y}^*) \text{WW}$ consisting of an optional minus sign for years B.C., followed by at least six digits, where the last two digits represent week number in the year.</p> <p>Example The string matches the regex pattern $-? ([1-9] [0-9] \{3\}) \{0 [0-9] \{3\}) (0[1-9] [1-4] [0-9] 5[2-3])$</p>

Annotations belonging to `Semantics.calendarItem` follow the iCalendar standard ([RFC5545](#)) for representing and exchanging calendaring and scheduling information such as events, to-dos, journal entries, and free or busy information, independent of any particular calendar service or protocol

Scope: [ELEMENT, PARAMETER]

Values:

<code>Semantics.calendarItem.categorie</code>	Boolean default true	The value of the annotated field is used to specify categories or subtypes of the calendar item. The categories are useful in searching for a calendar item of a particular type and category.
---	----------------------	--

Annotation	Meaning
<code>Semantics.calendarItem.class</code>	<p>The value of the annotated field provides a method of capturing the scope of the access the calendar owner intends for information within an individual calendar entry.</p> <p>The default value is PUBLIC. Other values are PRIVATE, CONFIDENTIAL, iana-token (iCalendar class registered with IANA), or x-name (experimental, non-standard parameter).</p>
	<p>i Note</p> <p>Applications must treat x-name and iana-token values they do not recognize the same way as they would the PRIVATE value.</p>
<code>Semantics.calendarItem.completed</code>	The value of the annotated field defines the date and time that a to-do was actually completed.
<code>Semantics.calendarItem.contact</code>	The value of the annotated field is used to represent contact information or alternately a reference to contact information associated with the calendar item.
<code>Semantics.calendarItem.description</code>	The value of the annotated field provides a description of the calendar item.
<code>Semantics.calendarItem.due</code>	The value of the annotated field defines the date and time that a to-do is expected to be completed.
<code>Semantics.calendarItem.duration</code>	The value of the annotated field is used to identify properties that contain a duration of time.
<code>Semantics.calendarItem.dtEnd</code>	The value of the annotated field specifies the date and time that a calendar item ends.
<code>Semantics.calendarItem.dtStart</code>	The value of the annotated field specifies the date and time that a calendar item starts.

Annotation	Meaning
<code>Semantics.calendarItem.fbType</code>	The value of the annotated field specifies the free or busy time type.
<code>Semantics.calendarItem.location</code>	The value of the annotated field defines a location related to a calendar component.
<code>Semantics.calendarItem.percentComplete</code>	<p>Example</p> <pre>LOCATION:Conference Room - F123\, Bldg. 002</pre> <pre>LOCATION;ALTREP="http://xyzcorp.com/ conf-rooms/ f123.vcf":</pre> <pre>Conference Room -</pre> <pre>F123\, Bldg. 002</pre>
<code>Semantics.calendarItem.priority</code>	The value of the annotated field is used by an assignee or delegatee of a to-do to convey the percent completion of a to-do to the "organizer".
<code>Semantics.calendarItem.status</code>	The value of the annotated field defines the relative priority for a calendar item.
<code>Semantics.calendarItem.summary</code>	The value of the annotated field defines the overall status or confirmation for the calendar item.
<code>Semantics.calendarItem.transparent</code>	The value of the annotated field defines a short summary or subject for the calendar item.
<code>Semantics.calendarItem.wholeDay</code>	The value of the annotated field defines whether or not an event is transparent to busy time searches.
The value of the annotated field defines whether or not an event covers whole days instead of exact time slots.	

Annotations belonging to `Semantics.contact` follow the vCard standard ([RFC6350](#))

Scope: [ELEMENT, PARAMETER]

Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations

Values:

Annotation	Meaning						
Semantics.contact.type	<p>Value: String (10)</p> <p>The following enumerations are provided:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>PERSON</td><td>Content relates to an individual</td></tr> <tr> <td>ORGANIZATION</td><td>Content relates to an organization, company, etc.</td></tr> </tbody> </table>	Value	Description	PERSON	Content relates to an individual	ORGANIZATION	Content relates to an organization, company, etc.
Value	Description						
PERSON	Content relates to an individual						
ORGANIZATION	Content relates to an organization, company, etc.						
Semantics.contact.birthDate	<p>Boolean default true</p> <p>This annotated field contains the birth date of the individual.</p>						
Semantics.contact.note	<p>This annotated field specifies supplemental information or a comment that is associated with the vCard.</p>						
Semantics.contact.photo	<p>This annotated field contains an image or photograph related to the contact.</p>						
Semantics.currencyCode	<p>This annotation tags a field containing a currency code</p> <p>This can be either an ISO code or an SAP currency code (data type CUKY).</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine): Interpreted by ABAP Runtime Environment</p> <p>Value: Boolean default true</p>						
Semantics.durationInSeconds	<p>This annotation defines whether or not a duration in seconds is displayed.</p> <p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values (optional): Boolean default true</p>						

Annotations belonging to [Semantics.email](#) follow [RFC5322](#)).

i Note

The attribute `address` should be used in case a mail address is included – independent from the specialized semantics `from`, `sender`, `to`, `cc`, or `bcc`.

Scope: [ELEMENT, PARAMETER]

Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations

Annotation	Description
Semantics.email.type	<p>Values: String (10)</p> <p>The following enumerations are provided:</p>

Annotation	Meaning	
	Value	Description
	<i>HOME</i>	Private email address
	<i>WORK</i>	Business email address
	<i>PREF</i>	Preferred email address
	<i>OTHER</i>	Other email address
<i>Semantics.email.address</i>	Boolean default true	The value of the annotation contains the addresses of the sender and the recipient of an email. An address can be an individual mailbox or a group of mailboxes.
<i>Semantics.email.bcc</i>		The value of the annotation contains the recipient list that receive an email, but whose email addresses are not revealed to other recipients of the same email.
<i>Semantics.email.body</i>		The value of the annotation contains lines of US-ASCII characters.
<i>Semantics.email.cc</i>		The value of the annotation contains the recipient list that receive an email that is not directed to them directly.
<i>Semantics.email.from</i>		The value of the annotation specifies the author(s) of a message. This can be a person or persons, or a system.
<i>Semantics.email.keywords</i>		The value of the annotated field contains a comma-separated list of one or more words or quoted strings.
<i>Semantics.email.received</i>		The value of the annotated field contains a trace information at the beginning of the message content when an SMTP server receives a message for delivery or further processing.
<i>Semantics.email.sender</i>		The value of the annotated field specifies the mailbox of the agent responsible for the actual transmission of the message. For example, if a secretary were to send a message for another person, the mailbox of the secretary would appear in the <i>Sender</i> field.
<i>Semantics.email.subject</i>		The value of the annotated field contains the topic of the message.

Annotation	Meaning
<code>Semantics.email.to</code>	The value of the annotation contains the recipient list that receive an email that is primarily directed to them directly.

Annotations belonging to `Semantics.fiscal` are required for time-based calculations in analytical use cases.

Scope: [ELEMENT, PARAMETER]

Evaluation Runtime (Engine): Some attributes contain the respective information (as plain integers), and the processor needs the semantics in order to identify them.

Values (optional): Boolean default true

<code>Semantics.fiscal.period</code>	Boolean default true	A fiscal period is covered by financial reports, for example, an annual report covers a fiscal period of one year, but a quarterly report includes accounting data for three months.
		The value of the annotated field encodes a fiscal period as a string following the logical pattern PPP consisting of three digits. This fiscal period usually is a quarter of a year.

❖ Example

The string matches the regex pattern [0-9]{3}

<code>Semantics.fiscal.year</code>	The value of the annotated field encodes a fiscal year number as a string following the logical pattern YYYY consisting of four digits.
	<p>❖ Example</p> <p>The string matches the regex pattern [1-9][0-9]{3}</p>

Annotation	Meaning
Semantics.fiscal.yearPeriod	The value of the annotated field encodes a fiscal year and period as a string following the logical pattern YYYYPPP consisting of seven digits. The last three digits represent the fiscal period in the year.
Semantics.fiscal.yearVariant	The value of the annotated field encodes a fiscal year variant, which describes the number of periods in a fiscal year and how they match the calendar year.

Annotations belong to [Semantics.geoLocation](#) contain geo-coordinates for depicting data on a map.

Scope: [ELEMENT, PARAMETER]

Evaluation Runtime (Engine):

Values:

Semantics.geoLocation.cartold	Boolean default true
Semantics.geoLocation.latitude	The value of the annotated field specifies the latitude of the location such as a city.
Semantics.geoLocation.longitude	The value of the annotated field specifies the longitude of the location such as a city.
Semantics.geoLocation.normalizedName	The value of the annotated field contains the readable name of the location.
Semantics.imageUrl	<p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine): SADL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values (optional): Boolean default true</p>
Semantics.language	<p>This annotation identifies a language.</p> <p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine): SADL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values (optional): Boolean default true</p>

Annotation	Meaning
<code>Semantics.languageReference</code>	<p>This annotation references a field that identifies languages. You can use this annotation if you cannot define a language as constant value.</p> <p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine):</p> <p>Values (optional): elementRef</p>
<code>Semantics.mimeType</code>	<p>This annotation describes the mime type of a resource (identified by a URL or directly available as binary content).</p> <p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine):</p> <p>This is required when accessing the document content, for example, in a content crawl, during text analysis, or when opening the document for viewing.</p> <p>In UIs, documents are usually presented with an icon that symbolizes their mime type.</p> <p>Values: Boolean default true</p>
Annotations belonging to <code>Semantics.name</code> follow the vCard standard (RFC6350).	
Scope: [ELEMENT, PARAMETER]	
Evaluation Runtime (Engine): SADL:	Translates CDS annotations into the corresponding OData annotations
Values:	
<code>Semantics.name.additionalName</code>	Boolean default true
	The value of the annotation contains the second first name of an individual.
<code>Semantics.name.familyName</code>	
	The value of the annotation contains the surname of an individual.
<code>Semantics.name.fullName</code>	
	The value of the annotation contains the full name of an individual.
<code>Semantics.name.givenName</code>	
	The value of the annotation contains the first name of an individual.
<code>Semantics.name.jobTitle</code>	
	The value of the annotation contains the job title of an individual.
<code>Semantics.name.nickName</code>	
	The value of the annotation contains the nickname of an individual.
<code>Semantics.name.prefix</code>	
	The value of the annotation contains the honorific prefix of an individual.
<code>Semantics.name.suffix</code>	
	The value of the annotation contains the honorific suffix of an individual.

Annotation	Meaning
Annotations belonging to <i>Semantics.organization</i> follow the vCard standard (RFC6350).	
Scope: [ELEMENT, PARAMETER]	
Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations	
Values:	
<i>Semantics.organization.name</i>	Boolean default true
The value of the annotated field contains the organization name.	
<i>Semantics.organization.role</i>	The value of the annotated field specify the function or part played in a particular situation by the object the vCard represents.
<p> Example ROLE:Project Leader</p>	
<i>Semantics.organization.unit</i>	The value of the annotated field contains the name of the organization unit.
Annotations belonging to <i>Semantics.quantity</i> contain a measured quantity, and the corresponding unit of measure is contained in the referenced field.	
Scope: [ELEMENT, PARAMETER]	
Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations	
Values:	
<i>Semantics.quantity.unitOfMeasure</i> elementRef	The value of the annotated field specifies a unit of measure related to a measured quantity.
Annotations belonging to <i>Semantics.systemDate</i> specify the date/time that is recorded by the technical infrastructure/database.	
<p> Note The sub-annotations have the same semantics as the equally named attributes of the <i>businessDate</i> annotation. The difference is that values contain the date/time of the creation/change and are recorded by the database.</p>	
Scope: [ELEMENT, PARAMETER]	
Evaluation Runtime (Engine):	
Values:	
<i>Semantics.systemDate.createdAt</i>	Boolean default true
Timestamp when database record was created.	
<i>Semantics.systemDate.lastChangeAt</i>	
Timestamp when database record was last changed.	

Annotation	Meaning																						
	<p>Annotations belonging to <i>Semantics.telephone</i> follow the vCard standard RFC5322 ()</p> <p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine): SADL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: String (10)</p>																						
<i>Semantics.telephone.type</i>	<p>Values: default #PREF</p> <p>The following enumerations are provided:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>CELL</i></td><td>Cell phone</td></tr> <tr> <td><i>FAX</i></td><td>Fax</td></tr> <tr> <td><i>HOME</i></td><td>Private phone</td></tr> <tr> <td><i>PAGER</i></td><td>Pager</td></tr> <tr> <td><i>PREF</i></td><td>Preferred phone</td></tr> <tr> <td><i>TEXT</i></td><td>Phone that supports text messages (SMS)</td></tr> <tr> <td><i>TEXT_PHONE</i></td><td>Telecommunication device for people with hearing or speech difficulties</td></tr> <tr> <td><i>VIDEO</i></td><td>Video conferencing phone</td></tr> <tr> <td><i>VOICE</i></td><td>Voice phone</td></tr> <tr> <td><i>WORK</i></td><td>Business phone</td></tr> </tbody> </table>	Value	Description	<i>CELL</i>	Cell phone	<i>FAX</i>	Fax	<i>HOME</i>	Private phone	<i>PAGER</i>	Pager	<i>PREF</i>	Preferred phone	<i>TEXT</i>	Phone that supports text messages (SMS)	<i>TEXT_PHONE</i>	Telecommunication device for people with hearing or speech difficulties	<i>VIDEO</i>	Video conferencing phone	<i>VOICE</i>	Voice phone	<i>WORK</i>	Business phone
Value	Description																						
<i>CELL</i>	Cell phone																						
<i>FAX</i>	Fax																						
<i>HOME</i>	Private phone																						
<i>PAGER</i>	Pager																						
<i>PREF</i>	Preferred phone																						
<i>TEXT</i>	Phone that supports text messages (SMS)																						
<i>TEXT_PHONE</i>	Telecommunication device for people with hearing or speech difficulties																						
<i>VIDEO</i>	Video conferencing phone																						
<i>VOICE</i>	Voice phone																						
<i>WORK</i>	Business phone																						
<i>Semantics.text</i>	<p>This annotation identifies a human-readable text that is not necessarily language-dependent.</p> <p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine): SADL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: Boolean default true</p>																						
<i>Semantics.time</i>	<p>This annotation is used to indicate a date semantic for the NVARCHAR-based ABAP type <i>TIMS</i>.</p> <p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine):</p> <p>Value : Boolean default true</p>																						

Annotation	Meaning
Semantics.unitOfMeasure	<p>This annotation tags a field as containing a unit of measure.</p> <p>i Note</p> <p>There seems to be currently no internationally recognized standard list for units of measure available.</p>
	<p>Scope: [ELEMENT, PARAMETER]</p> <p>Evaluation Runtime (Engine):</p> <p>Values: Boolean default true</p>
Annotations belonging to Semantics.url contain a URL, and its mime type is contained in the referenced second field.	
Scope: [ELEMENT, PARAMETER]	
Evaluation Runtime (Engine): SSDL:	Translates CDS annotations into the corresponding OData annotations
Values:	
Semantics.url.mimeType	<p>elementRef</p> <p>This annotated field is required when opening a document for viewing, or when accessing document content during text analysis. In UIs, documents are usually presented with an icon that symbolizes their mime type.</p>
Annotations belonging to Semantics.user define the ID of the user related to the data record.	
The attribute <code>id</code> should be used if the user ID without additional semantics is included. If the dedicated semantics of the user ID are known, the attributes <code>createdBy</code> , <code>lastChangedBy</code> or <code>responsible</code> should be used.	
Scope: [ELEMENT, PARAMETER]	
Evaluation Runtime (Engine):	
Values:	
Semantics.user.createdBy	<p>Boolean default true</p> <p>The value of the annotated field specifies who created a data record.</p>
Semantics.user.id	<p>The value of the annotated field contains the ID of a user.</p>
Semantics.user.lastChangedBy	<p>The value of the annotated field specifies who changed a data record at last.</p>
Semantics.user.responsible	<p>The value of the annotated field specifies who is the person responsible for a data record.</p>

Examples

Example 1

The following CDS view fetches the contact data. Here, the annotations assign to the corresponding fields the relevant semantic information, such as first name, last name, and so on.

↳ Sample Code

```
DEFINE VIEW ContactPerson ...
ASSOCIATION [1..1] TO FormattedName AS _FormattedName ON ...
{
  ...
  @Semantics.name.givenName
  FirstName,
  @Semantics.name.additionalName
  MiddleName,
  @Semantics.name.familyName
  LastName,
  @Semantics.user.id
  SystemUser,
  Initials,
  GenderCode,
  AddressUUID,
  @Semantics.telephone.type: [#WORK, #PREF]
  PhoneNumber,
  @Semantics.telephone.type: [#FAX]
  FaxNumber,
  @Semantics.telephone.type: [#CELL]
  MobilePhoneNumber,
  @Semantics.eMail.address
  EmailAddress,
  PreferredLanguage,
  @Semantics.contact.birthDate
  BirthDate,
  @Semantics.name.fullName
  _FormattedName.FormattedContactName,
  ...
}
```

Example 2

The following CDS view fetches sales order items. Here, the annotations assign the units and currencies to the corresponding fields.

↳ Sample Code

```
DEFINE VIEW SalesOrderItem as select from ...
{
  ...
  @Semantics.currencyCode
  currency_code as CurrencyCode,
  @Semantics.amount.currencyCode: 'CurrencyCode'
  gross_amount as GrossAmount,
  @Semantics.unitOfMeasure
  unit_of_measure as UnitOfMeasure,
  @Semantics.quantity.unitOfMeasure: 'UnitOfMeasure'
  quantity as Quantity,
```

```
    ...
}
```

Example 3

The following CDS view fetches geographic data of cities annotating the corresponding location fields according to a standardized format:

↳ Sample Code

```
DEFINE VIEW myGeoAttributeView as select from ...
{
  @Semantics.address.city
  cartoid,
  @Semantics.geoLocation.longitude
  Longitude,
  @Semantics.geoLocation.latitude
  latitude,
  @Semantics.geoLocation.normalizedName
  name
}
```

Example 4

The following CDS view fetches language-dependant data annotating the corresponding language fields and text fields:

↳ Sample Code

```
DEFINE VIEW chartOfAccountsTexts AS SELECT FROM ...
{
  key ktopl AS chartOfAccounts,
  @Semantics.language: true
  key spras AS language,
  @Semantics.text: true
  ktplt AS chartOfAccountsName
}
```

9.1.12 UI Annotations

Represent semantic views on business data through the use of specific patterns that are completely independent of UI technologies.

Scope and Definition

```
@MetadataExtension.usageAllowed : true
{
  @Scope:[#ENTITY]
  headerInfo
  {
    @LanguageDependency.maxLength : 40
  }
}
```

```

typeName : String(60);
@LanguageDependency.maxLength : 40
typeNamePlural : String(60);
typeImageUrl : String(1024);
imageUrl : ElementRef;
title
{
    type : String(40) enum
    {
        STANDARD;
        WITH_INTENT_BASED_NAVIGATION;
        WITH_NAVIGATION_PATH;
        WITH_URL;
    } default #STANDARD;
    @LanguageDependency.maxLength : 40
    label : String(60);
    iconUrl : String(1024);
    criticality : ElementRef;
    criticalityRepresentation : String(12) enum
    {
        WITHOUT_ICON;
        WITH_ICON;
    } default #WITHOUT_ICON;
    value : ElementRef;
    targetElement : ElementRef;
    url : ElementRef;
};
description
{
    type : String(40) enum
    {
        STANDARD;
        WITH_INTENT_BASED_NAVIGATION;
        WITH_NAVIGATION_PATH;
        WITH_URL;
    } default #STANDARD;
    @LanguageDependency.maxLength : 40
    label : String(60);
    iconUrl : String(1024);
    criticality : ElementRef;
    criticalityRepresentation : String(12) enum
    {
        WITHOUT_ICON;
        WITH_ICON;
    } default #WITHOUT_ICON;
    value : ElementRef;
    targetElement : ElementRef;
    url : ElementRef;
};
};

@Scope:[#ENTITY]
badge
{
    headLine
    {
        type : String(40) enum
        {
            STANDARD;
            WITH_INTENT_BASED_NAVIGATION;
            WITH_NAVIGATION_PATH;
            WITH_URL;
        } default #STANDARD;
        @LanguageDependency.maxLength : 40
        label : String(60);
        iconUrl : String(1024);
        criticality : ElementRef;
        criticalityRepresentation : String(12) enum
        {

```

```

        WITHOUT_ICON;
        WITH_ICON;
    } default #WITHOUT_ICON;
value : ElementRef;
targetElement : ElementRef;
url : ElementRef;
};
title
{
    type : String(40) enum
    {
        STANDARD;
        WITH_INTENT_BASED_NAVIGATION;
        WITH_NAVIGATION_PATH;
        WITH_URL;
    } default #STANDARD;
@LanguageDependency.maxLength : 40
label : String(60);
iconUrl : String(1024);
criticality : ElementRef;
criticalityRepresentation : String(12) enum
{
    WITHOUT_ICON;
    WITH_ICON;
} default #WITHOUT_ICON;
value : ElementRef;
targetElement : ElementRef;
url : ElementRef;
};
typeImageUrl : String(1024);
imageUrl : ElementRef;
mainInfo
{
    type : String(40) enum
    {
        STANDARD;
        WITH_INTENT_BASED_NAVIGATION;
        WITH_NAVIGATION_PATH;
        WITH_URL;
    } default #STANDARD;
@LanguageDependency.maxLength : 40
label : String(60);
iconUrl : String(1024);
criticality : ElementRef;
criticalityRepresentation : String(12) enum
{
    WITHOUT_ICON;
    WITH_ICON;
} default #WITHOUT_ICON;
value : ElementRef;
targetElement : ElementRef;
url : ElementRef;
};
secondaryInfo
{
    type : String(40) enum
    {
        STANDARD;
        WITH_INTENT_BASED_NAVIGATION;
        WITH_NAVIGATION_PATH;
        WITH_URL;
    } default #STANDARD;
@LanguageDependency.maxLength : 40
label : String(60);
iconUrl : String(1024);
criticality : ElementRef;
criticalityRepresentation : String(12) enum
{
}

```

```

        WITHOUT_ICON;
        WITH_ICON;
    } default #WITHOUT_ICON;
value : ElementRef;
targetElement : ElementRef;
url : ElementRef;
};

};

@Scope:[#ENTITY]
chart : array of
{
    qualifier : String(120);
    @LanguageDependency.maxLength : 40
    title : String(60);
    @LanguageDependency.maxLength : 80
    description : String(120);
    chartType : String(40) enum
    {
        COLUMN;
        COLUMN_STACKED;
        COLUMN_STACKED_100;
        COLUMN_DUAL;
        COLUMN_STACKED_DUAL;
        COLUMN_STACKED_DUAL_100;
        BAR;
        BAR_STACKED;
        BAR_STACKED_100;
        BAR_DUAL;
        BAR_STACKED_DUAL;
        BAR_STACKED_DUAL_100;
        AREA;
        AREA_STACKED;
        AREA_STACKED_100;
        HORIZONTAL_AREA;
        HORIZONTAL_AREA_STACKED;
        HORIZONTAL_AREA_STACKED_100;
        LINE;
        LINE_DUAL;
        COMBINATION;
        COMBINATION_STACKED;
        COMBINATION_STACKED_DUAL;
        HORIZONTAL_COMBINATION_STACKED;
        HORIZONTAL_COMBINATION_STACKED_DUAL;
        PIE;
        DONUT;
        SCATTER;
        BUBBLE;
        RADAR;
        HEAT_MAP;
        TREE_MAP;
        WATERFALL;
        BULLET;
        VERTICAL_BULLET;
        HORIZONTAL_WATERFALL;
        HORIZONTAL_COMBINATION_DUAL;
        DONUT_100;
    };
    dimensions : array of ElementRef;
    measures : array of ElementRef;
    dimensionAttributes : array of
    {
        dimension : ElementRef;
        role : String(10) enum
        {
            CATEGORY;
            SERIES;
            CATEGORY2;
        };
    };
}

