

SAP - ABAP CDS Development User Guide | CUSTOMER
%NW-ASABAP-LONG-7.55% FPS00 and SAP Cloud Platform ABAP Environment
Document Version: 3.16 – 2021-02-03

SAP - ABAP CDS Development User Guide
Client Version 3.16

© 2020 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 About the ABAP CDS Development User Guide. 4

2 Concepts. 6

2.1 CDS Annotations. 6

Annotation Propagation. 8

2.2 Data Definitions. 10

ABAP CDS Entities. 10

ABAP CDS Code Templates. 22

Element Information for CDS Views. 24

Extending CDS Entities. 26

Documentation of CDS Objects. 32

2.3 Access Controls. 32

3 Tasks. 33

3.1 Creating and Activating Data Models. 33

Creating a Data Definition. 33

Creating ABAP CDS Objects With Reference to Other Objects. 37

Using ABAP CDS Code Templates for Data Definitions. 39

Activating Data Definitions. 41

3.2 Editing DDL Source Code. 44

Getting Support from the Content Assist. 45

Getting Help for DDL Source Code. 51

Displaying Details in the Element Information Popup and the ABAP Element Info View. 54

Navigating Associations. 56

Applying Quick Fixes. 57

Defining ON Conditions by Use of a Wizard. 58

Adding and Removing Comments. 61

Hiding CDS Annotations and Comments. 61

Changing Colors of DDL and DCL Source Code. 62

Comparing DDL Source Code Versions. 63

Formatting DDL. 63

Checking Syntax of DDL Source Code. 73

3.3 Accessing Data Models. 75

Adding Access Controls to CDS Entities. 75

3.4 Previewing Data Records. 78

Following Associations in the CDS Data Preview. 79

3.5 Extending Data Models. 82

Extending the Structure of Data Models. 82
Extending the Metadata of Data Models. 85
3.6 Analyzing Data Models. 90
Analyzing Dependencies in Complex CDS Views. 90
Analyzing Annotation Propagations. 91
Displaying Annotation Values of an Active CDS View. 92
3.7 Tuning Access to SAP HANA. 93
Creating Dynamic Caches. 94
3.8 Ensuring Quality with ABAP Unit. 95

4 Reference. 96
4.1 Active Annotations View. 96
4.2 Annotation Propagation View. 98
4.3 Dependency Analyzer. 100
SQL Dependency Tree. 101
SQL Dependency Graph. 104
Complexity Metrics. 106
4.4 Glossary. 107

5 What's New in ABAP CDS Tools 110
5.1 ☁ Version 3.14. 110
5.2 ☁ Version 3.12. 111
5.3 ☁ Version 3.6. 114
5.4 ☁ Version 3.0. 117

1 About the ABAP CDS Development User Guide

Scope of Documentation

This documentation describes the functionality and usage of tools for Core Data Services (CDS) in ABAP development for SAP HANA scenarios. In particular, it focuses on use cases for creating, editing, testing, and analyzing **ABAP CDS entities** using the Eclipse-based IDE.

Context

This guide provides documentation about features which are client-specific or require a specific back-end version.

Consequently, this documentation covers all client-specific and back-end-specific dependencies.

To highlight and contrast back-end-specifics in the relevant context, the following icons are used:

- ☁ for SAP Cloud Platform ABAP Environment shipments

Target Audience

ABAP developers who are involved in creating and defining data models including code push-downs.

Validity of Documentation

This documentation belongs to the ABAP Development Tools **client version 3.16** and refers to the range of functions that have been shipped as part of the standard delivery for:

- ☁ SAP Cloud Platform ABAP Environment

More on ABAP for SAP HANA Scenario

→ Tip

To view discussions and find further resources on how ABAP-based applications can leverage, you can also visit our SAP HANA and ABAP Development spaces on SAP community.

2 Concepts

2.1 CDS Annotations

A CDS annotation (or annotation for short) enables you to add ABAP and component-specific metadata to the source code of any CDS entity.

Types in Accordance to the Evaluating Runtime

In accordance with consistency and how validity of annotations is evaluated, SAP's annotations are divided into the following categories:

- **ABAP annotations** are evaluated by the ABAP runtime environment.
- **Component annotations** are evaluated by the relevant SAP framework.

Use

You can use code completion (`Ctrl` + `Space`) to add annotations directly in a data definition, for example, before the `define` statement or within a `select` list in a CDS view. The validity of the annotation then depends on the corresponding position where you use it. If they are added at the wrong position, the source editor will mark and underline them in red.

In addition, you can use annotations in metadata extensions to define customer-specific metadata for a CDS view without modifying SAP's CDS entities itself. When using metadata extensions, you can overwrite specific annotation values defined in a data definition or add additional annotation values to an entity. Note that you can only use those annotations in metadata extensions that are not relevant when activating CDS entities.

Example

```
@AbapCatalog.sqlViewName: 'CUSTOMER'
@AccessControl.authorizationCheck: #NOT_REQUIRED
@Metadata.allowExtensions: true
DEFINE VIEW cust_book_view_entity
     AS SELECT FROM scustom
     JOIN sbook
     ON scustom.id = sbook.customid
     {
    @EndUserText.label: 'Customer ID'
    scustom.id,
    @EndUserText.label: 'Customer Name'
           scustom.name,
    @EndUserText.label: 'Customer Booking ID'
           sbook.bookid
     }
```

The example from above demonstrates how you can use annotations and at which positions you can add annotations:

- Annotations that are used before the `define view` statement are valid for the whole `cust_book_view_entity` CDS view:
  - `@AbapCatalog.sqlViewName: 'CUSTOMER'`: After activation, the `CUSTOMER` CDS database view is created in the ABAP Dictionary.
  - `@AccessControl.authorizationCheck: #NOT_REQUIRED`: There is no access control required to retrieve the selected data from the database.
  - `@Metadata.allowExtensions: true`: Allows you or other developers to overwrite or add annotations in a metadata extension.
- The `@EndUserText.label` annotation used before an **element** in the `select` list provides a text for the corresponding field.

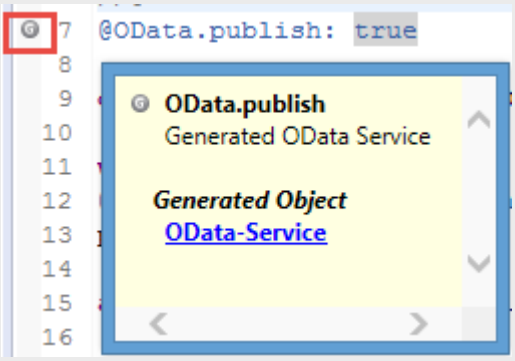
Activation

Errors resulting from the use of component annotations do not prevent activation or creation of a CDS entity at the first time. They are only evaluated if the activation of the entity was successful.

Component annotations can result in the generation of other ABAP repository objects.

Example

An OData service is generated when using the `@OData.publish: true` annotation. In this case, the annotation is highlighted with a marker that provides additional information about the generated object.



Marker that highlights the creation of an OData service

Note

When the CDS entity is activated, the OData service is generated automatically. After activation, it can be opened from the *ABAP Element Information* popup of the corresponding database table. To do this, select the underlined "OData-Service" link in the *Generated Object* section.

Related Information

ABAP CDS - Annotations
CDS Annotations
Annotation Propagation [page 8]
Active Annotations View [page 96]
Displaying Annotation Values of an Active CDS View [page 92]
Extracting CDS Annotations to a Metadata Extension [page 87]

2.1.1 Annotation Propagation

The values of CDS annotations can be inherited and merged between CDS entities.

Use

You as an ABAP developer have the following possibilities to use annotations in order to provide metadata in your data model:

- Use another CDS view as the data source
- Use a data element
- Use metadata extensions to enrich a CDS entity with customer-specific annotation values

You can build hierarchies when selecting data from other CDS views. In accordance to this hierarchy and the corresponding elements in the `select` list, the annotation values are propagated from bottom to top.