```

```

        valuesForSequentialColorLevels: array of String(1024);
        emphasizedValues: array of String(1024);
    };
    measureAttributes : array of
    {
        measure : ElementRef;
        role : String(10) enum
        {
            AXIS_1;
            AXIS_2;
            AXIS_3;
        };
        asDataPoint : Boolean default true;
        useSequentialColorLevels: Boolean default true;
    };
    actions : array of
    {
        type : String(40) enum
        {
            FOR_ACTION;
            FOR_INTENT_BASED_NAVIGATION;
        };
        @LanguageDependency.maxLength : 40
        label : String(60);
        dataAction : String(120);
        requiresContext : Boolean default true;
        invocationGrouping : String(12) enum
        {
            ISOLATED;
            CHANGE_SET;
        } default #ISOLATED;
        semanticObjectAction : String(120);
    };
},
@Scope:[#ENTITY]
selectionPresentationVariant : array of
{
    qualifier : String(120);
    id : String(120);
    @LanguageDependency.maxLength : 40
    text : String(60);
    selectionVariantQualifier : String(120);
    presentationVariantQualifier : String(120);
};
@Scope:[#ENTITY]
selectionVariant : array of
{
    qualifier : String(120);
    id : String(120);
    @LanguageDependency.maxLength : 40
    text : String(60);
    parameters : array of
    {
        name : ParameterRef;
        value : String(1024);
    };
    filter : String(1024);
};
@Scope:[#ENTITY]
presentationVariant : array of
{
    qualifier : String(120);
    id : String(120);
    @LanguageDependency.maxLength : 40
    text : String(60);
    maxItems : Integer;
    sortOrder : array of
    {

```

```

        by : ElementRef;
        direction : String(4) enum
        {
            ASC;
            DESC;
        };
    groupBy : array of ElementRef;
    totalBy : array of ElementRef;
    total : array of ElementRef;
    includeGrandTotal : Boolean default true;
    initialExpansionLevel : Integer;
    requestAtLeast : array of ElementRef;
    visualizations : array of
    {
        type : String(40) enum
        {
            AS_LINEITEM;
            AS_CHART;
            AS_DATAPOINT;
        };
        qualifier : String(120);
        element : ElementRef;
    };
    selectionFieldsQualifier : String(120);
};

@Scope:[#ELEMENT, #PARAMETER]
hidden : Boolean default true;
@Scope:[#ELEMENT]
masked : Boolean default true;
@Scope:[#ELEMENT]
multiLineText : Boolean default true;
@Scope:[#ELEMENT]
lineItem : array of
{
    @Scope: [#ELEMENT, #ENTITY]
    qualifier : String(120);
    position : DecimalFloat;
    exclude : Boolean default true;
    hidden : Boolean default true;
    importance : String(6) enum { HIGH; MEDIUM; LOW; };
    type : String(40) enum
    {
        AS_ADDRESS;
        AS_CHART;
        AS_CONNECTED_FIELDS;
        AS_CONTACT;
        AS_DATAPOINT;
        AS_FIELDGROUP;
        FOR_ACTION;
        FOR_INTENT_BASED_NAVIGATION;
        STANDARD;
        WITH_INTENT_BASED_NAVIGATION;
        WITH_NAVIGATION_PATH;
        WITH_URL;
    } default #STANDARD;
    @LanguageDependency.maxLength : 40
    label : String(60);
    iconUrl : String(1024);
    @Scope: [#ELEMENT, #ENTITY]
    criticality : ElementRef;
    criticalityRepresentation : String(12) enum
    {
        WITHOUT_ICON;
        WITH_ICON;
    } default #WITHOUT_ICON;
    dataAction : String(120);
    requiresContext : Boolean default true;
}

```

```

        invocationGrouping : String(12) enum { ISOLATED; CHANGE_SET; } default
#ISOLATED;
        semanticObjectAction : String(120);
        value : ElementRef;
        valueQualifier : String(120);
        targetElement : ElementRef;
        url : ElementRef;
    };
@Scope:[#ELEMENT]
identification : array of
{
    position : DecimalFloat;
    exclude : Boolean default true;
    importance : String(6) enum { HIGH; MEDIUM; LOW; };
    type : String(40) enum
    {
        AS_ADDRESS;
        AS_CHART;
        AS_CONNECTED_FIELDS;
        AS_CONTACT;
        AS_DATAPOINT;
        AS_FIELDGROUP;
        FOR_ACTION;
        FOR_INTENT_BASED_NAVIGATION;
        STANDARD;
        WITH_INTENT_BASED_NAVIGATION;
        WITH_NAVIGATION_PATH;
        WITH_URL;
    } default #STANDARD;
    @LanguageDependency.maxLength : 40
    label : String(60);
    iconUrl : String(1024);
    criticality : ElementRef;
    criticalityRepresentation : String(12) enum
    {
        WITHOUT_ICON;
        WITH_ICON;
    } default #WITHOUT_ICON;
    dataAction : String(120);
    requiresContext : Boolean default true;
    invocationGrouping : String(12) enum { ISOLATED; CHANGE_SET; } default
#ISOLATED;
    semanticObjectAction : String(120);
    value : ElementRef;
    valueQualifier : String(120);
    targetElement : ElementRef;
    url : ElementRef;
};
@Scope:[#ELEMENT]
statusInfo : array of
{
    position : DecimalFloat;
    exclude : Boolean default true;
    importance : String(6) enum { HIGH; MEDIUM; LOW; };
    type : String(40) enum
    {
        AS_ADDRESS;
        AS_CHART;
        AS_CONNECTED_FIELDS;
        AS_CONTACT;
        AS_DATAPOINT;
        AS_FIELDGROUP;
        FOR_ACTION;
        FOR_INTENT_BASED_NAVIGATION;
        STANDARD;
        WITH_INTENT_BASED_NAVIGATION;
        WITH_NAVIGATION_PATH;
        WITH_URL;
    }
};

```

```

} default #STANDARD;
@LanguageDependency.maxLength : 40
label : String(60);
iconUrl : String(1024);
criticality : ElementRef;
criticalityRepresentation : String(12) enum
{
    WITHOUT_ICON;
    WITH_ICON;
} default #WITHOUT_ICON;
dataAction : String(120);
requiresContext : Boolean default true;
invocationGrouping : String(12) enum { ISOLATED; CHANGE_SET; } default
#ISOLATED;
semanticObjectAction : String(120);
value : ElementRef;
valueQualifier : String(120);
targetElement : ElementRef;
url : ElementRef;
};

@Scope:[#ELEMENT]
fieldGroup : array of
{
    qualifier : String(120);
    @LanguageDependency.maxLength : 40
    groupLabel : String(60);
    position : DecimalFloat;
    exclude : Boolean default true;
    importance : String(6) enum { HIGH; MEDIUM; LOW; };
    type : String(40) enum
    {
        AS_ADDRESS;
        AS_CHART;
        AS_CONNECTED_FIELDS;
        AS_CONTACT;
        AS_DATAPOINT;
        AS_FIELDGROUP;
        FOR_ACTION;
        FOR_INTENT_BASED_NAVIGATION;
        STANDARD;
        WITH_INTENT_BASED_NAVIGATION;
        WITH_NAVIGATION_PATH;
        WITH_URL;
    } default #STANDARD;
    @LanguageDependency.maxLength : 40
    label : String(60);
    iconUrl : String(1024);
    criticality : ElementRef;
    criticalityRepresentation : String(12) enum
    {
        WITHOUT_ICON;
        WITH_ICON;
    } default #WITHOUT_ICON;
    dataAction : String(120);
    requiresContext : Boolean default true;
    invocationGrouping : String(12) enum { ISOLATED; CHANGE_SET; } default
#ISOLATED;
    semanticObjectAction : String(120);
    value : ElementRef;
    valueQualifier : String(120);
    targetElement : ElementRef;
    url : ElementRef;
};
@Scope:[#ELEMENT]
dataPoint
{
    qualifier : String(120);
    @LanguageDependency.maxLength : 40
}

```

```

title : String(60);
@LanguageDependency.maxLength : 80
description : String(120);
@LanguageDependency.maxLength : 193
longDescription : String(250);
targetValue : DecimalFloat;
targetValueElement : ElementRef;
forecastValue : ElementRef;
minimumValue : DecimalFloat;
maximumValue : DecimalFloat;
visualization : String(12) enum
{
    NUMBER;
    BULLET_CHART;
    DONUT;
    PROGRESS;
    RATING;
};
valueFormat
{
    scaleFactor : DecimalFloat;
    numberOfFractionalDigits : Integer;
};
referencePeriod
{
    @LanguageDependency.maxLength : 80
    description : String(120);
    start : ElementRef;
    end : ElementRef;
};
criticality : ElementRef;
criticalityValue : Integer enum
{
    NEGATIVE;
    CRITICAL;
    POSITIVE;
};
criticalityRepresentation : String(12) enum
{
    WITHOUT_ICON;
    WITH_ICON;
} default #WITHOUT_ICON;
criticalityCalculation
{
    improvementDirection : String(8) enum
    {
        MINIMIZE;
        TARGET;
        MAXIMIZE;
    };
    acceptanceRangeLowValue : DecimalFloat;
    acceptanceRangeHighValue : DecimalFloat;
    toleranceRangeLowValue : DecimalFloat;
    toleranceRangeLowValueElement : ElementRef;
    toleranceRangeHighValue : DecimalFloat;
    toleranceRangeHighValueElement : ElementRef;
    deviationRangeLowValue : DecimalFloat;
    deviationRangeLowValueElement : ElementRef;
    deviationRangeHighValue : DecimalFloat;
    deviationRangeHighValueElement : ElementRef;
    constantThresholds: array of
    {
        aggregationLevel: array of ElementRef;
        acceptanceRangeLowValue: DecimalFloat;
        acceptanceRangeHighValue: DecimalFloat;
        toleranceRangeLowValue: DecimalFloat;
        toleranceRangeHighValue: DecimalFloat;
        deviationRangeLowValue: DecimalFloat;
    }
}

```

```

        deviationRangeHighValue: DecimalFloat;
    };

};

trend : ElementRef;
trendCalculation
{
    referenceValue : ElementRef;
    isRelativeDifference : Boolean default true;
    upDifference : DecimalFloat;
    upDifferenceElement : ElementRef;
    strongUpDifference : DecimalFloat;
    strongUpDifferenceElement : ElementRef;
    downDifference : DecimalFloat;
    downDifferenceElement : ElementRef;
    strongDownDifference : DecimalFloat;
    strongDownDifferenceElement : ElementRef;
};
responsible : ElementRef;
responsibleName : String(120);
};

@Scope:[#ELEMENT]
selectionField : array of
{
    qualifier : String(120);
    position : DecimalFloat;
    exclude : Boolean default true;
    element : ElementRef;
};
@Scope:[#ELEMENT]
facet : array of
{
    qualifier : String(120);
    @CompatibilityContract: {
        c1: { usageAllowed: false },
        c2: { usageAllowed: true,
            allowedChanges.annotation: [ #REMOVE ],
            allowedChanges.value: [ #NONE ] } }
    feature : String(40);
    id : String(120);
    purpose : String(40) enum
    {
        STANDARD;
        HEADER;
        QUICK_VIEW;
        QUICK_CREATE;
        FILTER;
    } default #STANDARD;
    parentId : String(120);
    position : DecimalFloat;
    exclude : Boolean default true;
    hidden : Boolean default true;
    isPartOfPreview : Boolean default true;
    isSummary : Boolean default true;
    isMap : Boolean default true;
    importance : String(6) enum
    {
        HIGH;
        MEDIUM;
        LOW;
    };
    @LanguageDependency.maxLength : 40
    label : String(60);
    type : String(40) enum
    {
        COLLECTION;
        ADDRESS_REFERENCE;
        BADGE_REFERENCE;
    };
}

```

```

        CHART_REFERENCE;
        CONTACT_REFERENCE;
        DATAPOINT_REFERENCE;
        FIELDGROUP_REFERENCE;
        HEADERINFO_REFERENCE;
        IDENTIFICATION_REFERENCE;
        LINEITEM_REFERENCE;
        STATUSINFO_REFERENCE;
        URL_REFERENCE;
    };
    targetElement : ElementRef;
    targetQualifier : String(120);
    url : ElementRef;
};

@Scope:[#ENTITY, #ELEMENT]
textArrangement : String(13) enum
{
    TEXT_FIRST;
    TEXT_LAST;
    TEXT_ONLY;
    TEXT_SEPARATE;
};

//=====
// Version 7.69
//=====

@Scope: [#ELEMENT]
kpi : array of
{
    qualifier          : String(120);
    id                 : String(120);
    @LanguageDependency.maxLength: 10
    shortDescription   : String(20);
    selectionVariantQualifier : String(120);
    detail
    {
        defaultPresentationVariantQualifier      : String(120);
        alternativePresentationVariantQualifiers : array of String(120);
        semanticObject           : String(120);
        semanticObjectAction     : String(120);
    };
    dataPoint
    {
        @LanguageDependency.maxLength : 40
        title                  : String(60);
        @LanguageDependency.maxLength : 80
        description             : String(120);
        @LanguageDependency.maxLength : 193
        longDescription         : String(250);
        targetValue              : DecimalFloat;
        forecastValue            : DecimalFloat;
        minimumValue             : DecimalFloat;
        maximumValue             : DecimalFloat;
        valueFormat
        {
            scaleFactor           : DecimalFloat;
            numberOfFractionalDigits : Integer;
        };
        visualization : String(12) enum
        {
            NUMBER;
            BULLET_CHART;
            DONUT;
            PROGRESS;
            RATING;
        };
        referencePeriod {
            @LanguageDependency.maxLength: 80
            description : String(120);
        }
    }
}

```

```

        start      : ElementRef;
        end        : ElementRef;
    };
    criticality           : ElementRef;
    criticalityValue      : Integer enum
    {
        NEGATIVE;
        CRITICAL;
        POSITIVE;
    };
    criticalityRepresentation : String(12) enum
    {
        WITHOUT_ICON;
        WITH_ICON;
    } default #WITHOUT_ICON;
    criticalityCalculation
    {
        improvementDirection : String(8) enum
        {
            MINIMIZE;
            TARGET;
            MAXIMIZE;
        };
        acceptanceRangeLowValue   : DecimalFloat;
        acceptanceRangeHighValue  : DecimalFloat;
        toleranceRangeLowValue    : DecimalFloat;
        toleranceRangeHighValue   : DecimalFloat;
        deviationRangeLowValue    : DecimalFloat;
        deviationRangeHighValue   : DecimalFloat;
        constantThresholds       : array of
        {
            aggregationLevel      : array of ElementRef;
            acceptanceRangeLowValue : DecimalFloat;
            acceptanceRangeHighValue: DecimalFloat;
            toleranceRangeLowValue : DecimalFloat;
            toleranceRangeHighValue: DecimalFloat;
            deviationRangeLowValue : DecimalFloat;
            deviationRangeHighValue: DecimalFloat;
        };
    };
    trend : ElementRef;
    trendCalculation
    {
        referenceValue      : ElementRef;
        isRelativeDifference: Boolean ;
        upDifference        : DecimalFloat;
        strongUpDifference  : DecimalFloat;
        downDifference      : DecimalFloat;
        strongDownDifference: DecimalFloat;
    };
    responsible     : ElementRef;
    responsibleName: String(120);
};

};

@Scope: [#ELEMENT]
valueCriticality: array of
{
    qualifier   : String(120);
    value       : String(120);
    criticality : Integer enum
    {
        NEGATIVE;
        CRITICAL;
        POSITIVE;
    };
};

```

```

@Scope: [#ELEMENT]
criticalityLabels : array of {
qualifier: String(120);
criticality: Integer enum
{
NEGATIVE;
CRITICAL;
POSITIVE;
};
@LanguageDependency.maxLength: 40
label: String(60);
};

@Scope: [#ELEMENT]
connectedFields : array of
{
qualifier : String(120);
@LanguageDependency.maxLength : 40
groupLabel : String(60);
@LanguageDependency.maxLength : 197
template : String(255);
name : String(120);
exclude : Boolean default true;
hidden : Boolean default true;
importance : String(6) enum { HIGH; MEDIUM; LOW; };
type : String(40) enum
{
AS_ADDRESS;
AS_CHART;
AS_CONNECTED_FIELDS;
AS_CONTACT;
AS_DATAPOINT;
AS_FIELDGROUP;
FOR_ACTION;
FOR_INTENT_BASED_NAVIGATION;
STANDARD;
WITH_INTENT_BASED_NAVIGATION;
WITH_NAVIGATION_PATH;
WITH_URL;
} default #STANDARD;
@LanguageDependency.maxLength : 40
label : String(60);
iconUrl : String(1024);
criticality : ElementRef;
criticalityRepresentation : String(12) enum
{
WITHOUT_ICON;
WITH_ICON;
} default #WITHOUT_ICON;
dataAction : String(120);
requiresContext : Boolean default true;
invocationGrouping : String(12) enum { ISOLATED; CHANGE_SET; } default
#ISOLATED;
semanticObjectAction : String(120);
value : ElementRef;
valueQualifier : String(120);
targetElement : ElementRef;
url : ElementRef;
};

};


```

Usage

UI Annotations

Parent Annotation	Annotation	Type	Value	Description
UI.headerInfo				<p>Annotations belonging to UI.headerInfo null describe an entity, its title, and an optional short description, the name of its entity in singular and plural form, and optional image URLs for the individual entity.</p> <p>Scope: ENTITY</p> <p>Evaluation Runtime (Engine): SADL</p> <p>Translates CDS annotations into the corresponding OData annotations</p> <p>Values:</p>
UI.headerInfo	.typeName	String (60)	#mandatory	<p>This annotation represents the title of an object page, for example.</p> <p>Note</p> <p>This annotation can be omitted only when the @End-UserText.label is specified on view level.</p>
UI.headerInfo	typeNamePlural	String (60)	#mandatory	This annotation represents a list title, for example.

Parent Annotation	Annotation	Type	Value	Description
UI.headerInfo	typeImageUrl	String (1024)	#optional	<p>This annotation contains the URL of an image representing an entity.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> i Note When users open a SAP Fiori application, they can see an image related to the entity type to which all items displayed on that page belong to. </div>
UI.headerInfo	imageUrl	elementRef	#optional	<p>This annotation represents a path to an element containing the URL of an image representing the entity instance.</p>
UI.headerInfo	imageData	elementRef		<p>This annotation allows referencing an element that contains the image BLOB and is itself annotated with <code>Semantics.largeObject</code>.</p>
UI.headerInfo	title.		<ul style="list-style-type: none"> • STANDARD • WITH_NAVIGATION_PATH • WITH_URL 	<p>A required element restricted to the types STANDARD, WITH_NAVIGATION_PATH, and WITH_URL.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.headerInfo	title.type	String (40) enum	<ul style="list-style-type: none"> • STANDARD: Maps to standard <i>DataField</i>. You use this type if you want a field to be displayed without any additional functionality. A standard <i>DataField</i> refers to a property of the OData service used. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> ○ <i>criticality</i> • WITH_URL: Maps to <i>DataFieldWithURL</i>. <i>DataFieldWithURL</i> is based on <i>DataField</i>, and defines a label–value pair that refers a property of the OData service used. The definition consists a URL to navigate to a new target, that is a URL. For more information, see With URL [page 327]. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> When you use this type, you must use the following elements: <ul style="list-style-type: none"> ○ <i>url</i> 	Annotations belonging to UI.headerInfo.title represent a property of type UI.DataFieldAbstract restricted to the types STANDARD, WITH_NAVIGATION_PATH, WITH_URL, and WITH_INVENTORY_BASED_NAVIGATION. @UI.headerInfo.title annotations are mandatory and are usually used to represent the title of an item on the header of an item's object page.. The OData annotations DataFieldAbstract are the basis for all DataField types and represent values with optional labels that can trigger navigation to related data, or execute actions on data.

Parent Annotation	Annotation	Type	Value	Description
			<ul style="list-style-type: none"> • WITH_NAVIGATION_PATH: Maps to <i>DataFieldWithNavigationPath</i>. <i>DataFieldWithNavigationPath</i> is based on <i>DataField</i>, and defines a label-value pair that refers to a property of the OData service used. The definition consists of a link to navigate to a new target, based on a navigation property provided by the OData service, or defined in the annotation file. For more information, see With Navigation Path [page 326]. <p>When you use this type, you can use the following elements:</p> <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> <p>When you use this type, you must use the following elements:</p> <ul style="list-style-type: none"> ○ <i>targetElement</i> <ul style="list-style-type: none"> • WITH_INVENTENT_BASED_NAVIGATION; 	<p>Default: STANDARD;</p>

Parent Annotation	Annotation	Type	Value	Description
UI.headerInfo	title.label	String (60)	#optional	<p>This annotation contains a language-dependent text that can be used for titles in page headers of object-page floorplans. Object-page floorplans are SAP Fiori floorplan to view, edit and create objects.</p> <p>If omitted, the label of the annotated element, or the label of the element are referenced via the value.</p>
UI.headerInfo	title.iconUrl	String (1024)	#optional	This annotation contains the URL to an icon image.
UI.headerInfo	title.criticality	elementRef	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • Neutral: 0 • Negative: 1 • Critical: 2 • Positive: 3 	<p>i Note</p> <p>This annotation can be specified if the badge headline type is STANDARD.</p>
UI.headerInfo	title.criticalityRepresentation	String (12) enum	#mandatory	<p>Defines whether criticality is represented by an icon or not.</p> <p>Default:</p> <p>WITHOUT_ICON</p>
UI.headerInfo	title.value	ElementRef		<p>This annotation refers to a value. If you refer to a value that is in the same view, specify the element name. If you use an association to refer to a value, specify the path to the element.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.headerInfo</code>	<code>title.targetElement</code>	ElementRef		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData <code>NavigationPropertyPath</code>. Using This annotation, you can link from the header part of an object view floorplan to a target element. You need to specify <code>UI.badge.headLine.targetElement</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_NAVIGATION_PATH</code>. You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.headerInfo</code>	<code>title.url</code>	ElementRef		<p>This annotation represents the path to a structural element that contains a navigation URL. You need to specify <code>UI.badge.headLine.url</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_URL</code>.</p>
<code>UI.headerInfo</code>	<code>description</code>			An optional DataField restricted to the types STANDARD, WITH_NAVIGATION_PATH, and WITH_URL

Parent Annotation	Annotation	Type	Value	Description
UI.headerInfo	description.type	String (40) enum	<ul style="list-style-type: none"> • STANDARD: Maps to standard <i>DataField</i>. You use this type if you want a field to be displayed without any additional functionality. A standard <i>DataField</i> refers to a property of the OData service used. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> ○ <i>criticality</i> • WITH_URL: Maps to <i>DataFieldWithURL</i>. <i>DataFieldWithURL</i> is based on <i>DataField</i>, and defines a label–value pair that refers a property of the OData service used. The definition consists a URL to navigate to a new target, that is a URL. For more information, see With URL [page 327]. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> When you use this type, you must use the following elements: <ul style="list-style-type: none"> ○ <i>url</i> 	Annotations belonging to <code>UI.headerInfo.title</code> represent a property of type <code>UI.DataFieldAbstract</code> restricted to the types STANDARD, WITH_NAVIGATION_PATH, WITH_URL, and WITH_INVENTION_BASED_NAVIGATION. @ <code>UI.headerInfo.title</code> annotations are mandatory and are usually used to represent the title of an item on the header of an item's object page.. The OData annotations <code>DataFieldAbstract</code> are the basis for all <code>DataField</code> types and represent values with optional labels that can trigger navigation to related data, or execute actions on data.

Parent Annotation	Annotation	Type	Value	Description
			<ul style="list-style-type: none"> • WITH_NAVIGATION_PATH: Maps to <i>DataFieldWithNavigationPath</i>. <i>DataFieldWithNavigationPath</i> is based on <i>DataField</i>, and defines a label-value pair that refers to a property of the OData service used. The definition consists of a link to navigate to a new target, based on a navigation property provided by the OData service, or defined in the annotation file. For more information, see With Navigation Path [page 326]. <p>When you use this type, you can use the following elements:</p> <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> <p>When you use this type, you must use the following elements:</p> <ul style="list-style-type: none"> ○ <i>targetElement</i> <ul style="list-style-type: none"> • WITH_INVENTENT_BASED_NAVIGATION; 	<p>Default: STANDARD;</p>

Parent Annotation	Annotation	Type	Value	Description
UI.headerInfo	description.label	String (60)	#optional	<p>This annotation contains a language-dependent text that can be used for titles in page headers of object-page floorplans. Object-page floorplans are SAP Fiori floorplan to view, edit and create objects.</p> <p>If omitted, the label of the annotated element, or the label of the element are referenced via the value.</p>
UI.headerInfo	description.iconUrl	String (1024)	#optional	This annotation contains the URL to an icon image.
UI.headerInfo	description.criticality	elementRef	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • Neutral: 0 • Negative: 1 • Critical: 2 • Positive: 3 	<p>i Note</p> <p>This annotation can be specified if the badge headline type is STANDARD.</p>
UI.headerInfo	description.criticalityR	String (12) enum epresentation	#mandatory	<p>Defines whether criticality is represented by an icon or not.</p> <p>Default:</p> <p>WITHOUT_ICON</p>
UI.headerInfo	description.value	ElementRef		<p>This annotation refers to a value. If you refer to a value that is in the same view, specify the element name. If you use an association to refer to a value, specify the path to the element.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.headerInfo	description.targetElement	ElementRef		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData NavigationPropertyPath. Using This annotation, you can link from the header part of an object view floorplan to a target element. You need to specify UI.badge.headLine.targetElement when you use the annotation UI.badge.headLine.type of type WITH_NAVIGATION_PATH. You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
UI.headerInfo	description.url	ElementRef		<p>This annotation represents the path to a structural element that contains a navigation URL. You need to specify UI.badge.headLine.url when you use the annotation UI.badge.headLine.type of type WITH_URL.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>.headLine.type</code>	String (40) enum	<ul style="list-style-type: none"> • STANDARD: Maps to standard <code>DataField</code>. You use this type if you want a field to be displayed without any additional functionality. A standard <code>DataField</code> refers to a property of the OData service used. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ◦ <code>label</code> ◦ <code>value</code> ◦ <code>criticality</code> • WITH_URL: Maps to <code>DataFieldWithURL</code>. <code>DataFieldWithURL</code> is based on <code>DataField</code>, and defines a label–value pair that refers a property of the OData service used. The definition consists a URL to navigate to a new target, that is a URL. For more information, see With URL [page 327]. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ◦ <code>label</code> ◦ <code>value</code> When you use this type, you must use the following elements: <ul style="list-style-type: none"> ◦ <code>url</code> 	<p>This annotation is used to define, what can be shown in the title of a table or list. <code>UI.badge.headLine</code> represent a property of type <code>UI.DataFieldAbstract</code> restricted to the types <code>STANDARD</code>, <code>WITH_NAVIGATION_PATH</code>, and <code>WITH_URL</code>.</p> <p>The OData annotations <code>DataFieldAbstract</code> are the basis for all <code>DataField</code> types and represent values with optional labels that can trigger navigation to related data, or execute actions on data.</p>

Parent Annotation	Annotation	Type	Value	Description
			<ul style="list-style-type: none"> • WITH_NAVIGATION_PATH: Maps to <i>DataFieldWithNavigationPath</i>. <i>DataFieldWithNavigationPath</i> is based on <i>DataField</i>, and defines a label-value pair that refers to a property of the OData service used. The definition consists of a link to navigate to a new target, based on a navigation property provided by the OData service, or defined in the annotation file. For more information, see With Navigation Path [page 326]. <p>When you use this type, you can use the following elements:</p> <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> <p>When you use this type, you must use the following elements:</p> <ul style="list-style-type: none"> ○ <i>targetElement</i> <ul style="list-style-type: none"> • WITH_INVENTENT_BASED_NAVIGATION; 	<p>Default: STANDARD;</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>.headLine.label</code>	String (60)	#Optional	This annotation contains a language-dependent text. If omitted, the label of the annotated element, or the label of the element referenced via the value is used. The element is optional.
<code>UI.badge</code>	<code>.headLine.iconURL</code>	String (1024)	#Optional	This annotation contains the URL to an icon image. This annotation is optional .
<code>UI.badge</code>	<code>.headLine.criticality</code>	ElementRef	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • <code>Neutral</code>: 0 • <code>Negative</code>: 1 • <code>Critical</code>: 2 • <code>Positive</code>: 3 	i Note This annotation can be specified if the badge headline type is STANDARD .
<code>UI.badge</code>	<code>.headLine.criticalityRe</code>	String (12) enum presentation	<ul style="list-style-type: none"> • WITHOUT_ICON • WITH_ICON <p>Default: WITHOUT_ICON</p>	Defines whether criticality is represented by an icon or not.
<code>UI.badge</code>	<code>.headLine.value</code>	ElementRef		This annotation refers to a value. If you refer to a value that is in the same view, specify the element name. If you use an association to refer to a value, specify the path to the element.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>.headLine.targetElement</code>	ElementRef		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData <code>NavigationPropertyPath</code>. Using This annotation, you can link from the header part of an object view floorplan to a target element. You need to specify <code>UI.badge.headLine.targetElement</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_NAVIGATION_PATH</code>. You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.badge</code>	<code>.headLine.url</code>	ElementRef		<p>This annotation represents the path to a structural element that contains a navigation URL. You need to specify <code>UI.badge.headLine.url</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_URL</code>.</p>
<code>UI.badge</code>	<code>title.</code>		<ul style="list-style-type: none"> • STANDARD • WITH_NAVIGATION_PATH • WITH_URL 	A required DataField restricted to the types, <code>WITH_NAVIGATION_PATH</code> , and <code>WITH_URL</code> .

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>title.type</code>	String (40) enum	<ul style="list-style-type: none"> • STANDARD: Maps to standard <code>DataField</code>. You use this type if you want a field to be displayed without any additional functionality. A standard <code>DataField</code> refers to a property of the OData service used. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ◦ <code>label</code> ◦ <code>value</code> ◦ <code>criticality</code> • WITH_URL: Maps to <code>DataFieldWithURL</code>. <code>DataFieldWithURL</code> is based on <code>DataField</code>, and defines a label–value pair that refers a property of the OData service used. The definition consists a URL to navigate to a new target, that is a URL. For more information, see With URL [page 327]. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ◦ <code>label</code> ◦ <code>value</code> When you use this type, you must use the following elements: <ul style="list-style-type: none"> ◦ <code>url</code> 	Annotations belonging to <code>UI.headerInfo.title</code> represent a property of type <code>UI.DataFieldAbstract</code> restricted to the types <code>STANDARD</code> , <code>WITH_NAVIGATION_PATH</code> , <code>WITH_URL</code> , and <code>WITH_INVENTORY_BASED_NAVIGATION</code> . <code>@UI.headerInfo.title</code> annotations are mandatory and are usually used to represent the title of an item on the header of an item's object page.. The OData annotations <code>DataFieldAbstract</code> are the basis for all <code>DataField</code> types and represent values with optional labels that can trigger navigation to related data, or execute actions on data.

Parent Annotation	Annotation	Type	Value	Description
			<ul style="list-style-type: none"> • WITH_NAVIGATION_PATH: Maps to <i>DataFieldWithNavigationPath</i>. <i>DataFieldWithNavigationPath</i> is based on <i>DataField</i>, and defines a label-value pair that refers to a property of the OData service used. The definition consists of a link to navigate to a new target, based on a navigation property provided by the OData service, or defined in the annotation file. For more information, see With Navigation Path [page 326]. <p>When you use this type, you can use the following elements:</p> <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> <p>When you use this type, you must use the following elements:</p> <ul style="list-style-type: none"> ○ <i>targetElement</i> <ul style="list-style-type: none"> • WITH_INVENTENT_BASED_NAVIGATION; 	<p>Default: STANDARD;</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>title.label</code>	<code>String(60)</code>	<code>#optional</code>	<p>This annotation contains a language-dependent text that can be used for titles in page headers of object-page floorplans. Object-page floorplans are SAP Fiori floorplan to view, edit and create objects.</p> <p>If omitted, the label of the annotated element, or the label of the element are referenced via the value.</p>
<code>UI.badge</code>	<code>title.iconUrl</code>	<code>String(1024)</code>	<code>#optional</code>	This annotation contains the URL to an icon image.
<code>UI.badge</code>	<code>title.criticality</code>	<code>elementRef</code>	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • <code>Neutral</code>: 0 • <code>Negative</code>: 1 • <code>Critical</code>: 2 • <code>Positive</code>: 3 	<p>i Note</p> <p>This annotation can be specified if the badge headline type is <code>STANDARD</code>.</p>
<code>UI.badge</code>	<code>title.criticalityRepresentation</code>	<code>String(12) enum</code>	<code>#mandatory</code>	<p>Defines whether criticality is represented by an icon or not.</p> <p>Default:</p> <p><code>WITHOUT_ICON</code></p>
<code>UI.badge</code>	<code>title.value</code>	<code>ElementRef</code>		<p>This annotation refers to a value. If you refer to a value that is in the same view, specify the element name. If you use an association to refer to a value, specify the path to the element.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>title.targetElement</code>	ElementRef		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData <code>NavigationPropertyPath</code>. Using This annotation, you can link from the header part of an object view floorplan to a target element. You need to specify <code>UI.badge.headLine.targetElement</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_NAVIGATION_PATH</code>. You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.badge</code>	<code>title.url</code>	ElementRef		<p>This annotation represents the path to a structural element that contains a navigation URL. You need to specify <code>UI.badge.headLine.url</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_URL</code>.</p>
<code>UI.badge</code>	<code>typeImageUrl</code>	String(1024) ; #optional		<p>This annotation contains the URL of an image representing an entity.</p>
<code>UI.badge</code>	<code>imageUrl</code>	ElementRef	#optional	<p>This annotation represents a path to an element containing the URL of an image representing the entity instance.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>mainInfo</code>			<p>The content of <code>UI.badge.mainInfo</code> annotations is highlighted on the badge. These annotations represent a property of type <code>UI.DataFieldAbstract</code> restricted to the types <code>STANDARD</code>, <code>WITH_NAVIGATION_PATH</code>, and <code>WITH_URL</code>.</p> <p>The OData annotations <code>DataFieldAbstract</code> are the basis for all <code>DataField</code> types and represent values with optional labels that can trigger navigation to related data, or execute actions on data.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>mainInfo.type</code>	String (40) enum	<ul style="list-style-type: none"> • STANDARD: Maps to standard <code>DataField</code>. You use this type if you want a field to be displayed without any additional functionality. A standard <code>DataField</code> refers to a property of the OData service used. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ◦ <code>label</code> ◦ <code>value</code> ◦ <code>criticality</code> • WITH_URL: Maps to <code>DataFieldWithURL</code>. <code>DataFieldWithURL</code> is based on <code>DataField</code>, and defines a label–value pair that refers a property of the OData service used. The definition consists a URL to navigate to a new target, that is a URL. For more information, see With URL [page 327]. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ◦ <code>label</code> ◦ <code>value</code> When you use this type, you must use the following elements: <ul style="list-style-type: none"> ◦ <code>url</code> 	Annotations belonging to <code>UI.headerInfo.title</code> represent a property of type <code>UI.DataFieldAbstract</code> restricted to the types <code>STANDARD</code> , <code>WITH_NAVIGATION_PATH</code> , <code>WITH_URL</code> , and <code>WITH_INVENTORY_BASED_NAVIGATION</code> . <code>@UI.headerInfo.title</code> annotations are mandatory and are usually used to represent the title of an item on the header of an item's object page.. The OData annotations <code>DataFieldAbstract</code> are the basis for all <code>DataField</code> types and represent values with optional labels that can trigger navigation to related data, or execute actions on data.

Parent Annotation	Annotation	Type	Value	Description
			<ul style="list-style-type: none"> • WITH_NAVIGATION_PATH: Maps to <i>DataFieldWithNavigationPath</i>. <i>DataFieldWithNavigationPath</i> is based on <i>DataField</i>, and defines a label-value pair that refers to a property of the OData service used. The definition consists of a link to navigate to a new target, based on a navigation property provided by the OData service, or defined in the annotation file. For more information, see With Navigation Path [page 326]. <p>When you use this type, you can use the following elements:</p> <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> <p>When you use this type, you must use the following elements:</p> <ul style="list-style-type: none"> ○ <i>targetElement</i> <ul style="list-style-type: none"> • WITH_INVENTENT_BASED_NAVIGATION; 	<p>Default: STANDARD;</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>mainInfo.label</code>	<code>String(60)</code>	<code>#optional</code>	<p>This annotation contains a language-dependent text that can be used for titles in page headers of object-page floorplans. Object-page floorplans are SAP Fiori floorplan to view, edit and create objects.</p> <p>If omitted, the label of the annotated element, or the label of the element are referenced via the value.</p>
<code>UI.badge</code>	<code>mainInfo.iconUrl</code>	<code>String(1024)</code>	<code>#optional</code>	This annotation contains the URL to an icon image.
<code>UI.badge</code>	<code>mainInfo.criticality</code>	<code>elementRef</code>	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • <code>Neutral</code>: 0 • <code>Negative</code>: 1 • <code>Critical</code>: 2 • <code>Positive</code>: 3 	<p>i Note</p> <p>This annotation can be specified if the badge headline type is <code>STANDARD</code>.</p>
<code>UI.badge</code>	<code>mainInfo.criticalityRepresentation</code>	<code>String(12) enum</code>	<code>#mandatory</code>	<p>Defines whether criticality is represented by an icon or not.</p> <p>Default:</p> <p><code>WITHOUT_ICON</code></p>
<code>UI.badge</code>	<code>mainInfo.value</code>	<code>ElementRef</code>		<p>This annotation refers to a value. If you refer to a value that is in the same view, specify the element name. If you use an association to refer to a value, specify the path to the element.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>mainInfo.targetElement</code>	ElementRef		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData <code>NavigationPropertyPath</code>. Using This annotation, you can link from the header part of an object view floorplan to a target element. You need to specify <code>UI.badge.headLine.targetElement</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_NAVIGATION_PATH</code>. You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.badge</code>	<code>mainInfo.url</code>	ElementRef		<p>This annotation represents the path to a structural element that contains a navigation URL. You need to specify <code>UI.badge.headLine.url</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_URL</code>.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>secondaryInfo</code>			<p>The content of <code>UI.badge.secondaryInfo</code> annotations is subordinate to the content of the <code>UI.badge.mainInfo</code> annotations. This annotation represents a property of type <code>UI.DataFieldAbstract</code> restricted to the types <code>STANDARD</code>, <code>WITH_NAVIGATION_PATH</code>, and <code>WITH_URL</code>.</p> <p>The OData annotations <code>DataFieldAbstract</code> are the basis for all <code>DataField</code> types and represent values with optional labels that can trigger navigation to related data, or execute actions on data.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>secondaryInfo.type</code>	<code>String(40)</code> enum	<ul style="list-style-type: none"> • STANDARD: Maps to standard <code>DataField</code>. You use this type if you want a field to be displayed without any additional functionality. A standard <code>DataField</code> refers to a property of the OData service used. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ◦ <code>label</code> ◦ <code>value</code> ◦ <code>criticality</code> • WITH_URL: Maps to <code>DataFieldWithURL</code>. <code>DataFieldWithURL</code> is based on <code>DataField</code>, and defines a label–value pair that refers a property of the OData service used. The definition consists a URL to navigate to a new target, that is a URL. For more information, see With URL [page 327]. When you use this type, you can use the following elements: <ul style="list-style-type: none"> ◦ <code>label</code> ◦ <code>value</code> When you use this type, you must use the following elements: <ul style="list-style-type: none"> ◦ <code>url</code> 	Annotations belonging to <code>UI.headerInfo.title</code> represent a property of type <code>UI.DataFieldAbstract</code> restricted to the types <code>STANDARD</code> , <code>WITH_NAVIGATION_PATH</code> , <code>WITH_URL</code> , and <code>WITH_INVENTORY_BASED_NAVIGATION</code> . <code>@UI.headerInfo.title</code> annotations are mandatory and are usually used to represent the title of an item on the header of an item's object page.. The OData annotations <code>DataFieldAbstract</code> are the basis for all <code>DataField</code> types and represent values with optional labels that can trigger navigation to related data, or execute actions on data.

Parent Annotation	Annotation	Type	Value	Description
			<ul style="list-style-type: none"> • WITH_NAVIGATION_PATH: Maps to <i>DataFieldWithNavigationPath</i>. <i>DataFieldWithNavigationPath</i> is based on <i>DataField</i>, and defines a label-value pair that refers to a property of the OData service used. The definition consists of a link to navigate to a new target, based on a navigation property provided by the OData service, or defined in the annotation file. For more information, see With Navigation Path [page 326]. <p>When you use this type, you can use the following elements:</p> <ul style="list-style-type: none"> ○ <i>label</i> ○ <i>value</i> <p>When you use this type, you must use the following elements:</p> <ul style="list-style-type: none"> ○ <i>targetElement</i> <ul style="list-style-type: none"> • WITH_INVENTENT_BASED_NAVIGATION; 	<p>Default: STANDARD;</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>secondaryInfo.label</code>	<code>String(60)</code>	<code>#optional</code>	<p>This annotation contains a language-dependent text that can be used for titles in page headers of object-page floorplans. Object-page floorplans are SAP Fiori floorplan to view, edit and create objects.</p> <p>If omitted, the label of the annotated element, or the label of the element are referenced via the value.</p>
<code>UI.badge</code>	<code>secondaryInfo.iconUrl</code>	<code>String(1024)</code>	<code>#optional</code>	This annotation contains the URL to an icon image.
<code>UI.badge</code>	<code>secondaryInfo.criticality</code>	<code>elementRef</code>	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • <code>Neutral</code>: 0 • <code>Negative</code>: 1 • <code>Critical</code>: 2 • <code>Positive</code>: 3 	<p>i Note</p> <p>This annotation can be specified if the badge headline type is <code>STANDARD</code>.</p>
<code>UI.badge</code>	<code>secondaryInfo.criticalityRepresentation</code>	<code>String(12)</code> enum	<code>#mandatory</code>	<p>Defines whether criticality is represented by an icon or not.</p> <p>Default:</p> <p><code>WITHOUT_ICON</code></p>
<code>UI.badge</code>	<code>secondaryInfo.value</code>	<code>ElementRef</code>		<p>This annotation refers to a value. If you refer to a value that is in the same view, specify the element name. If you use an association to refer to a value, specify the path to the element.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.badge</code>	<code>secondaryInfo.targetElement</code>	ElementRef		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData <code>NavigationPropertyPath</code>. Using This annotation, you can link from the header part of an object view floorplan to a target element. You need to specify <code>UI.badge.headLine.targetElement</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_NAVIGATION_PATH</code>. You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.badge</code>	<code>secondaryInfo.url</code>	ElementRef		<p>This annotation represents the path to a structural element that contains a navigation URL. You need to specify <code>UI.badge.headLine.url</code> when you use the annotation <code>UI.badge.headLine.type</code> of type <code>WITH_URL</code>.</p>
<code>UI.datapoint</code>	<code>qualifier</code>	String(120)		<p>The qualifier is optional. In case the qualifier is set, it is used as OData term qualifier. If the qualifier is not set the name of the annotated element is used as OData term qualifier.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.datapoint</code>	<code>title</code>	<code>String(60)</code>	#mandatory	<p>i Note The element can be omitted only if the <code>@EndUserText.label</code> is specified.</p>
<code>UI.datapoint</code>	<code>description</code>	<code>String(120)</code>	#optional	This annotation contains a description of the data point. If omitted, the <code>@EndUserText.quickinfo</code> is used, if specified.
<code>UI.datapoint</code>	<code>longDescription</code>	<code>String(250)</code>	#optional	This annotation contains a detailed description of the data point.
<code>UI.datapoint</code>	<code>targetValue</code>	<code>DecimalFloat</code>	#not compatible with <code>UI.dataPoint.targetValue</code>	<p>⚠ Caution If you use this annotation, do not use the element <code>UI.dataPoint.targetValueElement</code>.</p> <p>→ Tip You create a KPI in which you specify a certain revenue that needs to be reached at the end of a specific year. This is the <code>UI.dataPoint.targetValue</code> that is a static value</p>
<code>UI.datapoint</code>	<code>forecastValue</code>	<code>ElementRef</code>		This annotation references a value such as predicted or intended quarterly results, for example.
<code>UI.datapoint</code>	<code>minimumValue</code>	<code>DecimalFloat</code>		This annotation specifies the minimum value of a threshold.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.datapoint</code>	<code>maximumValue</code>	DecimalFloat		This annotation specifies the maximum value of a threshold.
<code>UI.datapoint</code>	<code>valueFormat</code>	DecimalFloat	#optional	All <code>UI.dataPoint.valueFormat</code> annotations are optional. For more information about value formats, see Person Responsible and Reference Period [page 321] .
<code>UI.datapoint</code>	<code>valueFormat.scaleFact</code> or	DecimalFloat	#optional	This annotation contains the scale factor for the value, e.g. A value 1000 displayed with scaleFactor = 1000 is displayed as 1k.
<code>UI.datapoint</code>	<code>valueFormat.numberOfFractionalDigits</code>	Integer	#optional	This annotation contains the number of fractional digits to be displayed. If the element value is 1, one decimal place is rendered, for example, 34.5.