Using Metadata Extensions

In addition, you can also use metadata extensions. They can also reflect a hierarchy when assigning several metadata extensions to a CDS view.

You can assign customer-specific metadata through annotations in one or more metadata extensions to one data definition.

The precedence of the annotations contained in the metadata extensions is determined by the layer of the extension.

For this, the following values are provided:

Value	Description
`CUSTOMER`	Used by SAP's customers to define their own metadata **Example** `@Metadata.layer: #CUSTOMER` In this example, the value `CUSTOMER` is used for a metadata extensions. **Note** All annotations provided in metadata extensions are compounded with the annotations in the corresponding data definition. Element annotations (scope *ELEMENT*) are propagated in the entity hierarchy. `CUSTOMER` is the highest level. If there are several metadata extensions provided for a data definition, the metadata extension with the highest value will be considered.
`PARTNER`	Used by SAP partners to define their own metadata
`INDUSTRY`	Used by SAP to define metadata for industry solutions
`LOCALIZATION`	Used by SAP to define regional or country-specific metadata
`CORE`	Used by SAP to define metadata of their basis applications

Annotation Propagation View

The value of annotations can be propagated within the entity hierarchy.

You use the *Annotation Propagation* view to display the currently active and inactive values of CDS annotations and the CDS entities from which these values have been propagated in accordance to the current position of the cursor in the DDL editor.

This view displays the following information:

- Source CDS entity from which the value of a CDS annotation originates.
- If you provide several metadata extensions for a data definition, you can reproduce how metadata extensions provide metadata on different layers.

After generating, all involved annotation values and their corresponding data sources are listed. The effective entries are highlighted in black. Based on this list, you can now check which values are considered from your data definition.

Note

The metadata that is ignored is highlighted grey.

You can also adapt the selection of the data source at any time. To do this, choose the corresponding *Browse...* button in the *Selection* area. Select then the relevant data source or key.

Related Information

ABAP CDS - Evaluation of Annotations (ABAP Keyword Documentation)
Annotation Propagation View [page 98]
Analyzing Annotation Propagations [page 91]

2.2 Data Definitions

A CDS data definition is an ABAP repository object. It is created using the CDS DDL of the ABAP CDS in data definition language (DDL) source code.

Data definitions define CDS entities that can be accessed as a data type in ABAP programs and as a data source in reading ABAP SQL statements.

You can create and define data definitions for the following CDS entities:

- CDS View Entities [page 12]
- CDS Projection View [page 14]
- CDS DDIC-Based Views [page 15]
- CDS Table Functions [page 17]
- CDS Hierarchies [page 19]
- CDS Custom Entities [page 21]
- CDS Abstract Entities [page 22]

To do this, in the creation wizard use the relevant code template.

Related Information

Creating and Activating Data Models [page 33]
ABAP CDS Code Templates [page 22]
ABAP CDS - Data Definitions (ABAP Keyword Documentation)

2.2.1 ABAP CDS Entities

A CDS entity is a structured object in ABAP CDS. It represents a data model based on the data definition language (DDL) specification and are managed by ABAP Dictionary.

The following types of ABAP CDS entities are supported:

- CDS Views [page 11]
- CDS Table Functions [page 17]
- CDS Hierarchies [page 19]
- CDS Custom Entities [page 21]
- CDS Abstract Entities [page 22]

Related Information

ABAP CDS in ABAP Dictionary (ABAP Keyword Documentation)

2.2.1.1 CDS Views

A CDS view is a CDS entity defined in a CDS data definition that implements a view.

Use

Using CDS views, you can rearrange and rename the table fields according to application-specific needs from the ABAP source code of your implementation.

Defining CDS Views

A CDS view is defined for existing database tables and views, or for other CDS views in the ABAP Dictionary, starting with the `DEFINE VIEW ...` DDL statement. A CDS view represents a projection onto one or several database tables or database views in the ABAP Dictionary.