Parent Annotation	Annotation	Type	Value	Description
UI.datapoint	visualization	String(12) enum	<ul style="list-style-type: none"> • NUMBER: A data point is visualized as a number. <p>⚠ Caution</p> <p>The following visualizations require the annotation UI.dataPoint.targetValue.</p> <ul style="list-style-type: none"> • BULLET_CHART: A data point is visualized as a bullet chart. • DONUT: A data point is visualized as a donut chart. • PROGRESS: A data point is visualized as a progress indicator. • RATING: A data point is visualized as partly or completely filled symbols such as stars or hearts. 	This annotation defines the preferred visualization of a data point.
UI.datapoint	referencePeriod	#optional		All UI.dataPoint.referencePeriod annotations are optional. You either use UI.dataPoint.referencePeriod.description, or UI.dataPoint.referencePeriod.start and UI.dataPoint.referencePeriod.end.

Parent Annotation	Annotation	Type	Value	Description
<i>UI.datapoint</i>	<i>referencePeriod.description</i>	String (120)	#optional	This annotation describes the business period of evaluation, for example "Oct 2012". Typical patterns are calendar dates or fiscal dates.
<i>UI.datapoint</i>	<i>referencePeriod.start</i>	elementRef	#optional	This annotation contains a reference to the end date of the reference period.
<i>UI.datapoint</i>	<i>referencePeriod.end</i>	elementRef	#optional	This annotation contains a reference to the start date of the reference period.
<i>UI.datapoint</i>	<i>criticality</i>	elementRef	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • Neutral: 0 • Negative: 1 • Critical: 2 • Positive: 3 	i Note This annotation can be specified if the badge headline type is STANDARD .
<i>UI.datapoint</i>	<i>criticalityValue</i>	Integer enum	<ul style="list-style-type: none"> • <i>Negative</i> • <i>Critical</i> • <i>Positive</i> 	
<i>UI.datapoint</i>	<i>criticalityRepresentation</i>	String enum	#mandatory <ul style="list-style-type: none"> • WITHOUT_ICON • WITH_ICON 	Defines whether criticality is represented by an icon or not. Default: WITHOUT_ICON

Parent Annotation	Annotation	Type	Value	Description
UI.datapoint	criticalityCalculation			Annotations belonging to UI.dataPoint.criticalityCalculation can be used as an alternative to specifying the criticality in the criticality element. The criticality can be calculated based on the values of the criticalityCalculation annotations.
UI.datapoint	criticalityCalculation.improvementDirection	String (8) enum	<ul style="list-style-type: none"> • MINIMIZE • TARGET • MAXIMIZE 	<p>Description: This annotation calculates the criticality based on a specified improvement direction. For more information, see Trend-Criticality Calculation [page 319].</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.datapoint</code>	<code>criticalityCalculation.acceptanceRangeLowValue</code>	DecimalFloat	<p>Calculation rule for criticality (Value is the value of the DataPoint):</p> <p>ImprovementDirection = Target:</p> <ul style="list-style-type: none"> • <i>Positive</i>: Value > than AcceptanceRangeLowValue and < than AcceptanceRangeHighValue • <i>Neutral</i>: <ol style="list-style-type: none"> 1. Value > ToleranceRangeLowValue and Value < AcceptanceRangeLowValue 2. Value > AcceptanceRangeHighValue and Value < ToleranceRangeHighValue • <i>Critical</i> <ol style="list-style-type: none"> 1. Value > DeviationRangeLowValue and Value < ToleranceRangeLowValue 2. Value > ToleranceRangeHighValue and Value < DeviationRangeHighValue • <i>Negative</i>: Value < DeviationRangeLowValue and Value > DeviationRangeHighValue <p>ImprovementDirection = Minimize:</p> <ul style="list-style-type: none"> • <i>Positive</i>: Value > AcceptanceRangeHighValue 	Threshold as a constant (or reference to an element),

Parent Annotation	Annotation	Type	Value	Description
			<ul style="list-style-type: none"> • <i>Neutral</i>: Value > AcceptanceRangeHighValue and Value < ToleranceRangeHighValue • <i>Critical</i>: Value > ToleranceRangeHighValue and Value < DeviationRangeHighValue • <i>Negative</i>: Value > DeviationRangeHighValue <p>ImprovementDirection = Maximize:</p> <ul style="list-style-type: none"> • <i>Positive</i>: Value > AcceptanceRangeLowValue • <i>Neutral</i>: Value < AcceptanceRangeLowValue and Value > ToleranceRangeLowValue • <i>Critical</i>: Value < ToleranceRangeLowValue and Value > DeviationRangeLowValue • <i>Negative</i>: Value < DeviationRangeLowValue 	
<i>UI.datapoint</i>	<i>criticalityCalculation.acceptanceRangeHighValue</i>	DecimalFloat	For calculation, see <i>UI.datapoint.criticalityCalculation.acceptanceRangeLowValue</i>	Threshold as a constant (or reference to an element),
<i>UI.datapoint</i>	<i>criticalityCalculation.toleranceRangeLowValue</i>	DecimalFloat	For calculation, see <i>UI.datapoint.criticalityCalculation.acceptanceRangeLowValue</i>	Threshold as a constant (or reference to an element),
<i>UI.datapoint</i>	<i>criticalityCalculation.toleranceRangeHighValue</i>	DecimalFloat	For calculation, see <i>UI.datapoint.criticalityCalculation.acceptanceRangeLowValue</i>	Threshold as a constant (or reference to an element),
<i>UI.datapoint</i>	<i>criticalityCalculation.deviationRangeLowValue</i>	DecimalFloat	For calculation, see <i>UI.datapoint.criticalityCalculation.acceptanceRangeLowValue</i>	Threshold as a constant (or reference to an element),

Parent Annotation	Annotation	Type	Value	Description
<code>UI.datapoint</code>	<code>criticalityCalculation.deviationRangeHighValue</code>	DecimalFloat	For calculation, see <code>UI.datapoint.criticalityCalculation.acceptanceRangeLowValue</code>	Threshold as a constant (or reference to an element),
<code>UI.datapoint</code>	<code>criticalityCalculation.constantThresholds</code>	(array of)	<p>Calculation rule for trend (Value is the value of the DataPoint):</p> <p><code>if isRelativeDifference</code> then CompareValue := Value div Reference-Value else CompareValue := Value sub Reference-Value</p> <p><code>StrongUp</code>: CompareValue > StrongUpDifference</p> <p><code>Up</code>: CompareValue < StrongUpDifference and CompareValue > UpDifference</p> <p><code>Sideways</code>: CompareValue < UpDifference and CompareValue > DownDifference</p> <p><code>Down</code>: CompareValue > StrongDownDifference and CompareValue < DownDifference</p> <p><code>StrongDown</code>: CompareValue < StrongDownDifference</p>	<p>Thresholds depending on the aggregation level as a set of constant values. (Constant thresholds should only be used in order to refine constant values given for the datapoint overall (aggregationLevel []), but not if the thresholds are based on other measure elements.</p>

Parent Annotation	Annotation	Type	Value	Description
<i>UI.datapoint</i>	<i>criticalityCalculation.constantThresholds.aggregationLevel</i>	(array of) elementRef		An unordered tuple of dimensions, i.e. elements which are intended to be used for group-by in aggregating requests. (In analytical UIs, e.g. an analytical chart, the aggregation level typically corresponds to the visible dimensions.)
<i>UI.datapoint</i>	<i>criticalityCalculation.constantThresholds.acceptanceRangeLowValue</i>	DecimalFloat	For calculation see, <i>UI.datapoint.criticalityCalculation.constantThresholds</i>	Threshold as a constant (or reference to an element),
<i>UI.datapoint</i>	<i>criticalityCalculation.constantThresholds.acceptanceRangeHighValue</i>	DecimalFloat	For calculation see, <i>UI.datapoint.criticalityCalculation.constantThresholds</i>	Threshold as a constant (or reference to an element),
<i>UI.datapoint</i>	<i>criticalityCalculation.constantThresholds.toleranceRangeLowValue</i>	DecimalFloat	For calculation see, <i>UI.datapoint.criticalityCalculation.constantThresholds</i>	Threshold as a constant (or reference to an element),
<i>UI.datapoint</i>	<i>criticalityCalculation.constantThresholds.toleranceRangeHighValue</i>	DecimalFloat	For calculation see, <i>UI.datapoint.criticalityCalculation.constantThresholds</i>	Threshold as a constant (or reference to an element),
<i>UI.datapoint</i>	<i>criticalityCalculation.deviationRangeLowValue</i>	DecimalFloat	For calculation see, <i>UI.datapoint.criticalityCalculation.constantThresholds</i>	Threshold as a constant (or reference to an element),
<i>UI.datapoint</i>	<i>criticalityCalculation.deviationRangeHighValue</i>	DecimalFloat	<i>UI.datapoint.criticalityCalculation.constantThresholds</i>	Threshold as a constant (or reference to an element),
<i>UI.datapoint</i>	<i>trend</i>	elementRef	<ul style="list-style-type: none"> • 1: StrongUp • 2: Up • 3: Sideways • 4: Down • 5: StrongDown 	Reference to an element to visualize a trend in form of an arrow. For more information, see Trends [page 317] .

Parent Annotation	Annotation	Type	Value	Description
<code>UI.datapoint</code>	<code>trendCalculation</code>			As an alternative to specifying the trend in the <code>trend</code> element, the trend could be calculated based on the <code>trendCalculation</code> annotations.
<code>UI.datapoint</code>	<code>trendCalculation.referenceValue</code>	<code>elementReference</code>		This annotation specifies the reference value for the trend calculation as a reference to an element.
<code>UI.datapoint</code>	<code>trendCalculation.isRelativeDifference</code>	<code>boolean</code>	Default: False	This boolean constant expresses whether the difference between the value and <code>referenceValue</code> is absolute or relative.
<code>UI.datapoint</code>	<code>trendCalculation.upDiff</code>	<code>DecimalFloat</code>		This annotation specifies a threshold as a constant.
<code>UI.datapoint</code>	<code>trendCalculation strongUpDifference</code>	<code>DecimalFloat</code>		This annotation specifies a threshold as a constant.
<code>UI.datapoint</code>	<code>trendCalculation.downDifference</code>	<code>DecimalFloat</code>		This annotation specifies a threshold as a constant.
<code>UI.datapoint</code>	<code>trendCalculation.strongDownDifference</code>	<code>DecimalFloat</code>		This annotation specifies a threshold as a constant.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.datapoint</code>	<code>responsible</code>	ElementRef	# not compatible with <code>UI.dataPoint.responsibleName</code>	<p>This annotation contains an association to an entity that is annotated with @Semantics.name, @Semantics.eMail, @Semantics.telephone, @Semantics.address, or @Semantics.organization.</p> <p>⚠ Caution</p> <p>If you use this annotation, you can't use the annotation <code>UI.dataPoint.responsibleName</code>.</p> <p>For more information, see Person Responsible and Reference Period [page 321].</p> <p>For an overview of @Semantics annotations, see Semantics Annotations [page 445].</p>
<code>UI.datapoint</code>	<code>responsibleName</code>	String (120)	# not compatible with <code>UI.dataPoint.responsible</code>	<p>This annotation can be used as an alternative to the responsible element. Only the name of the responsible person can be specified here.</p> <p>⚠ Caution</p> <p>If you use this annotation, you can't use the annotation <code>UI.dataPoint.responsible</code>.</p>

Parent Annotation	Annotation	Type	Value	Description
<i>UI.kpi</i>				Annotations belonging to UI.KPI represent a single point of data, specialized for a specific data selection and extended with information about KPI details, especially the first level of drilldown, for a progressive disclosure. Scope: [ELEMENT]
<i>UI.kpi</i>	<i>qualifier</i>	String (120)		The qualifier is optional. In case the qualifier is set, it is used as OData term qualifier. If the qualifier is not set the name of the annotated element is used as OData term qualifier.
<i>UI.kpi</i>	<i>id</i>	String (120)		
<i>UI.kpi</i>	<i>shortDescription</i>	String (120)		Short description for the respective element.
<i>UI.kpi</i>	<i>selectionVariantQualifier</i>	String (120)		This element refers to a UI.selectionVariant annotation at the same view via its qualifier.
<i>UI.kpi</i>	<i>detail</i>	String (120)		This annotation bundles additional settings for a KPI which are relevant for a separate KPI detail display or for progressive disclosure of the KPI.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.kpi</code>	<code>detail.defaultPresentationVariantQualifier</code>	String (120)		This element refers to a UI.presentationVariant annotation at the same view via its qualifier. This presentation variant should be used to initially represent the KPI detail.
<code>UI.kpi</code>	<code>detail.alternativePresentationVariantQualifiers</code>	(array of) String (120)		This element refers to a UI.presentationVariant annotations at the same view via their qualifiers. These presentation variants should be used/offered as alternative representations of the KPI detail
<code>UI.kpi</code>	<code>detail.semanticObject</code>	String (120)		<p>Allows to annotate SAP-specific business semantics, extending the standardized business semantics, covered by the @Semantics domain.</p> <p>Consumers may leverage this enrichment for enhanced interoperability across applications. E.g. Fiori introduced the concept of intent-based navigation, where an intent is a combination <semanticObject>-<action>. A semanticObject annotation will be used in Fiori UIs to dynamically derive navigation targets (for the annotated view as a source).</p>

Parent Annotation	Annotation	Type	Value	Description
<i>UI.kpi</i>	<i>detail.semanticObject</i> <i>Action</i>	String(120)		This annotation refers to the name of an action on the semantic object. The semantic object is taken from @Consumption.semanticObject or derived via an association from the defining view.
<i>UI.kpi</i>	<i>dataPoint</i>			For the <i>UI.kpi.dataPoint</i> annotations you can refer to <i>UI.dataPoint</i> .
<i>UI.selectionField</i>		(array of)		<p>Annotations belonging to <i>UI.selectionField</i> allow filtering a list of data. <i>UI.selectionField</i> annotations are usually used in an initial page floorplan as filter bar.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations</p>
<i>UI.selectionField</i>	<i>qualifier</i>	String(120)		This annotation is used to group and uniquely identify annotations. You need to specify a qualifier as name of a selection field to ensure that the correct selection field can be referenced by the UI.
<i>UI.selectionField</i>	<i>position</i>	DecimalFloat	#mandatory	With this annotation you specify the order of selection fields that are used for filtering.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.selectionField</code>	<code>exclude</code>	Boolean	#optional	This annotation allows excluding the element from the OData annotation on the derived view by setting it to true.
			Default: true	For more information, see Inheritance of Annotations [page 338]
<code>UI.selectionField</code>	<code>element</code>	elementRef	#mandatory: Must be used when an association is annotated, the value is a path to an element of the associated view. You use this option if you want to filter a table for a column that is not defined in your CDS view but in another CDS view.	<p>⚠ Caution</p> <p>Must not be used when a structured element is annotated, in this case the annotated element is the value.</p>
<code>UI.valueCriticality</code>		(array of)		Scope: [ELEMENT]
<code>UI.valueCriticality</code>	<code>qualifier</code>	String(120);		The qualifier is optional. In case the qualifier is set, it is used as OData term qualifier. If the qualifier is not set the name of the annotated element is used as OData term qualifier.
<code>UI.valueCriticality</code>	<code>value</code>	Expression		This annotation refers to a value.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.valueCriticality</code>	<code>criticality</code>	Integer enum	<ul style="list-style-type: none"> • 1: <code>Negative</code> • 2: <code>Critical</code> • 3: <code>Positive</code> 	Reference to an element; valid element values are 1, 2, and 3. The criticality value 'Negative' is reflected by 1, the value 'Critical' by 2, and the value 'Positive' by 3.
<code>UI.criticalityLabels</code>		(array of)		Scope: [ELEMENT]
<code>UI.criticalityLabels</code>	<code>qualifier</code>	String(120);		A set of connected fields is identified by its qualifier and can be referenced in data fields for annotation similar to data points and charts.
<code>UI.criticalityLabels</code>	<code>criticality</code>	Integer enum	<ul style="list-style-type: none"> • 1: <code>Negative</code> • 2: <code>Critical</code> • 3: <code>Positive</code> 	Reference to an element; valid element values are 1, 2, and 3. The criticality value 'Negative' is reflected by 1, the value 'Critical' by 2, and the value 'Positive' by 3.
<code>UI.criticalityLabels</code>	<code>label</code>	String(60);		Optional element containing a language-dependent text; if omitted, the label of the annotated element or the label of the element referenced via value is used

Parent Annotation	Annotation	Type	Value	Description
<i>UI.chart</i>				<p>Annotations belonging to <i>UI.chart</i> are used to show a visual representation of aggregated data.</p> <p>Scope: [ENTITY]</p> <p>Evaluation Runtime (Engine): SADL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: array of</p>
<i>UI.chart</i>	<i>qualifier</i>	String(120)		This annotation is used to group and uniquely identify annotations. You need to specify a qualifier as name of a chart to ensure that the correct chart can be referenced by the UI.
<i>UI.chart</i>	<i>title</i>	String(60)	#optional	This annotation contains a language-dependent text. If omitted, the @EndUser-Text.label of the annotated entity or view is used.
<i>UI.chart</i>	<i>description</i>	String(120)	#optional	This annotation contains a language-dependent text. If omitted, the @EndUser-Text.quickInfo of the annotated entity or view is used.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.chart</code>	<code>chartType</code>	String enum	COLUMN; COLUMN_STACKED; COL- UMN_STACKED_100; COLUMN_DUAL; COL- UMN_STACKED_DUAL; COL- UMN_STACKED_DUAL_ 100; BAR; BAR_STACKED; BAR_STACKED_100; BAR_DUAL; BAR_STACKED_DUAL; BAR_STACKED_DUAL_1 00; AREA; AREA_STACKED; AREA_STACKED_100; HORIZONTAL_AREA; HORIZON- TAL_AREA_STACKED; HORIZON- TAL_AREA_STACKED_1 00; LINE; LINE_DUAL; COMBINATION; COMBINA- TION_STACKED; COMBINA- TION_STACKED_DUAL; HORIZONTAL_COMBI- NATION_STACKED;	This enumeration ele- ment specifies the type of graphical rep- resentation most ap- propriate for the data in the annotated view or entity.

Parent Annotation	Annotation	Type	Value	Description
			HORIZONTAL_COMBI- NA- TION_STACKED_DUAL; PIE; DONUT; SCATTER; BUBBLE; RADAR; HEAT_MAP; TREE_MAP; WATERFALL; BULLET; VERTICAL_BULLET; HORIZONTAL_WATER- FALL; HORIZONTAL_COMBI- NATION_DUAL; DONUT_100;	
<i>UI.chart</i>	<i>dimensions</i>	elementRef		This annotation is an array of one or more element references for the discrete axes of a chart. The exact semantics depend on the chart type.
<i>UI.chart</i>	<i>measures</i>	elementRef		This annotation is an array of zero or more element references for the numeric axes of a chart. The exact semantics depend on the chart type.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.chart</code>	<code>dimensionAttributes.dimension</code>	<code>elementRef</code>		This annotation defines the dimensions used in a chart. This annotation must reference an element that is contained in <code>UI.chart.dimensions</code> .
<code>UI.chart</code>	<code>dimensionAttributes.role</code>	<code>String(10) enum</code>	<p>These annotations determine the visualization of a chart.</p> <ul style="list-style-type: none"> • CATEGORY: For example: Line chart: Dimensions for which the role is set to CATEGORY, make up the X-axis (category axis). If no dimension is specified with this role, the first dimension is used as the X-axis. • SERIES: For example: Line chart: Dimensions for which the role is set to SERIES make up the line segments of the chart, with different colors assigned to each dimension value. If multiple dimensions are assigned to this role, the values of all such dimensions together are considered as one dimension and a color is assigned. • CATEGORY2 	This annotation defines the manner in which a dimension is used within a chart. This is configured differently for each chart type.
<code>UI.chart</code>	<code>dimensionAttributes.vacValuesForSequentialColorLevels</code>	<code>array of String(1024)</code>		

Parent Annotation	Annotation	Type	Value	Description
<code>UI.chart</code>	<code>dimensionAttributes.emphasizedValues</code>	array of String (1024)		
<code>UI.chart</code>	<code>measureAttributes</code>	(array of)		Annotations belonging to <code>UI.chart.measureAttributes</code> are used to specify the measure attributes of a chart.
<code>UI.chart</code>	<code>measureAttributes.measure</code>	ElementRef		This annotation defines the measures used in a chart. This annotation must reference an element that is contained in <code>UI.chart.measures</code> and has a <code>UI.dataPoint</code> annotation.
<code>UI.chart</code>	<code>measureAttributes.role</code>	String (10) enum	This annotation determines the visualization of a chart. <ul style="list-style-type: none">• <code>AXIS_1</code>: Example Bubble chart: The first measure for which the role is set to <code>AXIS_1</code>, or if none exists, the first measure for which the role is set to <code>AXIS_2</code>, or if none exists, the first measure for which the role is set to <code>AXIS_3</code>, is assigned to the feed <code>UID valueAxis</code>. This makes up the X-axis.• <code>AXIS_2</code>: For an example, see the description of <code>AXIS_1</code>.• <code>AXIS_3</code>: For an example, see the description of <code>AXIS_1</code>.	This annotation defines the manner in which a measure is used within a chart. This is configured differently for each chart type.

Parent Annotation	Annotation	Type	Value	Description
<i>UI.chart</i>	<i>measureAttributes.asDataPoint</i>	Boolean	Default : true	This annotation defines whether or not measures are displayed as data points in addition to a chart. The element annotated with this UI annotation needs to have an annotation to a data point.
<i>UI.chart</i>	<i>measureAttributes.useSequentialColorLevels</i>	Boolean	Default : false	
<i>UI.chart</i>	<i>actions</i>			Array of data fields of type FOR_ACTION or FOR_IN- TENT_BASED_NOTIFICATION. The meaning and usage of the elements is described in the general section on UI data fields.
<i>UI.chart</i>	<i>actions.type</i>	String(40) enum	<ul style="list-style-type: none"> • FOR_ACTION • FOR_IN- TENT_BASED_NAVIGATION 	
<i>UI.chart</i>	<i>actions.label</i>	String(60)		
<i>UI.chart</i>	<i>actions.dataAction</i>	String(120)		References the technical name of a BOPF action. String pattern is 'BOPF:<technical name of action in BOPF>'.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.chart</code>	<code>actions.requiresContext</code>	Boolean Default: true		<p>"Buttons used for external navigation are displayed only if the navigation target is supported on the current device. The following prerequisites must be fulfilled to display the buttons correctly:</p> <p>You have maintained the supported navigation target in the SAP Fiori launchpad.</p> <p>The action is independent of the context (selection of a line item) by default. That is, the button will always be displayed and can be clicked by the user. If you want the button to be enabled only when the user has selected one or more line items, you have set the annotation requiresContext to ""true"".</p> <p>The annotations *.requiresContext is only evaluated for data-Fields of type FOR_CONTEXT_BASED_NAVIGATION."</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.chart</code>	<code>actions. invocationGrouping</code>	String(12) enum	<ul style="list-style-type: none"> • ISOLATED: Describes the error handling when an action cannot be executed on all selected instances: The action is executed on all instances except for instance on which the action cannot be executed. • CHANGE_SET: Describes the error handling when an action cannot be executed on all selected instances: If an action cannot be executed on one of the selected instances, the action is executed on none of the selected instances. 	<p>This annotation expresses how multiple invocations of the same action on multiple instances are grouped.</p> <p>ISOLATED Example: A user selects five items in a list and wants to copy them. One item cannot be copied. This item will not be copied, the other four items are copied.</p> <p>CHANGE_SET Example: A user selects five items in a list and wants to copy them. One item cannot be copied. None of the selected items are copied.</p> <p>Default: ISOLATED</p>
<code>UI.chart</code>	<code>actions.semanticObjec tAction</code>	String(120)		This annotation refers to the name of an action on the semantic object. The semantic object is taken from <code>@Consumption.semanticObject</code> or derived via an association from the defining view.

Parent Annotation	Annotation	Type	Value	Description
<i>UI. selectionPresentationVariant</i>				<p>Annotations belonging to UI.selectionPresentationVariant are used to bundle annotations of UI.presentationVariant and UI.selectionVariant.</p> <p>Scope: [ENTITY]</p> <p>Evaluation Runtime (Engine): SADL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: array of</p>
<i>UI. selectionPresentationVariant</i>	<i>qualifier</i>	String (120)		This annotation is used to group and uniquely identify annotations. You need to specify a qualifier as name of a selection presentation variant to ensure that the correct selection presentation variant can be referenced by the UI.
<i>UI. selectionPresentationVariant</i>	<i>id</i>	String (120)		This annotation contains an identifier to reference this instance from an external context.
<i>UI. selectionPresentationVariant</i>	<i>text</i>	String (60)		This annotation contains the language-dependent name of the selection presentation variant.

Parent Annotation	Annotation	Type	Value	Description
<code>UI. selectionPresentationVariant</code>	<code>selectionVariantQualifier</code>	<code>String(120)</code>		This annotation is used to group and uniquely identify annotations. You need to specify a qualifier as name of a selection variant to ensure that the correct selection variant can be referenced by the selection presentation variant
<code>UI. selectionPresentationVariant</code>	<code>presentationVariantQualifier</code>	<code>String(120)</code>		This annotation is used to group and uniquely identify annotations. You need to specify a qualifier as name of a presentation variant to ensure that the correct presentation variant can be referenced by the selection presentation variant.
UI selectionVariant			<p>Annotations belonging to UI.selectionVariant are used to denote a combination of parameters and filters used to query the annotated entity set.</p> <p>Scope: [ENTITY]</p> <p>Evaluation Runtime (Engine): SADL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: array of</p>	

Parent Annotation	Annotation	Type	Value	Description
<code>UI.selectionVariant</code>	<code>qualifier</code>	String (120)		This annotation is used to group and uniquely identify annotations. You need to specify a qualifier as name of a selection variant to ensure that the correct selection variant can be referenced by the UI.
<code>UI.selectionVariant</code>	<code>id:</code>	String (120)		This annotation can contain an identifier to reference this instance from an external context.
<code>UI.selectionVariant</code>	<code>text</code>	String (60)		This annotation contains the language-dependent name of the selection variant.
<code>UI.selectionVariant</code>	<code>parameters</code>	(array of)		Annotations belonging to <code>UI.selectionVariant.parameters</code> represent a collection of parameters used to query the annotated entity set.
<code>UI.selectionVariant</code>	<code>parameters.name</code>	ParameterRef		This annotation references to a parameter name.
<code>UI.selectionVariant</code>	<code>parameters.value</code>	String (1024)		This annotation contains a parameter value.
<code>UI.selectionVariant</code>	<code>filter</code>	String (1024)		This annotation contains a filter used to query the annotated entity set.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.presentationVariant</code>	<code>qualifier</code>	String (120)		This annotation is used to group and uniquely identify annotations. You need to specify a qualifier as name of a presentation variant to ensure that the correct presentation variant can be referenced by the UI.
<code>UI.presentationVariant</code>	<code>id:</code>	String (120)		This annotation contains an identifier to reference this instance from an external context.
<code>UI.presentationVariant</code>	<code>text</code>	String (60)		This annotation contains the language-dependent name of the presentation variant.
<code>UI.presentationVariant</code>	<code>maxItems</code>	Integer		This annotation defines the maximum number of items that should be included in the result.
<code>UI.presentationVariant</code>	<code>sortOrder</code>	(array of)		Annotations belonging to <code>UI.presentationVariant.sortOrder</code> represent a collection of sorting parameters that can be provided inline or by a reference to a <code>Common.SortOrder</code> annotation (syntax is identical to <code>AnnotationPath</code>).
<code>UI.presentationVariant</code>	<code>sortOrder.by</code>	elementRef		This annotation defines by what property queried collections can be sorted.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.presentationVariant</code>	<code>sortOrder.direction</code>	<code>String(4) enum</code>	<ul style="list-style-type: none"> • <code>ASC</code>: Sort in ascending order. • <code>DESC</code>: Sort in descending order. 	This annotation defines the sorting direction of queried collections.
<code>UI.presentationVariant</code>	<code>groupBy</code>	<code>array of elementRef</code>	<p>This annotation defines a sequence of groupable properties (p_1, p_2, p_n) that define how the result of a queried collection is composed of instances that represent groups, one group for each combination of value properties in the queried collection.</p> <p>The sequence specifies a certain level of aggregation for the queried collection, and every group instance provides aggregated values for properties that are aggregatable. Moreover, the series of sub-sequences, for example $(p_1), (p_1, p_2), \dots$, forms a leveled hierarchy that can be used in combination with the annotation <code>UI.presentationVariant.initialExpansionLevel</code>.</p>	

Parent Annotation	Annotation	Type	Value	Description
<code>UI.presentationVariant</code>	<code>totalBy</code>	array of elementRef		This annotation defines the sub-sequence q1, q2, qn of the properties p1, p2, pn specified in the annotation <code>UI.presentationVariant.groupBy</code> . With this, additional levels of aggregation are requested in addition to the most granular level defined by the annotation <code>UI.presentationVariant.groupBy</code> . Every element in the series of sub-sequences, for example (q1), (q1, q2), ..., introduces an additional aggregation level included in the result of the queried collection.
<code>UI.presentationVariant</code>	<code>total</code>	array of elementRef		This annotation contains aggregatable properties for which aggregated values are to be provided for the additional aggregation levels specified in the annotation <code>UI.presentationVariant.totalBy</code> .
<code>UI.presentationVariant</code>	<code>includeGrandTotal</code>	Boolean Default: true		This annotation specifies that the result of the queried collection includes a grand total for the properties specified in the annotation <code>UI.presentationVariant.total</code> .

Parent Annotation	Annotation	Type	Value	Description
<code>UI.presentationVariant</code>	<code>initialExpansionLevel</code>	Integer		This annotation contains the initial number of expansion levels of a hierarchy defined for the queried collection. The hierarchy can be implicitly imposed by the sequence of the annotation <code>UI.presentationVariant.groupBy</code> , or by an explicit hierarchy annotation.
<code>UI.presentationVariant</code>	<code>requestAtLeast</code>	array of elementRef		This annotation defines the properties that should always be included in the result of the queried collection.
<code>UI.presentationVariant</code>	<code>visualizations</code>	(array of)		<p>Annotations belonging to <code>UI.presentationVariant.visualizations</code> represent a collection of available visualization types. The following types are supported:</p> <ul style="list-style-type: none"> <code>UI.lineItem</code> <code>UI.chart</code> <code>UI.dataPoint</code>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.presentationVariant</code>	<code>visualizations.type</code>	<code>String(40)</code> enum	<ul style="list-style-type: none"> • <code>AS_LINEITEM</code>: The queried collection is visualized as line item. • <code>AS_CHART</code>: The queried collection is visualized as chart. • <code>AS_DATAPOINT</code>: The queried collection is visualized as data point. <p>Default: AS_LINEITEM</p>	This annotation defines the representation type. For each type, only one single annotation is meaningful. Multiple instances of the same visualization type shall be modeled with different presentation variants. A reference to the annotation <code>UI.lineItem</code> should always be part of the collection (least common denominator for renderers).
<code>UI.presentationVariant</code>	<code>visualizations.qualifier</code>	<code>String(120)</code>		This annotation is used to group and uniquely identify annotations. You need to specify a qualifier as name of a visualization to ensure that the correct visualization can be referenced by the UI.
<code>UI.presentationVariant</code>	<code>visualizations.element</code>	<code>ElementRef</code>		This annotation references the annotation <code>UI.lineItem</code> .
<code>UI.presentationVariant</code>	<code>selectionFieldsQualifier</code>	<code>String(120)</code>		eElement referring to <code>UI.selectionField</code> annotations at elements of the same view via their qualifier.

Parent Annotation	Annotation	Type	Value	Description
<i>UI.hidden</i>		Boolean		This annotation allows to show or hide data fields based on the state of the data instance. For more information, see Field Hiding.
		Default true		Scope: [ELEMENT, PARAMETER]
				Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations
<i>UI.masked</i>		Boolean		This annotation refers to, for example, passwords or pass phrases. The user interface may offer to show the value in clear text upon explicit user interaction. For more information, see Field Masking.
		Default true		Scope: [ELEMENT]
				Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations

Parent Annotation	Annotation	Type	Value	Description
<i>UI.multiLineText</i>		Boolean		UI.multiLineText
		Default true		This annotation contains text that is rendered as multiple lines. For more information, see Multi-Line Text [page 333] .
				Scope: [ELEMENT] Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations
<i>UI.textArrangement</i>		String (13) enum	<ul style="list-style-type: none"> • <i>TEXT_FIRST</i>: The text is displayed in front of the code. • <i>TEXT_LAST</i>: The code is displayed in front of the text. • <i>TEXT_ONLY</i>: The text is displayed without the code. • <i>TEXT_SEPARATE</i>: TEXT_SEPARATE The text and the code are displayed separately. 	<p>Description: This annotation specifies the arrangement of code-text pairs.</p> <p>Scope: [ENTITY, ELEMENT]</p> <p>Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: String</p>
<i>UI.facet</i>	<i>qualifier</i>	String (120)		This element uniquely identifies alternative values for an annotation; only allowed if parentId is not specified.
<i>UI.facet</i>	<i>feature</i>	String (40)		This annotation specifies the name of a feature toggle that determines visibility of the facet.
<i>UI.facet</i>	<i>id</i>	String (120)		This element is the identifier of the facet

Parent Annotation	Annotation	Type	Value	Description
<code>UI.facet</code>	<code>purpose</code>	String(40) enum	#not compatible with UI.facet.parentID STANDARD; HEADER; QUICK_VIEW; QUICK_CREATE; FILTER; default: STANDARD;	This annotation specifies the purpose of a facet; only allowed if parentId is not specified
			⚠ Caution You can only use this annotation, if parentId is not specified.	This annotation identifies the parent facet. only allowed if purpose is not specified
<code>UI.facet</code>	<code>parentId</code>	String(120)	#not compatible with UI.facet.purpose STANDARD; HEADER; QUICK_VIEW; QUICK_CREATE; FILTER; default: STANDARD;	
			⚠ Caution You can only use this annotation, if purpose is not specified.	This annotation specifies the relative position of the Facet within its parent or term. Must be specified to allow interspersing extension elements via extend views.
<code>UI.facet</code>	<code>position</code>	DecimalFloat		
<code>UI.facet</code>	<code>exclude</code>	Boolean Default: true		Marker to explicitly exclude a CDS element from the collection of DataField for a derived view.

Parent Annotation	Annotation	Type	Value	Description
<i>UI.facet</i>	<i>hidden</i>	Boolean		The annotation <i>hidden</i> allows providing a static Boolean value (for facets hidden by default and e.g. made visible via code exit in the UI), but usually the value will be provided by referencing a Boolean element of the same view using the #(...) syntax. The latter allows to dynamically switch the facet on or off.
<i>UI.facet</i>	<i>isPartOfPreview</i>	Boolean	Default: true	This annotation determines that this facet and all included facets are part of the thing preview.
<i>UI.facet</i>	<i>isSummary</i>	Boolean	Default: true	This annotation specifies that this facet and all included facets are the summary of the thing. At most one facet of a thing can be tagged with this term
<i>UI.facet</i>	<i>isMap</i>	Boolean	Default: true	This annotation specifies that this facet is best represented as a geographic map.
<i>UI.facet</i>	<i>importance</i>	String (6) enum	<ul style="list-style-type: none"> • HIGH • MEDIUM • LOW 	<p>This expresses the importance of dataFields or other annotations. It can be used e.g. in dynamic rendering approaches with responsive design.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.facet</code>	<code>label</code>	String (60)	#optional	This annotation can contain a language-dependent text; if omitted, the label of the annotated element or the label of the element referenced via value which is used.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.facet</code>	<code>type</code>	String(40) enum	<ul style="list-style-type: none"> • COLLECTION: <ul style="list-style-type: none"> • id element is required • no additional elements are available • ADDRESS_REFERENCE <ul style="list-style-type: none"> - targetElement element is available - targetQualifier element is available - isMap element is available • BADGE_REFERENCE <ul style="list-style-type: none"> - targetElement element is available • CHART_REFERENCE <ul style="list-style-type: none"> - targetElement element is available - targetQualifier element is available • CONTACT_REFERENCE <ul style="list-style-type: none"> - targetElement element is available - targetQualifier element is available • DATAPOINT_REFERENCE <ul style="list-style-type: none"> - targetElement element is available - targetQualifier element is required • HEADERINFO_REFERENCE <ul style="list-style-type: none"> - targetElement element is available - targetQualifier element is required • IDENTIFICATION_REFERENCE <ul style="list-style-type: none"> - targetElement element is available • LINEITEM_REFERENCE 	This enumeration element specifies the concrete type of the facet. The enumeration type value determines which CDS annotation elements are required or available

Parent Annotation	Annotation	Type	Value	Description
			<ul style="list-style-type: none"> - targetElement element is available - targetQualifier element is available • STATUSINFO_REFERENCE - targetElement element is available - targetQualifier element is available • URL_REFERENCE - url element is available 	
<i>UI.facet</i>	<i>targetElement</i>	elementRef		This annotation specifies a path to the CDS view whose annotation is referenced. Not specified means the current view. The path is converted to an OData Annotation-Path.
<i>UI.facet</i>	<i>targetQualifier</i>	String (120)		This annotation is the qualifier of the referenced annotation.
<i>UI.facet</i>	<i>url</i>	ElementRef		This annotation specifies a path to structural element containing the navigation URL.

Parent Annotation	Annotation	Type	Value	Description
UI.lineItem				<p>Annotations belonging to UI.lineItem represent an ordered collection of data fields that is used to represent data from multiple data instances in a table or a list. For more information, see Columns.</p> <p>Scope: [ELEMENT, ENTITY]</p> <p>Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations</p>
UI.lineItem	<i>qualifier</i>	String (120)		<p>This annotation is used to group and uniquely identify annotations.</p> <p>If you want to use more than one table, you need a qualifier to distinguish them on the UI.</p>
UI.lineItem	<i>position</i>	DecimalFloat	#mandatory	<p>With this annotation you specify the order of the columns of a list.</p>
UI.lineItem	<i>exclude</i>	Boolean Default: True		<p>This annotation allows excluding the element from the OData annotation on the derived view by setting it to true. The element is optional.</p> <p>For more information, see Inheritance of Annotations [page 338]</p>

Parent Annotation	Annotation	Type	Value	Description
UI.lineItem	<i>hidden</i>	Boolean		The annotation <code>hidden</code> allows providing a static boolean value, but usually the value will be provided by referencing a Boolean element of the same view using the <code>#(...)</code> syntax.
UI.lineItem	<i>importance</i>	String(6) enum		<p>i Note</p> <p>If no importance is defined, the line item is treated like having importance LOW.</p> <ul style="list-style-type: none"> • HIGH • MEDIUM • LOW <p>This annotation expresses the importance of dataFields or other annotations. The element can be used, for example, in dynamic rendering approaches with responsive design patterns.</p> <p>Example:</p> <p>You defined a table with several columns. The columns that need to be displayed always, get importance HIGH. This ensures that these columns are displayed in a table when this table is rendered on a small display.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.lineItem	<code>type</code>	<code>String(40) enum</code>	<code>AS_ADDRESS;</code> <code>AS_CHART;</code> <code>AS_CONNECTED_FIELDS;</code> <code>AS_CONTACT;</code> <code>AS_DATAPOINT;</code> <code>AS_FIELDGROUP;</code> <code>FOR_ACTION;</code> <code>FOR_IN-</code> <code>TENT_BASED_NAVI-</code> <code>GA-</code> <code>TION;</code> <code>STANDARD;</code> <code>WITH_IN-</code> <code>TENT_BASED_NAVI-</code> <code>GA-</code> <code>TION;</code> <code>WITH_NAVI-</code> <code>GATION_PATH;</code> <code>WITH_URL;</code> <code>} default #STANDARD;</code> <code>label</code>	This enumeration element specifies the concrete used type of the data field type hierarchy. The enumeration type value determines which CDS elements are required or available.
UI.lineItem	<code>label</code>	<code>String(60)</code>	<code>#optional</code>	<p>This annotation contains a language-dependent text that can be used for column titles in tables headers.</p> <p>If omitted, the label of the annotated element, or the label of the element referenced via the value is used.</p>
UI.lineItem	<code>iconUrl</code>	<code>String(1024)</code>	<code>#optional</code>	This annotation contains the URL to an icon image.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.lineItem</code>	<code>criticality</code>	<code>ElementRef</code>	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • <code>Neutral</code>: 0 • <code>Negative</code>: 1 • <code>Critical</code>: 2 • <code>Positive</code>: 3 	<p>Reference to an element; valid element values are 1, 2, and 3.</p> <p>The criticality value 'Negative' is reflected by 1, the value 'Critical' by 2, and the value 'Positive' by 3.</p>
<code>UI.lineItem</code>	<code>criticalityRepresentation</code>	<code>String(12)</code> enum	<ul style="list-style-type: none"> • <code>WITHOUT_ICON</code> • <code>WITH_ICON</code> <p>Default:</p>	Defines whether criticality is represented by an icon or not.
<code>UI.lineItem</code>	<code>dataAction</code>	<code>String(120)</code>		<p>This annotation can be used if the line item type is <code>FOR_ACTION</code>. The element references the technical name of an action of the Business Object Processing Framework (BOPF), for example. In this case, the string pattern is <code>BOPF:<technical name of action in BOPF></code>.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.lineItem</code>	<code>requiresContext</code>	Boolean Default: True		<p>Buttons used for external navigation are displayed only if the navigation target is supported on the current device. The following prerequisites must be fulfilled to display the buttons correctly:</p> <p>You have maintained the supported navigation target in the SAP Fiori launchpad.</p> <p>The action is independent of the context (selection of a line item) by default. That is, the button will always be displayed and can be clicked by the user. If you want the button to be enabled only when the user has selected one or more line items, you have set the annotation <code>requiresContext</code> to <code>"true"</code>.</p> <p>The annotations <code>*.requiresContext</code> is only evaluated for data-Fields of type <code>FOR_INTENT_BASED_NAVIGATION</code>.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.lineItem	<i>invocationGrouping</i>	String(12) enum	#optional	<p>i Note</p> <p>This annotation needs to be specified if you use UI.lineItem.type of type FOR_ACTION.</p> <ul style="list-style-type: none"> • <i>ISOLATED</i>: Describes the error handling when an action cannot be executed on all selected instances: The action is executed on all instances except for instance on which the action cannot be executed. • <i>CHANGE_SET</i>: Describes the error handling when an action cannot be executed on all selected instances: If an action cannot be executed on one of the selected instances, the action is executed on none of the selected instances. <p>ISOLATED Example: A user selects five items in a list and wants to copy them. One item cannot be copied. This item will not be copied, the other four items are copied.</p> <p>CHANGE_SET Example: A user selects five items in a list and wants to copy them. One item cannot be copied. None of the selected items are copied.</p> <p>Default: ISOLATED</p>
UI.lineItem	<i>semanticObjectAction</i>	String(120)		This annotation refers to the name of an action on the semantic object. The semantic object is taken from @Consumption.semanticObject or derived via an association from the defining view.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.lineItem</code>	<code>value</code>	<code>ElementRef</code>	<p>For type AS_ADDRESS:</p> <ul style="list-style-type: none"> Value element must not be used when a structural element is annotated. Use instead <code>@com.sap.vocabularies.Communication.v1.Address</code> (or a shorter alias-qualified name) as value. Value element must be used when an element of an associated CDS view is annotated. A value of '.' refers to <code>@Semantics.address</code> on the view that is directly associated. If you want to reference <code>@Semantics.address</code> on a view that is indirectly associated, use a path starting with a dot as value. <p>All other types:</p> <ul style="list-style-type: none"> Value element must not be used when an element is annotated, in this case the annotated element is the value. Value element must be used when an association is annotated. The value is a path to an element of the associated view. 	This annotation refers to a value.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.lineItem</code>	<code>valueQualifier</code>	<code>String(120)</code>		Qualifier of the referenced @UI.chart annotation, only allowed for type AS_CHART.
<code>UI.lineItem</code>	<code>targetElement</code>	<code>ElementRef</code>		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData NavigationPropertyPath. Using this annotation, you can link from the header part of an object view floorplan to a target element. You need to specify <code>UI.lineItem.targetElement</code> when you use the annotation <code>UI.lineItem.type</code> of type WITH_NAVIGATION_PATH.</p> <p>You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.lineItem</code>	<code>url</code>	<code>ElementRef</code>	<p>⚠ Caution</p> <p>You need to specify <code>UI.lineItem.url</code> when you use the annotation <code>UI.lineItem.type</code> of type WITH_URL</p>	<p>This annotation represents the path to a structural element that contains a navigation URL.</p>

Parent Annotation	Annotation	Type	Value	Description
<i>UI.identification</i>				<p>Annotation belonging to <i>UI.identification</i> represent an ordered collection of specific data fields that together with <i>headerInfo</i> identifies an entity to an end user.</p> <p>Example</p> <p>This annotation is displayed in the General Information section in the body of the object view floorplan of an item, for example.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: array of</p>
<i>UI.identification</i>	<i>qualifier</i>	String(120)		<p>This annotation is used to group and uniquely identify annotations.</p> <p>If you want to use more than one table, you need a qualifier to distinguish them on the UI.</p>
<i>UI.identification</i>	<i>position</i>	DecimalFloat	#mandatory	With this annotation you specify the order of the columns of a list.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.identification</code>	<code>exclude</code>	Boolean Default: True		<p>This annotation allows excluding the element from the OData annotation on the derived view by setting it to <code>true</code>. The element is optional.</p> <p>For more information, see Inheritance of Annotations [page 338]</p>
<code>UI.identification</code>	<code>hidden</code>	Boolean Default: True		<p>The annotation <code>hidden</code> allows providing a static boolean value, but usually the value will be provided by referencing a Boolean element of the same view using the <code>#(...)</code> syntax.</p>
<code>UI.identification</code>	<code>importance</code>	String(6) enum	<p>i Note</p> <p>If no importance is defined, the line item is treated like having importance <code>LOW</code>.</p> <ul style="list-style-type: none"> • <code>HIGH</code> • <code>MEDIUM</code> • <code>LOW</code> 	<p>This annotation expresses the importance of dataFields or other annotations. The element can be used, for example, in dynamic rendering approaches with responsive design patterns.</p> <p>Example:</p> <p>You defined a table with several columns. The columns that need to be displayed always, get importance <code>HIGH</code>. This ensures that these columns are displayed in a table when this table is rendered on a small display.</p>