Example

```
@AbapCatalog.sqlViewName: 'CUSTOMER'
DEFINE VIEW ENTITY cust_book_cds_view AS SELECT FROM scustom
     JOIN sbook
     ON scustom.id = sbook.customid
     {
           scustom.id,
           scustom.name,
           sbook.bookid
     }
```

The `cust_book_cds_view` CDS entity view defines a projection onto the database tables `scustom` and `sbook` by joining both tables. The generated CDS-managed DDIC view (`CUSTOMER`) comprises the ID, the name, and the booking ID of all customers for which the bookings exist.

Accessing CDS Views in ABAP

CDS views can be used in ABAP SQL for data selection. The following method lists the customer's booking data that is stored in the underlying database tables. As demonstrated in the listing below, the CDS entity (in this case: `cust_book_cds_view`) is used for data selection in the ABAP source code.

Example

```
CLASS cl_demo_access_cds_entity IMPLEMENTATION.
...
   METHOD get_data.

    SELECT id name bookid
      FROM cust_book_cds_view
      INTO TABLE @DATA(result_data)
      WHERE ...   .
  ENDMETHOD.
...
ENDCLASS.
```

Help Content

The following tools documentation is provided for the relevant types of CDS views:

- CDS View Entities [page 12]
- CDS Projection View [page 14]
- CDS DDIC-Based Views [page 15]

Related Information

CDS View (ABAP Keyword Documentation)
Creating and Activating Data Models [page 33]
Editing DDL Source Code [page 44]

2.2.1.1.1 CDS View Entities

A CDS view entity is an ABAP repository object and part of a data definition in the ABAP CDS DDL source code.

Prerequisites

CDS view entities are only supported using SAP HANA.

Use

A CDS view entity is defined for existing database tables, views, or other non-abstract CDS entities using the `DEFINE VIEW ENTITY` statement in the CDS DDL of ABAP Core Data Services (CDS).

CDS view entities are the successor of CDS DDIC-based views and offer more and improved features. There is **no** more CDS-managed DDIC view or CDS DDIC-based view created upon activation.

> → Recommendation
>
> SAP recommends using CDS view entities instead of DDIC-based view due to technical improvements, such as performance at activation, and so on.

Comparison of CDS DDIC-Based Views vs. CDS View Entities

The following table contrasts both CDS entities:

Description	CDS DDIC-Based Views	CDS View Entities
Optimized CDS activation	The CDS entity, CDS DDIC-based view, and the database view need to be activated.	• Only the CDS entity and the respective database view on SAP HANA need to be activated. Consequently, the performance of the activation runs faster. • Simplified handling of CDS extends
Syntax and annotation checks	Simple checks	Stricter checks that indicate critical situations more explicitly
Client handling	• Based on different algorithms • Controlled using the `ClientHandling.type` and `ClientHandling.algorithm` annotations	• Automatized for complete transparency • No annotation allowed for client handling
Database object	View / table function	View
CDS-managed DDIC view in ABAP Dictionary	Creation upon activation	No CDS-managed DDIC view is generated upon activation
Naming convention	Three names for the CDS object, CDS entity, and the CDS-managed DDIC view on SAP HANA	One name for the CDS object, CDS entity, and database view on SAP HANA

Sample Code

```
DEFINE VIEW ENTITY cust_book_cds_view_entity AS SELECT FROM scustom
    JOIN sbook
     ON scustom.id = sbook.customid
     {
           scustom.id,
           scustom.name,
           sbook.bookid
     }
```

The `cust_book_cds_view_entity` CDS entity works with the database tables `scustom` and `sbook`. It joins both tables and specifies the elements of both database tables in the select list.

Related Information

ABAP CDS - View Entities

2.2.1.1.2 CDS Projection View

A CDS projection view is a special view that is based on another CDS view and exposes only a subset of elements of the projected entity.