Parent Annotation	Annotation	Type	Value	Description
<i>UI.identification</i>	<i>type</i>	String(40) enum	AS_ADDRESS; AS_CHART; AS_CONNECTED_FIELDS; AS_CONTACT; AS_DATAPOINT; AS_FIELDGROUP; FOR_ACTION; FOR_IN- TENT_BASED_NAVI- GATION; STANDARD; WITH_IN- TENT_BASED_NAVI- GATION; WITH_NAVI- GATION_PATH; WITH_URL; } default #STANDARD; label	This enumeration element specifies the concrete used type of the data field type hierarchy. The enumeration type value determines which CDS elements are required or available.
<i>UI.identification</i>	<i>label</i>	String(60)	#optional	This annotation contains a language-dependent text that can be used for column titles in tables headers. If omitted, the label of the annotated element, or the label of the element referenced via the value is used.
<i>UI.identification</i>	<i>iconUrl</i>	String(1024)	#optional	This annotation contains the URL to an icon image.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.identification</code>	<code>criticality</code>	<code>ElementRef</code>	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • <code>Neutral</code>: 0 • <code>Negative</code>: 1 • <code>Critical</code>: 2 • <code>Positive</code>: 3 	<p>Reference to an element; valid element values are 1, 2, and 3.</p> <p>The criticality value 'Negative' is reflected by 1, the value 'Critical' by 2, and the value 'Positive' by 3.</p>
<code>UI.identification</code>	<code>criticalityRepresentation</code>	<code>String(12)</code> enum	<ul style="list-style-type: none"> • <code>WITHOUT_ICON</code> • <code>WITH_ICON</code> <p>Default:</p>	Defines whether criticality is represented by an icon or not.
<code>UI.identification</code>	<code>dataAction</code>	<code>String(120)</code>		<p>This annotation can be used if the line item type is <code>FOR_ACTION</code>. The element references the technical name of an action of the Business Object Processing Framework (BOPF), for example. In this case, the string pattern is <code>BOPF:<technical name of action in BOPF></code>.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.identification</code>	<code>requiresContext</code>	Boolean Default: True		<p>Buttons used for external navigation are displayed only if the navigation target is supported on the current device. The following prerequisites must be fulfilled to display the buttons correctly:</p> <p>You have maintained the supported navigation target in the SAP Fiori launchpad.</p> <p>The action is independent of the context (selection of a line item) by default. That is, the button will always be displayed and can be clicked by the user. If you want the button to be enabled only when the user has selected one or more line items, you have set the annotation <code>requiresContext</code> to <code>""true""</code>.</p> <p>The annotations <code>*.requiresContext</code> is only evaluated for data-Fields of type <code>FOR_CONTEXT_BASED_NAVIGATION</code>.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.identification</code>	<code>invocationGrouping</code>	String(12) enum	#optional	<p>i Note</p> <p>This annotation needs to be specified if you use <code>UI.lineItem.type</code> of type <code>FOR_ACTION</code>.</p> <ul style="list-style-type: none"> • <code>ISOLATED</code>: Describes the error handling when an action cannot be executed on all selected instances: The action is executed on all instances except for instance on which the action cannot be executed. • <code>CHANGE_SET</code>: Describes the error handling when an action cannot be executed on all selected instances: If an action cannot be executed on one of the selected instances, the action is executed on none of the selected instances. <p>ISOLATED Example: A user selects five items in a list and wants to copy them. One item cannot be copied. This item will not be copied, the other four items are copied.</p> <p>CHANGE_SET Example: A user selects five items in a list and wants to copy them. One item cannot be copied. None of the selected items are copied.</p> <p>Default: ISOLATED</p>
<code>UI.identification</code>	<code>semanticObjectAction</code>	String(120)		This annotation refers to the name of an action on the semantic object. The semantic object is taken from <code>@Consumption.semanticObject</code> or derived via an association from the defining view.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.identification</code>	<code>value</code>	<code>ElementRef</code>	<p>For type AS_ADDRESS:</p> <ul style="list-style-type: none"> Value element must not be used when a structural element is annotated. Use instead <code>@com.sap.vocabularies.Communication.v1.Address</code> (or a shorter alias-qualified name) as value. Value element must be used when an element of an associated CDS view is annotated. A value of '.' refers to <code>@Semantics.address</code> on the view that is directly associated. If you want to reference <code>@Semantics.address</code> on a view that is indirectly associated, use a path starting with a dot as value. <p>All other types:</p> <ul style="list-style-type: none"> Value element must not be used when an element is annotated, in this case the annotated element is the value. Value element must be used when an association is annotated. The value is a path to an element of the associated view. 	This annotation refers to a value.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.identification</code>	<code>valueQualifier</code>	<code>String(120)</code>		Qualifier of the referenced @UI.chart annotation, only allowed for type AS_CHART.
<code>UI.identification</code>	<code>targetElement</code>	<code>ElementRef</code>		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData NavigationPropertyPath. Using this annotation, you can link from the header part of an object view floorplan to a target element. You need to specify UI.lineItem.targetElement when you use the annotation UI.lineItem.type of type WITH_NAVIGATION_PATH.</p> <p>You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.identification</code>	<code>url</code>	<code>ElementRef</code>		<p>⚠ Caution</p> <p>You need to specify UI.identification.url when you use the annotation UI.identification.type of type WITH_URL.</p> <p>This annotation represents the path to a structural element that contains a navigation URL.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.statusInfo				<p>Annotations belonging to UI.statusInfo represent a list of abstract data fields that convey the status of an entity. UI.statusInfo annotations are usually used in the header section of an item's object view floorplan.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: array of</p>
UI.statusInfo	<i>qualifier</i>	String(120)		<p>This annotation is used to group and uniquely identify annotations.</p> <p>If you want to use more than one table, you need a qualifier to distinguish them on the UI.</p>
UI.statusInfo	<i>position</i>	DecimalFloat	#mandatory	<p>With this annotation you specify the order of the columns of a list.</p>
UI.statusInfo	<i>exclude</i>	Boolean Default: True		<p>This annotation allows excluding the element from the OData annotation on the derived view by setting it to true. The element is optional.</p> <p>For more information, see Inheritance of Annotations [page 338]</p>

Parent Annotation	Annotation	Type	Value	Description
UI.statusInfo	<i>hidden</i>	Boolean		The annotation <code>hidden</code> allows providing a static boolean value, but usually the value will be provided by referencing a Boolean element of the same view using the <code>#(...)</code> syntax.
UI.statusInfo	<i>importance</i>	String(6) enum		<p>i Note</p> <p>If no importance is defined, the line item is treated like having importance LOW.</p> <ul style="list-style-type: none"> • HIGH • MEDIUM • LOW <p>This annotation expresses the importance of dataFields or other annotations. The element can be used, for example, in dynamic rendering approaches with responsive design patterns.</p> <p>Example:</p> <p>You defined a table with several columns. The columns that need to be displayed always, get importance HIGH. This ensures that these columns are displayed in a table when this table is rendered on a small display.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.statusInfo	<code>type</code>	<code>String(40) enum</code>	<code>AS_ADDRESS;</code> <code>AS_CHART;</code> <code>AS_CONNECTED_FIELDS;</code> <code>AS_CONTACT;</code> <code>AS_DATAPOINT;</code> <code>AS_FIELDGROUP;</code> <code>FOR_ACTION;</code> <code>FOR_IN-</code> <code>TENT_BASED_NAVI-</code> <code>GA-</code> <code>TION;</code> <code>STANDARD;</code> <code>WITH_IN-</code> <code>TENT_BASED_NAVI-</code> <code>GA-</code> <code>TION;</code> <code>WITH_NAVI-</code> <code>GATION_PATH;</code> <code>WITH_URL;</code> <code>} default #STANDARD;</code> <code>label</code>	This enumeration element specifies the concrete used type of the data field type hierarchy. The enumeration type value determines which CDS elements are required or available
UI.statusInfo	<code>label</code>	<code>String(60)</code>	<code>#optional</code>	<p>This annotation contains a language-dependent text that can be used for column titles in tables headers.</p> <p>If omitted, the label of the annotated element, or the label of the element referenced via the value is used.</p>
UI.statusInfo	<code>iconUrl</code>	<code>String(1024)</code>	<code>#optional</code>	This annotation contains the URL to an icon image.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.statusInfo</code>	<code>criticality</code>	<code>ElementRef</code>	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • <code>Neutral</code>: 0 • <code>Negative</code>: 1 • <code>Critical</code>: 2 • <code>Positive</code>: 3 	<p>Reference to an element; valid element values are 1, 2, and 3.</p> <p>The criticality value 'Negative' is reflected by 1, the value 'Critical' by 2, and the value 'Positive' by 3.</p>
<code>UI.statusInfo</code>	<code>criticalityRepresentation</code>	<code>String(12)</code> enum	<ul style="list-style-type: none"> • <code>WITHOUT_ICON</code> • <code>WITH_ICON</code> <p>Default:</p>	Defines whether criticality is represented by an icon or not.
<code>UI.statusInfo</code>	<code>dataAction</code>	<code>String(120)</code>		<p>This annotation can be used if the line item type is <code>FOR_ACTION</code>. The element references the technical name of an action of the Business Object Processing Framework (BOPF), for example. In this case, the string pattern is <code>BOPF:<technical name of action in BOPF></code>.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.statusInfo	<code>requiresContext</code>	Boolean Default: True		<p>Buttons used for external navigation are displayed only if the navigation target is supported on the current device. The following prerequisites must be fulfilled to display the buttons correctly:</p> <p>You have maintained the supported navigation target in the SAP Fiori launchpad.</p> <p>The action is independent of the context (selection of a line item) by default. That is, the button will always be displayed and can be clicked by the user. If you want the button to be enabled only when the user has selected one or more line items, you have set the annotation <code>requiresContext</code> to <code>""true""</code>.</p> <p>The annotations <code>*.requiresContext</code> is only evaluated for data-Fields of type FOR_CONTEXT_BASED_NAVIGATION.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.statusInfo	<i>invocationGrouping</i>	String(12) enum	#optional	<p>i Note</p> <p>This annotation needs to be specified if you use UI.lineItem.type of type FOR_ACTION.</p> <ul style="list-style-type: none"> • <i>ISOLATED</i>: Describes the error handling when an action cannot be executed on all selected instances: The action is executed on all instances except for instance on which the action cannot be executed. • <i>CHANGE_SET</i>: Describes the error handling when an action cannot be executed on all selected instances: If an action cannot be executed on one of the selected instances, the action is executed on none of the selected instances. <p>ISOLATED Example: A user selects five items in a list and wants to copy them. One item cannot be copied. This item will not be copied, the other four items are copied.</p> <p>CHANGE_SET Example: A user selects five items in a list and wants to copy them. One item cannot be copied. None of the selected items are copied.</p> <p>Default: ISOLATED</p>
UI.statusInfo	<i>semanticObjectAction</i>	String(120)		This annotation refers to the name of an action on the semantic object. The semantic object is taken from @Consumption.semanticObject or derived via an association from the defining view.

Parent Annotation	Annotation	Type	Value	Description
UI.statusInfo	<code>value</code>	ElementRef	<p>For type AS_ADDRESS:</p> <ul style="list-style-type: none"> Value element must not be used when a structural element is annotated. Use instead <code>@com.sap.vocabularies.Communication.v1.Address</code> (or a shorter alias-qualified name) as value. Value element must be used when an element of an associated CDS view is annotated. A value of '.' refers to <code>@Semantics.address</code> on the view that is directly associated. If you want to reference <code>@Semantics.address</code> on a view that is indirectly associated, use a path starting with a dot as value. <p>All other types:</p> <ul style="list-style-type: none"> Value element must not be used when an element is annotated, in this case the annotated element is the value. Value element must be used when an association is annotated. The value is a path to an element of the associated view. 	This annotation refers to a value.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.statusInfo</code>	<code>valueQualifier</code>	<code>String(120)</code>		Qualifier of the referenced @UI.chart annotation, only allowed for type AS_CHART.
<code>UI.statusInfo</code>	<code>targetElement</code>	<code>ElementRef</code>		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData NavigationPropertyPath. Using this annotation, you can link from the header part of an object view floorplan to a target element. You need to specify UI.lineItem.targetElement when you use the annotation UI.lineItem.type of type WITH_NAVIGATION_PATH.</p> <p>You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.statusInfo</code>	<code>url</code>	<code>ElementRef</code>		<p>⚠ Caution</p> <p>You need to specify UI.statusInfo.url when you use the annotation UI.statusInfo.type of type WITH_URL..</p> <p>This annotation represents the path to a structural element that contains a navigation URL.</p>

Parent Annotation	Annotation	Type	Value	Description
<i>UI.fieldGroup</i>				<p>Annotations belonging to <code>UI.fieldGroup</code> is an ordered collection of data fields with a label for the group.</p> <p><code>UI.fieldGroup</code> annotations are used to represent parts of a single data instance in a form.</p> <p>Scope: [ELEMENT]</p> <p>Evaluation Runtime (Engine): SSDL: Translates CDS annotations into the corresponding OData annotations</p> <p>Values: array of</p>
<i>UI.fieldGroup</i>	<i>qualifier</i>	String(120)		<p>This annotation is used to group and uniquely identify annotations.</p> <p>If you want to use more than one table, you need a qualifier to distinguish them on the UI.</p>
<i>UI.fieldGroup</i>	<i>groupLabel</i>		#optional	<p>This annotation contains language-dependent text that is used as label for the field group. The first occurrence for a given qualifier wins. Other occurrences for the same qualifier are redundant.</p>
<i>UI.fieldGroup</i>	<i>position</i>	DecimalFloat	#mandatory	<p>With this annotation you specify the order of the columns of a list.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.fieldGroup</code>	<code>exclude</code>	Boolean Default: True		<p>This annotation allows excluding the element from the OData annotation on the derived view by setting it to <code>true</code>. The element is optional.</p> <p>For more information, see Inheritance of Annotations [page 338]</p>
<code>UI.fieldGroup</code>	<code>hidden</code>	Boolean Default: True		<p>The annotation <code>hidden</code> allows providing a static boolean value, but usually the value will be provided by referencing a Boolean element of the same view using the <code>#(...)</code> syntax.</p>
<code>UI.fieldGroup</code>	<code>importance</code>	String (6) enum	<p>i Note</p> <p>If no importance is defined, the line item is treated like having importance <code>LOW</code>.</p> <ul style="list-style-type: none"> • <code>HIGH</code> • <code>MEDIUM</code> • <code>LOW</code> 	<p>This annotation expresses the importance of dataFields or other annotations. The element can be used, for example, in dynamic rendering approaches with responsive design patterns.</p> <p>Example:</p> <p>You defined a table with several columns. The columns that need to be displayed always, get importance <code>HIGH</code>. This ensures that these columns are displayed in a table when this table is rendered on a small display.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.fieldGroup</code>	<code>type</code>	<code>String(40) enum</code>	<code>AS_ADDRESS;</code> <code>AS_CHART;</code> <code>AS_CONNECTED_FIELDS;</code> <code>AS_CONTACT;</code> <code>AS_DATAPOINT;</code> <code>AS_FIELDGROUP;</code> <code>FOR_ACTION;</code> <code>FOR_IN-</code> <code>TENT_BASED_NAVI-</code> <code>GA-</code> <code>TION;</code> <code>STANDARD;</code> <code>WITH_IN-</code> <code>TENT_BASED_NAVI-</code> <code>GA-</code> <code>TION;</code> <code>WITH_NAVI-</code> <code>GA-</code> <code>TION_PATH;</code> <code>WITH_URL;</code> <code>} default #STANDARD;</code> <code>label</code>	This enumeration element specifies the concrete used type of the data field type hierarchy. The enumeration type value determines which CDS elements are required or available
<code>UI.fieldGroup</code>	<code>label</code>	<code>String(60)</code>	<code>#optional</code>	<p>This annotation contains a language-dependent text that can be used for column titles in tables headers.</p> <p>If omitted, the label of the annotated element, or the label of the element referenced via the value is used.</p>
<code>UI.fieldGroup</code>	<code>iconUrl</code>	<code>String(1024)</code>	<code>#optional</code>	This annotation contains the URL to an icon image.

Parent Annotation	Annotation	Type	Value	Description
UI.fieldGroup	criticality	ElementRef	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • Neutral: 0 • Negative: 1 • Critical: 2 • Positive: 3 	<p>Reference to an element; valid element values are 1, 2, and 3.</p> <p>The criticality value 'Negative' is reflected by 1, the value 'Critical' by 2, and the value 'Positive' by 3.</p>
UI.fieldGroup	criticalityRepresentation	String (12) enum	<ul style="list-style-type: none"> • WITHOUT_ICON • WITH_ICON <p>Default:</p>	Defines whether criticality is represented by an icon or not.
UI.fieldGroup	dataAction	String (120)		<p>This annotation can be used if the line item type is FOR_ACTION. The element references the technical name of an action of the Business Object Processing Framework (BOPF), for example. In this case, the string pattern is BOPF:<technical name of action in BOPF>.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.fieldGroup</code>	<code>requiresContext</code>	Boolean Default: True		<p>Buttons used for external navigation are displayed only if the navigation target is supported on the current device. The following prerequisites must be fulfilled to display the buttons correctly:</p> <p>You have maintained the supported navigation target in the SAP Fiori launchpad.</p> <p>The action is independent of the context (selection of a line item) by default. That is, the button will always be displayed and can be clicked by the user. If you want the button to be enabled only when the user has selected one or more line items, you have set the annotation <code>requiresContext</code> to <code>""true""</code>.</p> <p>The annotations <code>*.requiresContext</code> is only evaluated for data-Fields of type <code>FOR_INTENT_BASED_NAVIGATION</code>.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.fieldGroup	invocationGrouping	String(12) enum	#optional	<p>i Note</p> <p>This annotation needs to be specified if you use UI.lineItem.type of type FOR_ACTION.</p> <ul style="list-style-type: none"> • ISOLATED: Describes the error handling when an action cannot be executed on all selected instances: The action is executed on all instances except for instance on which the action cannot be executed. • CHANGE_SET: Describes the error handling when an action cannot be executed on all selected instances: If an action cannot be executed on one of the selected instances, the action is executed on none of the selected instances. <p>ISOLATED Example: A user selects five items in a list and wants to copy them. One item cannot be copied. This item will not be copied, the other four items are copied.</p> <p>CHANGE_SET Example: A user selects five items in a list and wants to copy them. One item cannot be copied. None of the selected items are copied.</p> <p>Default: ISOLATED</p>
UI.fieldGroup	semanticObjectAction	String(120)		This annotation refers to the name of an action on the semantic object. The semantic object is taken from @Consumption.semanticObject or derived via an association from the defining view.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.fieldGroup</code>	<code>value</code>	<code>ElementRef</code>	<p>For type AS_ADDRESS:</p> <ul style="list-style-type: none"> Value element must not be used when a structural element is annotated. Use instead <code>@com.sap.vocabularies.Communication.v1.Address</code> (or a shorter alias-qualified name) as value. Value element must be used when an element of an associated CDS view is annotated. A value of '.' refers to <code>@Semantics.address</code> on the view that is directly associated. If you want to reference <code>@Semantics.address</code> on a view that is indirectly associated, use a path starting with a dot as value. <p>All other types:</p> <ul style="list-style-type: none"> Value element must not be used when an element is annotated, in this case the annotated element is the value. Value element must be used when an association is annotated. The value is a path to an element of the associated view. 	This annotation refers to a value.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.fieldGroup</code>	<code>valueQualifier</code>	<code>String(120)</code>		Qualifier of the referenced @UI.chart annotation, only allowed for type AS_CHART.
<code>UI.fieldGroup</code>	<code>targetElement</code>	<code>ElementRef</code>		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData NavigationPropertyPath. Using this annotation, you can link from the header part of an object view floorplan to a target element. You need to specify UI.lineItem.targetElement when you use the annotation UI.lineItem.type of type WITH_NAVIGATION_PATH.</p> <p>You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.fieldGroup</code>	<code>url</code>	<code>ElementRef</code>		<p>⚠ Caution</p> <p>You need to specify UI.fieldGroup.url when you use the annotation UI.fieldGroup.type of type WITH_URL..</p> <p>This annotation represents the path to a structural element that contains a navigation URL.</p>

Parent Annotation	Annotation	Type	Value	Description
UI.connectedFields				<p>Connected fields have a read-only representation that resembles a single field value, only on edit they are visually disassembled into several input controls.</p> <p>Connected fields can appear in forms and tables. A set of connected fields is identified by its qualifier and can be referenced in data fields for annotation similar to data points and charts.</p> <p>Its language-dependent template specifies how to render the connected fields. The template consists of placeholders for values (must be valid OData simple identifiers - think: variable names) enclosed in curly braces, plus printable characters and white-space outside of the curly braces. The example template {rate}%({amount}) consists of two placeholders rate and amount. The value to be inserted for rate will be followed by a percent sign, the value to be inserted for amount will be enclosed in parentheses. Note that the ""values"" to be inserted are themselves data fields and thus can</p>

Parent Annotation	Annotation	Type	Value	Description
				have advanced formatting, here a currency amount with currency for amount. Other possibilities include micro-charts and data points, making this rather powerful.
<i>UI.connectedFields</i>	<i>qualifier</i>	String (120)		the first occurrence of groupLabel for a given qualifier wins, same for template. Other occurrences for the same qualifier are redundant and should be avoided.
<i>UI.connectedFields</i>	<i>groupLabel</i>	#optional		<p>This annotation is used to group and uniquely identify annotations.</p> <p>If you want to use more than one table, you need a qualifier to distinguish them on the UI.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.connectedFields</code>	<code>template</code>	String (255)		The language-dependent template specifies how to render the connected fields. The template consists of placeholders for values (must be valid OData simple identifiers - think: variable names) enclosed in curly braces, plus printable characters and white-space outside of the curly braces. The example template {rate}%({amount}) consists of two placeholders rate and amount. The value to be inserted for rate will be followed by a percent sign, the value to be inserted for amount will be enclosed in parentheses. Note that the "values" to be inserted are themselves data fields and thus can have advanced formatting, here a currency amount with currency for amount.
<code>UI.connectedFields</code>	<code>name</code>	String (120)		
<code>UI.connectedFields</code>	<code>exclude</code>	Boolean Default: True		This annotation allows excluding the element from the OData annotation on the derived view by setting it to true. The element is optional. For more information, see Inheritance of Annotations [page 338]

Parent Annotation	Annotation	Type	Value	Description
UI.connectedFields	<code>hidden</code>	Boolean		The annotation <code>hidden</code> allows providing a static boolean value, but usually the value will be provided by referencing a Boolean element of the same view using the <code>#(...)</code> syntax.
UI.connectedFields	<code>importance</code>	String(6) enum	<p>i Note</p> <p>If no importance is defined, the line item is treated like having importance LOW.</p> <ul style="list-style-type: none"> • HIGH • MEDIUM • LOW 	<p>This annotation expresses the importance of dataFields or other annotations. The element can be used, for example, in dynamic rendering approaches with responsive design patterns.</p> <p>Example:</p> <p>You defined a table with several columns. The columns that need to be displayed always, get importance HIGH. This ensures that these columns are displayed in a table when this table is rendered on a small display.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.connectedFields</code>	<code>type</code>	<code>String(40) enum</code>	AS_ADDRESS; AS_CHART; AS_CONNECTED_FIELDS; AS_CONTACT; AS_DATAPOINT; AS_FIELDGROUP; FOR_ACTION; FOR_IN- TENT_BASED_NAVI- GATION; STANDARD; WITH_IN- TENT_BASED_NAVI- GATION; WITH_NAVI- GATION_PATH; WITH_URL; } default #STANDARD; label	This enumeration element specifies the concrete used type of the data field type hierarchy. The enumeration type value determines which CDS elements are required or available
<code>UI.connectedFields</code>	<code>label</code>	<code>String(60)</code>	#optional	This annotation contains a language-dependent text that can be used for column titles in tables headers. If omitted, the label of the annotated element, or the label of the element referenced via the value is used.
<code>UI.connectedFields</code>	<code>iconUrl</code>	<code>String(1024)</code>	#optional	This annotation contains the URL to an icon image.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.connectedFields</code>	<code>criticality</code>	<code>ElementRef</code>	<p>This annotation references to another element that has the values 0, 1, 2, or 3.</p> <ul style="list-style-type: none"> • <code>Neutral</code>: 0 • <code>Negative</code>: 1 • <code>Critical</code>: 2 • <code>Positive</code>: 3 	<p>Reference to an element; valid element values are 1, 2, and 3.</p> <p>The criticality value 'Negative' is reflected by 1, the value 'Critical' by 2, and the value 'Positive' by 3.</p>
<code>UI.connectedFields</code>	<code>criticalityRepresentation</code>	<code>String(12)</code> enum	<ul style="list-style-type: none"> • <code>WITHOUT_ICON</code> • <code>WITH_ICON</code> <p>Default:</p>	Defines whether criticality is represented by an icon or not.
<code>UI.connectedFields</code>	<code>dataAction</code>	<code>String(120)</code>		<p>This annotation can be used if the line item type is <code>FOR_ACTION</code>. The element references the technical name of an action of the Business Object Processing Framework (BOPF), for example. In this case, the string pattern is <code>BOPF:<technical name of action in BOPF></code>.</p>

Parent Annotation	Annotation	Type	Value	Description
<code>UI.connectedFields</code>	<code>requiresContext</code>	Boolean Default: True	"Buttons used for external navigation are displayed only if the navigation target is supported on the current device. The following prerequisites must be fulfilled to display the buttons correctly: You have maintained the supported navigation target in the SAP Fiori launchpad. The action is independent of the context (selection of a line item) by default. That is, the button will always be displayed and can be clicked by the user. If you want the button to be enabled only when the user has selected one or more line items, you have set the annotation requiresContext to ""true"". The annotations *.requiresContext is only evaluated for data-Fields of type FOR_IN-TENT_BASED_NAVI-GATION."	

Parent Annotation	Annotation	Type	Value	Description
<code>UI.connectedFields</code>	<code>invocationGrouping</code>	String(12) enum	#optional	<p>i Note</p> <p>This annotation needs to be specified if you use <code>UI.lineItem.type</code> of type <code>FOR_ACTION</code>.</p> <ul style="list-style-type: none"> • <code>ISOLATED</code>: Describes the error handling when an action cannot be executed on all selected instances: The action is executed on all instances except for instance on which the action cannot be executed. • <code>CHANGE_SET</code>: Describes the error handling when an action cannot be executed on all selected instances: If an action cannot be executed on one of the selected instances, the action is executed on none of the selected instances. <p>ISOLATED Example: A user selects five items in a list and wants to copy them. One item cannot be copied. This item will not be copied, the other four items are copied.</p> <p>CHANGE_SET Example: A user selects five items in a list and wants to copy them. One item cannot be copied. None of the selected items are copied.</p> <p>Default: ISOLATED</p>
<code>UI.connectedFields</code>	<code>semanticObjectAction</code>	String(120)		This annotation refers to the name of an action on the semantic object. The semantic object is taken from <code>@Consumption.semanticObject</code> or derived via an association from the defining view.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.connectedFields</code>	<code>value</code>	<code>ElementRef</code>	<p>For type AS_ADDRESS:</p> <ul style="list-style-type: none"> Value element must not be used when a structural element is annotated. Use instead <code>@com.sap.vocabularies.Communication.v1.Address</code> (or a shorter alias-qualified name) as value. Value element must be used when an element of an associated CDS view is annotated. A value of '.' refers to <code>@Semantics.address</code> on the view that is directly associated. If you want to reference <code>@Semantics.address</code> on a view that is indirectly associated, use a path starting with a dot as value. <p>All other types:</p> <ul style="list-style-type: none"> Value element must not be used when an element is annotated, in this case the annotated element is the value. Value element must be used when an association is annotated. The value is a path to an element of the associated view. 	This annotation refers to a value.

Parent Annotation	Annotation	Type	Value	Description
<code>UI.connectedFields</code>	<code>valueQualifier</code>	<code>String(120)</code>		Qualifier of the referenced @UI.chart annotation, only allowed for type AS_CHART.
<code>UI.connectedFields</code>	<code>targetElement</code>	<code>ElementRef</code>		<p>This annotation represents the path to an element of an associated CDS view. The path is converted to an OData NavigationPropertyPath. Using this annotation, you can link from the header part of an object view floorplan to a target element. You need to specify UI.lineItem.targetElement when you use the annotation UI.lineItem.type of type WITH_NAVIGATION_PATH.</p> <p>You might, for example, provide background information to an item that is opened on the object view floorplan.</p>
<code>UI.connectedFields</code>	<code>url</code>	<code>ElementRef</code>		<p>⚠ Caution</p> <p>You need to specify UI.fieldGroup.url when you use the annotation UI.connectedFields..type of type WITH_URL..</p> <p>This annotation represents the path to a structural element that contains a navigation URL.</p>

Example

9.1.13 VDM Annotations

Allow classifying views of the virtual data model in terms of their admissible reuse options and provisioned content

Scope and Definition

```
@CompatibilityContract:{ c1.usageAllowed: false,
                        c2.usageAllowed: false }
annotation VDM
{
  @Scope:[#VIEW]
  @CompatibilityContract:{ c1: { usageAllowed: false },
                           c2: { usageAllowed: false }
                         }
  auxiliaryEntity: { for: { entity: EntityRef; },
                     usage: { type: array of String(30) enum
{ENTERPRISE_SEARCH}; } };
  @CompatibilityContract:{ c1: { usageAllowed: true,
                                allowedChanges: { annotation: [#ANY],
                                                  value: [#ANY] }
                               },
                           c2: { usageAllowed: true,
                                 allowedChanges: { annotation: [#ANY],
                                                  value: [#ANY] }
                               }
                         }
  @Scope:[#ENTITY]
  viewType : String(30) enum { BASIC; COMPOSITE; CONSUMPTION; EXTENSION;
DERIVATION_FUNCTION; TRANSACTIONAL; };
  @Scope:[#VIEW, #TABLE_FUNCTION]
  private : Boolean default true;
  @Scope:[#EXTEND_VIEW]
  viewExtension : Boolean default true;
  @CompatibilityContract:{ c1: { usageAllowed: true,
                                allowedChanges: { annotation: [#ANY],
                                                  value: [#ANY] }
                               },
                           c2: { usageAllowed: false,
                                 allowedChanges: { annotation: [#ANY],
                                                  value: [#ANY] }
                               }
                         }
  lifecycle : {
    @CompatibilityContract:{ c2: { usageAllowed: true,
                                  allowedChanges: { annotation:
[#ANY], value: [#ANY] }
                               }
                            }
  }
  @Scope:[#ENTITY]
  contract : { type : String(30) enum { PUBLIC_REMOTE_API;
                                         PUBLIC_LOCAL_API;
                                         SAP_INTERNAL_API;
                                         NONE; } };
}
```

```

    @Scope:[#ENTITY, #ELEMENT]
    status : String(30) enum { DEPRECATED; };
    @Scope:[#ENTITY, #ELEMENT]
    successor : String(30);
}
@CompatibilityContract:{ c1: { usageAllowed: false
},
c2: { usageAllowed: true,
    allowedChanges: { annotation: [#ANY],
                    value: [#ANY] }
}
}
@Scope:[#ENTITY]
usage : {
    type: array of String(40) enum { ACTION_PARAMETER_STRUCTURE;
                                    ACTION_RESULT_STRUCTURE;
                                    EVENT_SIGNATURE;
TRANSACTIONAL_PROCESSING_SERVICE; };
}
;

```

Usage

VDM is intended to be interpreted by view browsers and other functionality which is based on the virtual data model.

This classification is used only for SAP internal structuring and interpretation of the CDS entities. Releasing CDS entities for customers and partners is controlled by additional internal classification of the entities.

Annotation	Meaning
VDM.auxiliaryEntity .for.entity	Establishes a connection between an auxiliary entity and its supported entity. Scope: #VIEW Evaluation Runtime (Engine): None - Used for SAP internal structuring and interpretation of the CDS entities
VDM.auxiliaryEntity .usage.type	Describes the intended usage of the auxiliary entity. Scope: #VIEW Evaluation Runtime (Engine): None - Used for SAP internal structuring and interpretation of the CDS entities Auxiliary entities may not be defined for auxiliary entities. Auxiliary entities may only be defined for VDM entities. Both, the usage type and origin entity have to be defined in parallel.

Annotation	Meaning														
VDM.viewType	<p>Defines the type of a VDM view</p> <p>Scope: #ENTITY</p> <p>Evaluation Runtime (Engine): None - Used for SAP internal structuring and interpretation of the CDS views</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>BASIC</td><td>Views that form the core data basis without data redundancies.</td></tr> <tr> <td>COMPOSITE</td><td>Views that provide data derived and/or composed from the BASIC views.</td></tr> <tr> <td>CONSUMPTION</td><td>Views that serve for specific application purposes and are defined based upon BASIC or COMPOSITE or TRANSACTIONAL views.</td></tr> <tr> <td>EXTENSION</td><td>Technical carrier views for key user driven field extensibility.</td></tr> <tr> <td>TRANSACTIONAL</td><td>Special composite views that define the transactional runtime of an object (including draft support).</td></tr> <tr> <td>DERIVATION_FUNCTION</td><td>Views that can be used to define derivations (via the annotation @Consumption.derivation) for parameter values and for filter values (via the annotation @Consumption.filter).</td></tr> </tbody> </table>	Value	Description	BASIC	Views that form the core data basis without data redundancies.	COMPOSITE	Views that provide data derived and/or composed from the BASIC views.	CONSUMPTION	Views that serve for specific application purposes and are defined based upon BASIC or COMPOSITE or TRANSACTIONAL views.	EXTENSION	Technical carrier views for key user driven field extensibility.	TRANSACTIONAL	Special composite views that define the transactional runtime of an object (including draft support).	DERIVATION_FUNCTION	Views that can be used to define derivations (via the annotation @Consumption.derivation) for parameter values and for filter values (via the annotation @Consumption.filter).
Value	Description														
BASIC	Views that form the core data basis without data redundancies.														
COMPOSITE	Views that provide data derived and/or composed from the BASIC views.														
CONSUMPTION	Views that serve for specific application purposes and are defined based upon BASIC or COMPOSITE or TRANSACTIONAL views.														
EXTENSION	Technical carrier views for key user driven field extensibility.														
TRANSACTIONAL	Special composite views that define the transactional runtime of an object (including draft support).														
DERIVATION_FUNCTION	Views that can be used to define derivations (via the annotation @Consumption.derivation) for parameter values and for filter values (via the annotation @Consumption.filter).														
VDM.private	<p>Private views represent technical helper views which may only be used by their defining responsibilities.</p> <p>Scope: #VIEW, #TABLE_FUNCTION</p> <p>Evaluation Runtime (Engine): None - Used for SAP internal structuring and interpretation of the CDS entities</p>														
VDM.viewExtension	<p>Defines an extension of a VDM view. It is not allowed to use @VDM.viewExtension in combination with @VDM.viewType or @VDM.private or @VDM.lifecycle (on view level). The latter annotations are inherited from the extended view.</p> <p>Scope: #EXTEND_VIEW</p> <p>Evaluation Runtime (Engine): None - Used for SAP internal structuring and interpretation of the CDS entities</p>														
VDM.lifecycle.contr act.type	<p>Describes the intended publication or usage of the view as an API</p> <p>Scope: #ENTITY</p> <p>Evaluation Runtime (Engine): None - Used for SAP internal structuring and interpretation of the CDS entities</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> </table>	Value	Description												
Value	Description														

Annotation	Meaning										
	<p>NONE</p> <p>Not subject to any contract. Default for private and consumption VDM views.</p>										
	<p>PUBLIC_LOCAL_API</p> <p>System local API contract. Default for VDM Basic and Composite Views.</p>										
	<p>SAP_INTERNAL_API</p> <p>Marks a BASIC or COMPOSITE view for SAP internal usage only. Default for TRANSACTIONAL views.</p>										
	<p>PUBLIC_REMOTE_API</p> <p>Annotated CDS view is used to define a remote API.</p>										
VDM.lifecycle.status	<p>Defines the lifecycle status of an entity or element</p> <p>Scope: #ENTITY, #ELEMENT</p> <p>Evaluation Runtime (Engine): None - Used for SAP internal structuring and interpretation of the CDS entities</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>DEPERECATED</td><td>The annotated CDS entity or element is deprecated and shall no longer be used. Instead the given successor shall be used.</td></tr> </tbody> </table>	Value	Description	DEPERECATED	The annotated CDS entity or element is deprecated and shall no longer be used. Instead the given successor shall be used.						
Value	Description										
DEPERECATED	The annotated CDS entity or element is deprecated and shall no longer be used. Instead the given successor shall be used.										
VDM.lifecycle.successor	<p>Defines the name of the successor of a deprecated entity or element.</p> <p>Scope: #ENTITY, #ELEMENT</p> <p>Evaluation Runtime (Engine): None - Used for SAP internal structuring and interpretation of the CDS entities</p>										
VDM.usage.type	<p>Defines the usage scenario, which the VDM CDS entity shall support.</p> <p>Scope: #ENTITY</p> <p>Evaluation Runtime (Engine): None - Used for SAP internal structuring and interpretation of the CDS entities</p> <p>Values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>ACTION_PARAMETER_STRUCTURE</td><td>Input structure of an action respective function. This value may only be used for annotating CDS abstract entities.</td></tr> <tr> <td>ACTION_RESULT_STRUCTURE</td><td>Result structure of an action respectively function. This value may only be used for annotating CDS abstract entities.</td></tr> <tr> <td>EVENT_SIGNATURE</td><td>Signature of a business event. This value may only be used for annotating CDS abstract entities.</td></tr> <tr> <td>TRANSACTIONAL_PROCESSING_SERVICE</td><td>Used for CDS models which are exposed in services supporting the transactional processing.</td></tr> </tbody> </table>	Value	Description	ACTION_PARAMETER_STRUCTURE	Input structure of an action respective function. This value may only be used for annotating CDS abstract entities.	ACTION_RESULT_STRUCTURE	Result structure of an action respectively function. This value may only be used for annotating CDS abstract entities.	EVENT_SIGNATURE	Signature of a business event. This value may only be used for annotating CDS abstract entities.	TRANSACTIONAL_PROCESSING_SERVICE	Used for CDS models which are exposed in services supporting the transactional processing.
Value	Description										
ACTION_PARAMETER_STRUCTURE	Input structure of an action respective function. This value may only be used for annotating CDS abstract entities.										
ACTION_RESULT_STRUCTURE	Result structure of an action respectively function. This value may only be used for annotating CDS abstract entities.										
EVENT_SIGNATURE	Signature of a business event. This value may only be used for annotating CDS abstract entities.										
TRANSACTIONAL_PROCESSING_SERVICE	Used for CDS models which are exposed in services supporting the transactional processing.										

Examples

Example 1

Defining a basic VDM view:

↳ Sample Code

```
@VDM.viewType: #BASIC  
define view I_MyBasicView ...
```

Example 2

Defining a private composite VDM view:

↳ Sample Code

```
@VDM.private: true  
@VDM.viewType: #COMPOSITE  
define view P_MyPrivateCompositeView ...
```

Example 3

Defining Indian-specific VDM view extension:

↳ Sample Code

```
@VDM.viewExtension: true  
extend view I_MyExtendedView with X_IN_I_MyExtendedView ...
```

Example 4

Lifecycle annotations:

↳ Sample Code

```
@VDM.lifecycle.contract.type:#PUBLIC_LOCAL_API  
define view I_MyPublicLocalAPIView ...  
  
@VDM.lifecycle.contract.type:#PUBLIC_REMOTE_API  
define view A_MyPublicRemoteAPIView ...  
  
@VDM.lifecycle.status: #DEPRECATED  
@VDM.lifecycle.successor: 'I_MySuccessorView'  
define view I_MyDeprecatedView ...
```

Example 5

Auxiliary entity annotations:

↳ Sample Code

```
@VDM.auxiliaryEntity:{ for.entity: 'I_BasicView',  
                      usage.type: [#ENTERPRISE_SEARCH] }  
define view N_BasicView  
      as select from DatabaseTable1  
      left outer to one join DatabaseTable2
```

```

        on DatabaseTable2.Field2 eq DatabaseTable1.Field2
{
    key DatabaseTable1.Field1,
        cast( DatabaseTable2.Field3 as DataElement preserving type) as Field3
}
where DatabaseTable1.Field4 eq 'VALUE4'
and DatabaseTable1.Field5 eq 'VALUE5'

```

Example 6

VDM usage type:

↳ Sample Code

```

@VDM.usage.type:[#ACTION_PARAMETER_STRUCTURE]
define abstract entity D_SampleEntityActionParameter ...
@VDM.usage.type: [#EVENT_SIGNATURE]
@VDM.usage.type: [#TRANSACTIONAL_PROCESSING_SERVICE]
@VDM.viewType:#CONSUMPTION
@ObjectModel.transactionalProcessingDelegated:true
define view C_SampleTransactionalProcessingDelegatingView ...
@Event: {sapObjectType: 'SAPObject',...}
define abstract entity MyEventSignature ...

```

9.2 Understanding the BOPF Authorization API

The generic authorization concept is based on the abstract authorization check superclass /BOBF/CL_FRW_AUTHORITY_CHECK.

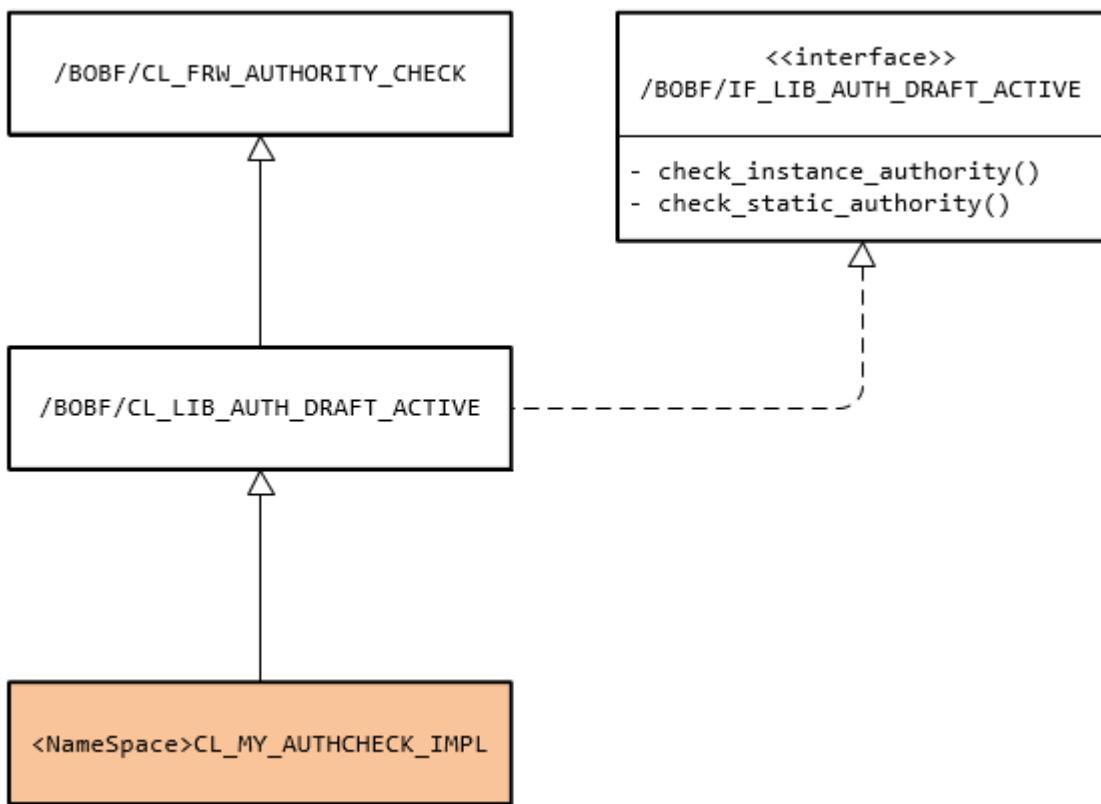
/BOBF/CL_FRW_AUTHORITY_CHECK

This **abstract** class is the superclass of the authorization check classes and acts as an interface definition.

/BOBF/CL_LIB_AUTH_DRAFT_ACTIVE

Authorization class for active or draft version of BO instances - for BOs that have been generated based on the CDS data model definition

When implementing application-specific authorization classes, you must inherit from this class and implement the abstract methods of interface /BOBF/IF_LIB_AUTH_DRAFT_ACTIVE.



Implementing an application-specific authorization class

/BOBF/IF_LIB_AUTH_DRAFT_ACTIVE

Interface used to check the authorizations for active and draft version of BO instances

This interface defines the following methods for implementing authorization checks.

Method Summary

Method	Description
CHECK_STATIC_AUTHORITY	Executes static authorization check
CHECK_INSTANCE_AUTHORITY	Executes instance-based authorization check

i Note

Note that `check_instance_authority` and `check_static_authority` methods are not suitable to limit read access when using OData services. To limit read access in this case, add the DCL-based access control to CDS views (BO CDS views and consumption CDS views). **More on this:** . Depending on the needs of your application however, it could still be necessary to implement authorization checks for read access using the BOPF API.

Method Details

- Method `CHECK_STATIC_AUTHORITY` [page 594]
- Method `CHECK_INSTANCE_AUTHORITY` [page 596]

9.2.1 Method `CHECK_STATIC_AUTHORITY`

Executes the static authorization check

→ Remember

Static authorization checks do not depend on BO instance data such as node attributes.

This method is usually called before the execution of a BOPF core service to check whether the user has authorization to execute a specific task, such as modifying data or performing an action (regardless of a specific instance that might be performed for). The context parameter `IS_CTX` provides information about the situation in which the authorization check will be executed.

Signature

<i>importing</i>	<code>IS_CTX</code>	type	<code>/BOBF/S_LIB_DRAFT_CTX_AUTH</code>
<i>exporting</i>	<code>EO_MESSAGE</code>	type <i>ref to</i>	<code>/BOPF/IF_FRW_MESSAGE</code>
<i>returning value</i>	<code>RV_FAILED</code>	type	<code>ABAP_BOOL</code>

Parameters

Parameter	Description
<code>IS_CTX</code>	Provides context information for the authorization check (authorization context) More: Context Information for Implementing Authorization Checks [page 599]
<code>EO_MESSAGE</code>	Message object used for returning information, warning, or error messages related to the execution of the authorization check
<code>RV_FAILED</code>	The value (<code>true/false</code>) indicates whether the authorization check failed

Example

The source code listing below provides you with an example for implementing static checks based on the sales order application scenario. You can use this source code as a template for your own implementations.