Use

A projection view enables you to expose a subset of data from an underlying data model, for example, to be used in an OData service. It is a direct projection of an underlying CDS view without parameters and exposes a subset of elements of the projected entity, which are defined in the list of elements.

In a business application, a projection view allows to restrict access to, denormalize, and fine-tune the underlying data model.

Use Cases

As an application developer, you want to, for example, ...

- contemplate a subset of the business object as a business service.
- alias or rename the projected subset.
- omit or hide chosen elements or associations.

Related Information

ABAP CDS - Projection Views (ABAP Keyword Documentation)

2.2.1.1.3 CDS DDIC-Based Views

A CDS DDIC-based view is a CDS view that is technically based on a CDS-managed DDIC view in the ABAP Dictionary. It is defined with the CDS DDL statement `DEFINE VIEW`.

Definition

A CDS DDIC-based view is a CDS entity that is defined in a CDS data definition.

Use

Using CDS views, you can rearrange the table fields according to application-specific needs from the ABAP source code of your implementation.

> → Recommendation
>
> SAP recommends using CDS view entities instead of CDS DDIC-based view due to technical improvements, such as performance at activation, and so on.

Activating CDS DDIC-Based Views

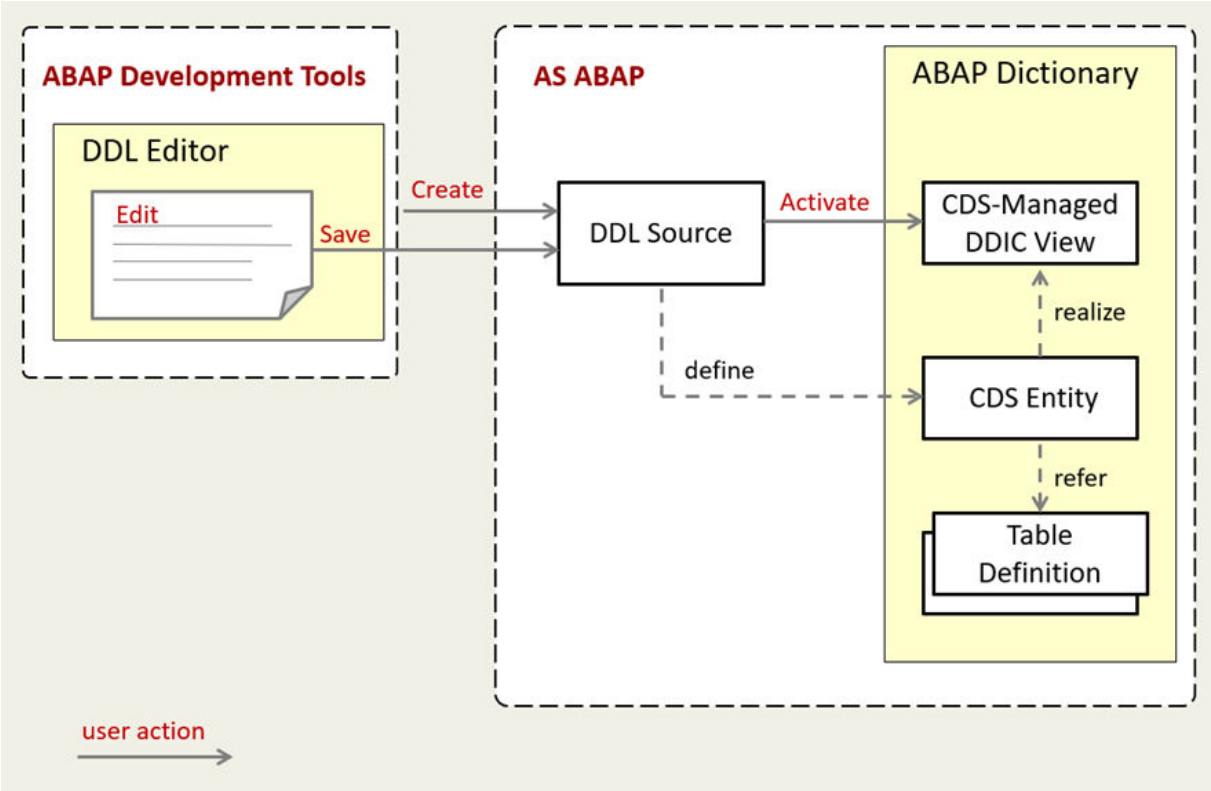
When activating a CDS view, the following objects are created in the ABAP Dictionary:

- The actual CDS entity
- A CDS-managed DDIC view

Overview of Process and Architecture

The following figure combines the main components of the view-building architecture and also displays the most important activities that are involved in the view-building process. Using a wizard within the Eclipse-based IDE, you first create the data definition as the relevant development object. In ABAP Development Tools (ADT), the text-based DDL editor is used to write source code in which you specify the data definition for a new CDS view. For each CDS DDIC-based view that is defined in the data definition, you will generate – using the

activation process – exactly one CDS-managed DDIC view and the corresponding CDS entity in the ABAP Dictionary.



CDS view building architecture

> **i Note**
>
> When activating a data definition, a CDS entity and CDS-managed DDIC view form a unity with the data definition as development object. So, after transporting the data definition, the name of the CDS entity and CDS-managed DDIC view can no more be changed. To rename any part of this unity, you need to delete the corresponding data definition. Consequently, you recreate it and use the new name for the relevant part.

> **⚠ Caution**
>
> Before deleting DDL, check whether it is still being used by other development objects. To find out if an object is still in use, call the where-used function ().

Related Information

ABAP CDS - DDIC-Based Views (ABAP Keyword Documentation)
CDS View Entities [page 12]
CDS Views Extension [page 28]
Creating and Activating Data Models [page 33]
Editing DDL Source Code [page 44]

2.2.1.2 CDS Table Functions

CDS table functions define table functions that are implemented natively on the database and can be called in CDS. As such, they support the SAP HANA platform code push-down capabilities in ABAP CDS.

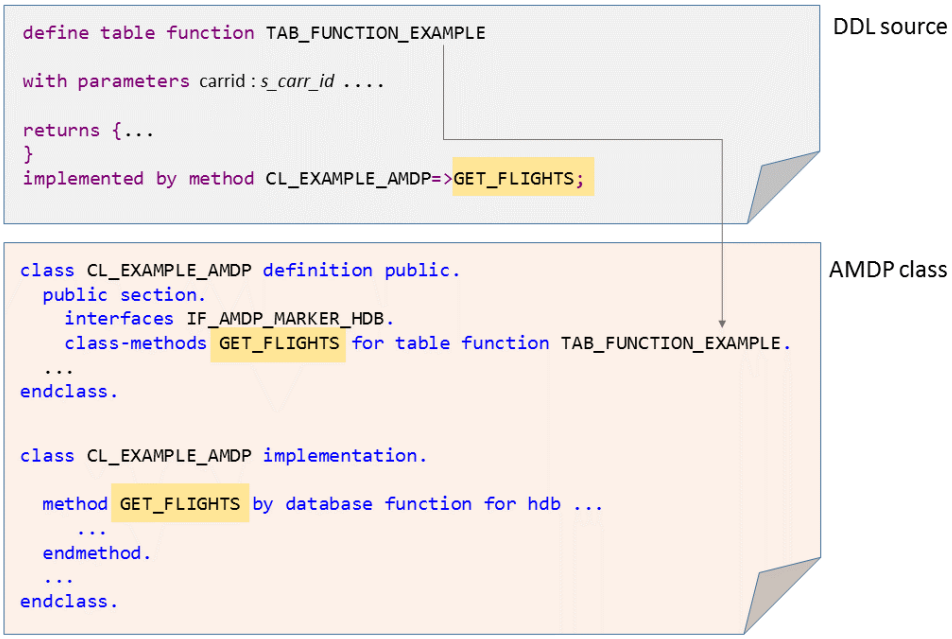
Defining ABAP CDS Table Functions

A CDS table function is defined using the ABAP CDS statement `DEFINE TABLE FUNCTION`.