The main steps when implementing this method are as follows:

1. Executing static check for the relevant activity

The type of activity (CREATE, UPDATE, DELETE, and so on) is determined with the help of the context information `is_ctx-activity`. The corresponding values are defined in the constant structure `/BOBF/CL_FRW_AUTHORITY_CHECK=>SC_ACTIVITY`.

2. Message handling in case the static authorization check failed

The `eo_message->add_cm()` call is used to add the message to the BOPF message object. When instantiating the message object, you can specify, for example the error message ID, the error text for severity, the point in the lifetime of the business object and an original error location.

```
method /BOBF/IF_LIB_AUTH_DRAFT_ACTIVE~CHECK_STATIC_AUTHORITY.

  data: ls_message_textid type scx_t100Key.    " T100 key with parameters
        rv_failed = abap_true.                  " Deny access by default

  case is_ctx-activity.

    when /BOBF/CL_FRW_AUTHORITY_CHECK=>SC_ACTIVITY-create.
      " Check the creation of new (draft) instance here
      " This check is executed before a new (draft) instance data is
      created

    when /BOBF/CL_FRW_AUTHORITY_CHECK=>SC_ACTIVITY-display.
      " Note that DISPLAY authorization mostly depends on the DCL of the BO
      CDS view.
      " For DISPLAY authorization checks, this method is only executed in rare
      use cases.

    when /BOBF/CL_FRW_AUTHORITY_CHECK=>SC_ACTIVITY-delete.
      " Check the DELETE authorization here...
      if sy-subrc = 0.
        rv_failed = abap_false.      " Grant deletion
      else.
        rv_failed = abap_true.      " Deny deletion
        ls_message_textid = /bobf/cm_lib=>no_auth_delete.

      endif.

    when /BOBF/CL_FRW_AUTHORITY_CHECK=>SC_ACTIVITY-change.
      " Check the static UPDATE authorization here...
      if sy-subrc = 0.
        rv_failed = abap_false.      " Grant update
      else.
        rv_failed = abap_true.      " Deny update
        ls_message_textid = /bobf/cm_lib=>no_auth_update.

      endif.

    when /BOBF/CL_FRW_AUTHORITY_CHECK=>SC_ACTIVITY-execute.

      case is_ctx-action_name.

        when 'SET_TO_DELIVERED'.
          " Check static authorization for changing status of sales orders
          here...
          if sy-subrc = 0.
            rv_failed = abap_false. " Grant creation
          else.
            rv_failed = abap_true. " Deny creation
            ls_message_textid = /bobf/cm_lib=>no_auth_execute_action.  "
          endif.

        Generic message
```

```

        " You have the option to define an action-specific message
    endif.
endcase.

endcase.

" Message handling if authorization check failed
if rv_failed = abap_true and ls_message_textid is not initial.
    eo_message->add_cm( new /bobf/cm_lib(
        textid                  = ls_message_textid
        severity                = /bobf/cm_lib=>co_severity_error
        lifetime                = /bobf/
    cm_lib=>co_lifetime_transition
        ms_origin_location     = value #( bo_key      = is_ctx-
bo_key
                                         node_key = is_ctx-
node_key )
    )
).
endif.

endmethod.

```

Related Information

[Method CHECK_INSTANCE_AUTHORITY \[page 596\]](#)
[AUTHORITY-CHECK \(ABAP Keyword Documentation\)](#)

9.2.2 Method CHECK_INSTANCE_AUTHORITY

Executes instance-based authorization check

An instance-based authorization check depends on a BO instance data such as node attributes. Applications must implement this method to check whether the current user has the authorization required to perform a specific task for the given BO instance.

The context parameter `IS_CTX` is used as an input parameter in the method call. It provides information about the situation in which the authorization check is invoked (when performing a certain action for example).

Signature

<i>importing</i>	<code>IS_CTX</code>	<i>type</i>	<code>/BOBF/S_LIB_DRAFT_CTX_AUTH</code>
	<code>IT_KEY</code>	<i>type</i>	<code>/BOBF/T_FRW_KEY</code>
	<code>IO_READ</code>	<i>type ref to</i>	<code>/BOBF/IF_FRW_READ</code>
<i>exporting</i>	<code>EO_MESSAGE</code>	<i>type ref to</i>	<code>/BOPF/IF_FRW_MESSAGE</code>

ET_FAILED_KEY type /BOBF/T_FRW_KEY

Parameters

Parameter	Description
<i>IS_CTX</i>	Provides context information for the authorization check (authorization context) More: Context Information for Implementing Authorization Checks [page 599]
<i>IT_KEY</i>	Set of BOPF keys for BO instances for which the authorization check is to be executed
<i>IO_READ</i>	Reference that provides read access to the data of the business object instances More :
<i>EO_MESSAGE</i>	Message object used for returning information, warning, or error messages related to the execution of the authorization check
<i>ET_FAILED_KEY</i>	Set of keys for node instances (subset of <i>IT_KEY</i>) that do not match the criteria for authorization checks (keys with failed authority check)

Failed Authorization Check

In case of failed authorization checks, the parameter *ET_FAILED_KEY* must be filled with the failed instances. If messages should be forwarded to the consumer, the message container *eo_message* must be used.

Example

The following source code (listing below) provides you with an example (or rather a template) for implementing an instance-specific check based on the sales order application scenario.

The main steps when implementing this method are as follows:

1. Retrieving all data from the requested business object instances
To retrieve data of the node, the method call `io_read->retrieve()` is used. This method call exports the data of the requested node instances. If this node instance does not exist, its key is returned in the parameter `et_failed_key`.)
2. Executing static check for the relevant activity
The type of activity (CREATE, UPDATE, DELETE, and so on) is determined with the help of the context information `is_ctx-activity`. The corresponding values are defined in the constant structure `/BOBF/CL_FRW_AUTHORITY_CHECK=>SC_ACTIVITY`.

3. Message handling in case the static authorization check failed

The `eo_message->add_cm()` call is used to add the message to the BOPF message object. When instantiating the message object, you can specify, for example the error message ID, the error text for severity, the point in the lifetime of the business object and an original error location where you can provide specific information about the error location (using node-specific attributes `lt_attributes`).

```

method /bobf/if_lib_auth_draft_active~check_instance_authority.
  data:
    lv_denied      type abap_bool,                      " Specifies whether the
    user access is granted or denied
    lt_sales_order type ztdemo_i_salesorder_wd,   " Instance data typed with
    node's combined table type
    ls_message_textid type scx_t100key,                " T100 key with parameters
    lt_attributes   type /bobf/t_frw_name.             " BO attribute names
  " Retrieve the data of the requested node instance
  io_read->retrieve(
    exporting
      iv_node      = is_ctx-node_key
      it_key       = it_key
    importing
      et_data      = lt_sales_order
      et_failed_key = et_failed_key
  ).

loop at lt_sales_order assigning field-symbol(<s_sales_order>).
  lv_denied = abap_true. " Deny access by default
  case is_ctx-activity.
    when /bobf/cl_frw_authority_check=>sc_activity-delete.
      " Check the instance-specific DELETE authorization here...
      if sy-subrc = 0.
        lv_denied = abap_false.      " Grant deletion
      else.                        " Deny deletion
        ls_message_textid = /bobf/cm_lib=>no_auth_delete.
      endif.
    when /bobf/cl_frw_authority_check=>sc_activity-change.
      " Check the instance-specific UPDATE authorization here...
      if sy-subrc = 0.
        lv_denied = abap_false.      " Grant update
      else.                        " Deny update
        ls_message_textid = /bobf/cm_lib=>no_auth_update.
      endif.
    when /bobf/cl_frw_authority_check=>sc_activity-execute.
      case is_ctx-action-name.
        when 'SET_TO_DELIVERED'.
          " Check the instance-specific authorization for changing status of
          sales orders here...
          if sy-subrc = 0.
            lv_denied = abap_false.    " Grant action execution
          else.                      " Deny action execution
            " Generic message:
            ls_message_textid = /bobf/cm_lib=>no_auth_execute_action.
            " You have the option to define an action-specific message
            " In addition, you can link the message with specific
            attributes, for example:
            lt_attributes = value #( ( `CUSTOMER` ) ).
          endif.
        endcase.
      endcase.
    " Message handling if authorization check failed
    if lv_denied = abap_true.
      insert value #( key = <s_sales_order>-key ) into table et_failed_key.
      if ls_message_textid is not initial.
        eo_message->add_cm( new /bobf/cm_lib(
          textid           =
          ls_message_textid           " Message ID
        )
      )
    endif.
  endif.
endloop.

```

```

        severity      = /bobf/
cm_lib=>co_severity_error      " Error text for severity
        lifetime     = /bobf/
cm_lib=>co_lifetime_transition " Point in BO's lifetime
        ms_origin_location = value #( bo_key      = is_ctx-
bo_key       " Error location
node_key      " Relevant node key
key          " Node ID
        node_key     = is_ctx-
key          = <s_sales_order>-
        attributes =
lt_attributes )      " BO-specific attribute
)
).
clear: ls_message_textid.
endif.
endif.
endloop.
endmethod.

```

Related Information

Method [CHECK_STATIC_AUTHORITY \[page 594\]](#)
[AUTHORITY-CHECK \(ABAP Keyword Documentation\)](#)

9.2.3 Context Information for Implementing Authorization Checks

The context information is provided with parameter `IS_CTX` that provides further information about the situation in which the authorization check will be executed. This allows applications to closely control the behavior depending on the situation. It is used as an input parameter for all methods of the authorization check class.

This DDIC structure `/BOBF/S_LIB_DRAFT_CTX_AUTH` defines the authorization context in BOPF.

Structure Components Summary

Structure Components	Description
<code>BO_KEY</code>	Key of the business object that the authorization will be executed for
<code>NODE_KEY</code>	Node ID of the BO that the authorization check will be executed for

Structure Components	Description
ACTIVITY	<p>Activity type (create, change, delete and so on) that the authorization check will be executed for</p> <p>See constants in the DDIC structure /BOBF/ CL_FRW_AUTHORITY_CHECK=>SC_ACTIVITY</p>
ACTION_NAME	Name of the business object's action that the authorization check will be executed for

10 What's New in ABAP Programming Model

Here are descriptions of some of the changes of interest to developers made to ABAP Programming Model for SAP Fiori.

→ Remember

This documentation is integral part of the client installation of ABAP Development Tools (ADT).

Back End Shipments/Version vs. ADT Client Versions

Application Server ABAP 7.53	SAP NetWeaver AS for ABAP 7.52	SAP NetWeaver AS for ABAP 7.51 innovation package	ABAP Development Tools (Client)
-	-	-	Version ADT 3.4
1809 FPS 1 [page 601]	-	-	Version ADT 3.0
1809 [page 602]	-	-	Version ADT 2.96
	7.52, SP02 [page 602]	-	Version ADT 2.94
	7.52, SP01 [page 603]	-	Version ADT 2.89
	7.52, SP00 [page 604]	-	Version ADT 2.83
		7.51, SP03 [page 604]	Version ADT 2.80
		7.51, SP02 [page 605]	Version ADT 2.77

10.1 ABAP Platform 1809 FPS1 (7.53, SP01)

Value Help Definition

The annotation `@Consumption.ValueHelpDefinition` replaces `@Consumption.ValueHelp` and allows new functionalities. You can now define a value help without explicitly implementing an association to the value help provider view. You simply reference the value help provider view in the annotation with the respective element. In this way, you can also define multiple value helps for a single element or define an additional binding for the value help.

For further information, see [Providing Value Help \[page 275\]](#).

10.2 ABAP Platform 1809 (7.53, SP00)

Deriving Filter Options from Foreign Entities

Apart from deriving values for parameters, you can now derive filters from foreign entities. The frameworks allows you to implement single-value filters or filter ranges that are derived from foreign entities. The filter value is set automatically in CDS if it is annotated with the relevant annotations that provide the navigation information to the single value or the filter range. You have the option to determine the value(s) for the filter by using a parameter or an element binding.

For further information, look at [Deriving Values from Foreign Entities \[page 247\]](#).

Implementing Draft-Enabled Associations

A draft-enabled association retrieves active data, if you follow the association from the active source instance, and draft data, if you follow it from the draft source instance. An association that is not draft-enabled always retrieves active data even if the source entity is a draft instance. There are associations that are draft-enabled by default and associations for which draft-enablement needs to be implemented. This implementation is done by annotations on the association in the CDS view. It is now possible to draft-enable associations with cardinality to-many

For further information, refer to [Draft-Enabling for Reverse Associations \[page 294\]](#).

10.3 7.52, SP02

Using Aggregate Data in SAP Fiori Apps

Aggregate Data is used to combine numerical values to form a single value of signifying meaning. Aggregate functions, such as sum, maximum, minimum and average, as well as counting options are now available to be implemented in CDS and displayed in your SAP Fiori App. Annotations are used to mark the elements as measures whose values can be aggregated. The other elements are interpreted as dimensions or attributes of dimensions. The data records can be grouped by the dimensions whereby the measure values are aggregated with regard to the group.

For further information, see [Using Aggregate Data in SAP Fiori Apps \[page 263\]](#).

Implementing Draft-Enabled Associations

A draft-enabled association retrieves active data, if you follow the association from the active source instance, and draft data, if you follow it from the draft source instance. An association that is not draft-enabled always retrieves active data even if the source entity is a draft instance. There are associations that are draft-enabled by default and associations for which draft-enablement needs to be implemented. This implementation is done by annotations on the association in the CDS view.

For further information, refer to [Implementing Draft-Enabled Associations \[page 288\]](#).

10.4 7.52, SP01

Consuming Temporal Data

Temporal data is data that is time-dependent. The data properties are only valid during a specific validity period. If temporal data is stored in the database with table fields that define the beginning and the end of the valid time frame, you can model temporal data in CDS entities with the help of annotations that specify the relevant elements for the temporal data. This gives you the opportunity to retrieve temporal data by using a query option in OData requests.

For further information, look at [Consuming Temporal Data \[page 236\]](#).

Deriving Values from Foreign Entities

It is now possible to derive a value for a parameter of one CDS entity from a different one. This helps you to create dependencies between two different entities. The parameter is filled automatically in CDS if it is annotated with the relevant annotations that provide the navigation information to the value in the foreign entity. You have the option to determine the value of the foreign entity by using a parameter or an element binding.

For further information, look at [Deriving Values from Foreign Entities \[page 247\]](#).

Creating Conditions with the Simple Condition Factory

Filtering on virtual elements requires the substitution of the filter condition with a condition that takes only persistent elements into account. A new ABAP API provides a simple condition factory that uses methods for relational and logical operators. This makes it easy to concatenate several logical expressions.

For further information, look at [Creating Conditions with the Simple Condition Factory \[page 218\]](#).

Understanding Concepts

Business applications that follow a stateless programming paradigm are based on a RESTful protocol for communication. In various concept topics, we explain how the ABAP application infrastructure is used in transactional applications to ensure the data consistency for draft and active instances of business objects in the context of stateless programming.

For further information, look at [Concepts \[page 35\]](#).

10.5 7.52, SP00

Implementing Authorization Exits for Generated Business Objects

Business applications require an authorization concept for their data and for the operations on their data. Authorizations for CRUD operations and specific business-related activities are based on specific authorization exists that are implemented using a coreesponding BOPF API. Applications using business objects that have been generated based on the CDS data model definition, use a new BOPF API to implement their application-specific authorization checks. This API allows you to implement the authorization checks for active or draft version of BO instances.

For further information, look at: [Implementing Authorizations for Generated Business Objects \[page 160\]](#)

i Note

This authorization API has already been released with SAP NetWeaver AS for ABAP 7.51 innovation package, SP02.

10.6 7.51, SP03

Using Virtual Elements in CDS

Virtual elements represent transient fields in business applications. They come into play if these fields are not provided as part of the original persistence data model or cannot be easily integrated in CDS so that their values can be calculated directly on SAP HANA.

For further information, look at: [Using Virtual Elements in CDS \[page 206\]](#)

10.7 7.51, SP02

Combining Transactional Processing With Draft Capabilities

As the application developer, you have now the option to develop a completely new transactional apps for SAP Fiori UI that also provide the draft data support. This can be important for you, if you have to provide the end user with capabilities to store changed data at any time in the backend and proceed at a later point in time or to recover such data, even if the application client has crashed.

For further information, look at: [Developing New Transactional Apps with Draft Capabilities \[page 105\]](#)

Adding Dynamic Field and Entity Control

With dynamic fields and entities, you can control the visibility and changeability of fields or the entire business object nodes depending on the value selected from another field. In the context of the ABAP programming model for SAP Fiori, the field and entity control is a means in a transactional scenario that you can provide as information to the OData service on how data has to be displayed for consumption on the SAP Fiori UI.

For further information, look at: [Dynamic Field and Entity Control \[page 194\]](#)

Adding Dynamic Action Control

In addition to the field and entity control, you can also implement the visibility and changeability of actions. To provide the dynamic action control, the BOPF property determinations are used.

For further information, look at: [Dynamic Action Control \[page 201\]](#)

11 Glossary

ABAP Application Infrastructure (ABAP AI)

A combination of various technologies such as CDS (Core Data Services), BOPF (Business Object Processing Framework), SSDL (Service Adaption Description Language) and SAP Gateway.

It is introduced with SAP NetWeaver AS ABAP 7.5 to support the development of all types of SAP Fiori applications like transactional, search, analytics and planning apps.

ABAP CDS Entities

ABAP CDS entities (also referred as CDS entities) are data models based on the DDL (Data Definition Language) specification and are managed by the ABAP Dictionary.

Currently, the following types of ABAP CDS entities are supported:

- ABAP CDS views
- ABAP CDS table functions.

ABAP CDS Views

An ABAP CDS view (also referred as CDS view) is defined for existing database tables, database views, or for other CDS views by using the ABAP CDS statement DEFINE VIEW within a DDL source.

A CDS view serves to define the structure of an SQL view and represents a projection onto one or several ABAP Dictionary tables or ABAP Dictionary views.

For each CDS view, two objects are created in the ABAP Dictionary:

- An SQL view
- The actual CDS entity.

Active Data

Active data represents the state of a business entity (a business object, nodes of a business object) which is stored in the existing persistence (or original persistence).

Backend System

SAP S/4HANA On-Premise or SAP S/4HANA Cloud Edition backend system

BOPF

The Business Object Processing Framework (in short: BOPF) is an ABAP OO-based framework that provides a set of generic services and functionalities to speed up, standardize, and modularize development of business applications.

It manages the entire life cycle of business objects and covers all aspects of business application development. Furthermore, BOPF is a fundamental technology of the ABAP Application Framework.

Business Object (in BOPF)

In BOPF, a business object is represented as a hierarchical tree of nodes. A single node includes a set of semantically related attributes and the corresponding business logic. For each node, several types of entities can be defined to describe the specific business logic part of the business object.

Core Data Services (CDS)

CDS provides an infrastructure for defining and consuming semantically rich data models in SAP HANA.

In particular, ABAP CDS provides a framework for defining and consuming semantic data models on the central database of the application server AS ABAP. The specified data models are based on the data definition language (DDL) and the data control language (DCL).

CDS Annotations

Annotations describe semantics related to business data.

An annotation enriches a definition of a model element in the CDS with metadata. It can be specified for specific scopes of a CDS objects, namely specific places in a piece of CDS source code.

Consumption Model

Data model that represents an OData service by bundling a set of SAP Gateway service artifacts for a specific use-case (consumption).

Data Definition

ABAP development object used to define an ABAP CDS entity (for example, a CDS view).

After creation of a *Data Definition*, the developer is able to use the standard functions of the ABAP Workbench - such as syntax check, activation, or connecting to the *Transport Organizer*. The developer creates a *Data Definition* with the help of a wizard in *ABAP Development Tools*.

Data Model

Root entity that represents a certain self-contained business object and is used to define a people-centric view on respective business information.

Draft Data

A draft instance represents the transient state of a business entity (a business object, nodes of a business object) until it is permanently stored in the original persistency as an active instance. A draft instance is used to store inactive data in the draft persistency until its transition to the active data.

Element List

An element list in a CDS view contains all parameters and their aliases that are used by CDS annotations.

• Example

If you need to use the parameter `p_StartDate`, you add it to the element list of the corresponding CDS view by following declaration:

```
$parameters.p_StartDate as StartDate
```

EPM

Enterprise Procurement Model (in short: EPM) is a demo application that integrates many SAP NetWeaver technologies used by SAP Business Suite applications.

It is intended to be used for demonstration and testing purposes in the SAP NetWeaver technology stack. This demo application is based on real-world business logic and scenarios that can be used for various testing approaches and also provides you with utilities to generate meaningful test data.

ETag Check

The ETag check is used to determine whether two representations of a business entity, such as an active instance and the corresponding draft BO instance, are the same (draft scenario). If the representation of the entity ever changes, a new and different ETag value is assigned.

Usually, fields like last changed timestamp, hash values, or version counters are used as ETags.

Exposure

The CDS data model (CDS views), along with the CDS annotations fully specify the resulting OData model and the runtime behavior - without any additional metadata or OData-specific implementation.

Exposure means that the related CDS views, including all extensions and all applicable annotations, are mapped to a corresponding OData service.

Full Text Searching

Full text searching (or just text search) provides the capability to identify natural-language terms that satisfy a query and, optionally, to sort them by relevance (ranking) to the query.

Fuzzy Search

Fuzzy search is a fast and fault-tolerant search feature of SAP HANA. The concept behind the fault-tolerant search means that a database query returns records even if the search term (user input) contains additional or missing characters, or other types of spelling errors.

Hub System

SAP Gateway system

L UW (Logical Unit of Work)

When data in database tables is modified by application programs, it must be ensured that the data is consistent after the changes have been made. This is particularly important when data is edited in the database. The time span in which a consistent data state is transferred to another consistent state is known as an LUW (Logical Unit of Work).

OData

The Open Data (in short: OData) protocol is a Web protocol for querying and updating data. It applies several Web technologies such as HTTP, Atom Publishing Protocol, and JSON to provide access to information from a variety of applications.

OData is based on industry standards and offers database-like access to business data using REST-based (**Representational State Transfer**) architecture.

SADL

Service Adaptation Description Language (in short: SADL) is an ABAP technology that enables the consumption of entity relationship-like data models in ABAP based on a model-driven approach.

In context of SAP HANA, SADL enables fast read access to database data for scenarios on mobile and desktop applications using query push-down.

SAP Fiori UX

SAP Fiori is the new user experience (UX) for SAP software that applies modern design principles. SAP solutions, such as the SAP Business Suite powered by SAP HANA, are using the SAP Fiori UX to provide a personalized, responsive, and simple user experience.

SAP Gateway

Interface for communication between SAP backend system and external applications using the CPI-C protocol

SAP Gateway handles the communication between a client and the SAP Business Suite backend and is available as an SAP NetWeaver Application Server ABAP (AS ABAP) add-on, which you can be installed on top of SAP Business Suite application platform. SAP Gateway uses OData services to provide backend data and functions, and processes HTTPS requests for OData services.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
 - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
 - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