Each CDS table function includes the following components:

- The actual **CDS entity** of the table function that is generated in the ABAP Dictionary
- The **CDS table function implementation** (ABAP class library)

> **i** Note
>
> In contrast to the CDS views, the CDS table functions can be implemented using Native SQL. This implementation is done within an AMDP method of an AMDP class and is managed as an AMDP function by the AMDP framework in the database system.



Defining and Implementing CDS table functions

> **i** Note
>
> The name of the implementing AMDP method can only be specified in a single CDS table function (1: 1 relation).

Sample

Table Function Definition

In the following listing, a client-specific ABAP CDS table function `TAB_FUNCTION_EXAMPLE` is defined using the DDL syntax. This table function declares two input parameters `clnt` (with the predefined value: `#CLIENT`) and `carrid`, and a list of elements that provide the return values of the AMDP method that implements the table function. The table function is associated with the AMDP class `CL_EXAMPLE_AMDP`, where the method `GET_FLIGHTS` is used to implement the table function.

Sample Code

```
@ClientDependent: true
 @AccessControl.authorizationCheck: #NOT_REQUIRED
 define table function TAB_FUNCTION_EXAMPLE

    with parameters @Environment.systemField: #CLIENT
            clnt:abap.clnt, carrid : s_carr_id
    returns {
         client : s_mandt;
         carrname : s_carrname;
         connid : s_conn_id;
         cityfrom : s_from_cit;
         cityto : s_to_city;

        }

    implemented by method CL_EXAMPLE_AMDP=>GET_FLIGHTS;
```

Table Function Implementation

The public ABAP class (AMDP class) in this example provides the AMDP method `get_flights`, which serves as the implementation of the `table function tab_function_example`. As with any other AMDP class, `cl_example_amdp` must implement the marker interface `IF_AMDP_MARKER_HDB`. The AMDP method `get_flights` implements the data selection using Native SQL code.

Sample Code

```
CLASS cl_example_amdp DEFINITION PUBLIC.

     PUBLIC SECTION.
        interfaces IF_AMDP_MARKER_HDB.
        class-methods get_flights  FOR TABLE FUNCTION tab_function_example.

     PROTECTED SECTION.
     PRIVATE SECTION.
 ENDCLASS.


 CLASS cl_example_amdp IMPLEMENTATION.

    METHOD get_flights BY DATABASE FUNCTION
        FOR HDB
        LANGUAGE sqlscript
        OPTIONS read-only
        USING scarr spfli.
        RETURN SELECT sc.mandt as client,
            sc.carrname, sp.connid, sp.cityfrom, sp.cityto
            FROM scarr AS sc
            INNER JOIN spfli AS sp ON sc.mandt = sp.mandt AND sc.carrid =
sp.carrid
```

```
            WHERE sp.mandt = :clnt AND
                  sp.carrid = :carrid
            ORDER BY sc.mandt, sc.carrname, sp.connid;
    ENDMETHOD.
ENDCLASS.
```

Developer-Relevant Activities

1. Creating a Data Definition [page 33]
2. Defining a CDS table function
   See also: Editing DDL Source Code [page 44]
3. Adding Access Controls to CDS Entities [page 75]
4. Checking Syntax of DDL Source Code [page 73]
5. Activating Data Definitions [page 41]
6. Implementing a CDS table function in the AMDP class
7. Debugging CDS Table Functions
   See also:

Related Information

ABAP CDS Synatx – Table Functions (ABAP Keyword Documentation)

2.2.1.3 CDS Hierarchies

Hierarchies are a popular method to structure application data for easier consumption. A CDS hierarchy enables you to access data which relates to each other on a hierarchical way.

Prerequisites

CDS view entities are only supported using SAP HANA.

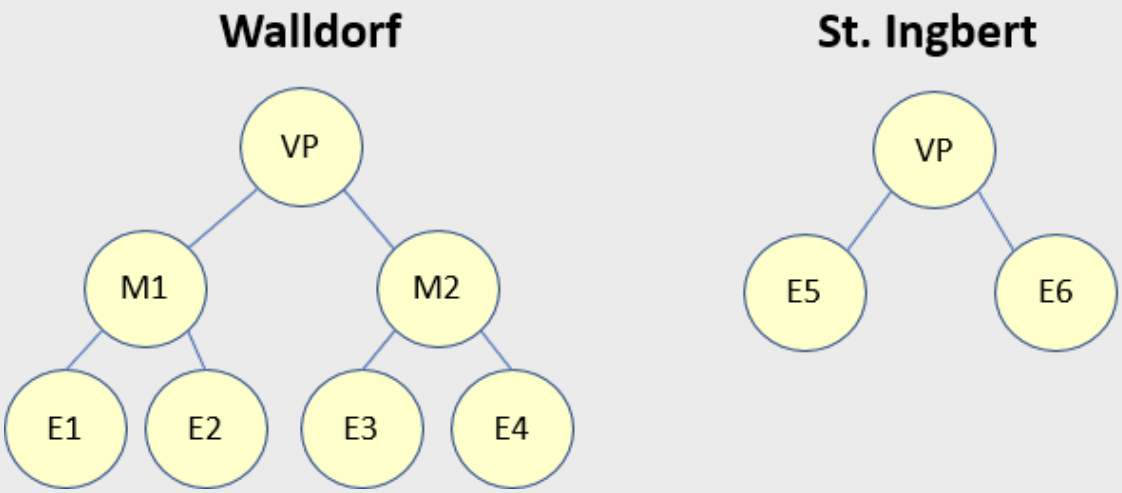
Use

When a CDS hierarchy is activated, a view with the same name is created as an ABAP-managed database object in the database. In ABAP SQL read statements, a CDS hierarchy can be accessed as an SQL hierarchy. Here, additional hierarchy columns contain the hierarchy attributes which can be read.

As a global data type, the CDS entity represents a structured type with the elements of the CDS hierarchy as components and can be used like any CDS entity.

A CDS association is used to define the parent-child relationships of an SQL hierarchy or CDS hierarchy. The hierarchy association defines the parent-child relationship between the hierarchy nodes. A hierarchy association is always a self-association.

✤ Example



Sample structure of a employee hierarchy

In a schema, you use two structures for the vice presidents (VP) from two locations (Walldorf and St. Ingbert) of your company. Each VP is responsible for one or more manager(s) (M) who are responsible for one or more employee(s) (E).

CDS hierarchies enable you to run queries, for example,

- to find out which manager is responsible for which managers and/or employees or
- who are the managers in Germany
- and so on.

Use Cases

As an application developer, you must consider, for example, one of the following use cases:

- To run complex queries on the employee database on a fast way, you need to model the employee/manager relationship of your company in an efficient way.
- Your company is structured into various legal entities and subunits. You now want to structure them in a hierarchy.

Related Information

ABAP CDS - Hierarchies
Creating a Data Definition [page 33]

2.2.1.4 CDS Custom Entities

A CDS custom entity is a non-SQL CDS entity with a custom query implemented in an ABAP class. They do not come with a `select` statement on the datasource.

Use

CDS custom entities are used for data models whose runtime is implemented manually.

They make it possible ...

- to provide data in AS ABAP using classes.
- to split the signature and implementation of the CDS entity.

> **i** Note
>
> It is no longer possible to change the type of the implementation once the CDS custom entity is transported.

Use Cases

The following use cases apply for custom entities:

- Data is stored in non-relational storage, for example, Binary Large Objects (BLOB)
- Data is stored in LiveCache and access is done using ABAP APIs
- Data cannot be computed by means of CDS due to additional logic on application server side
- Data model can be defined using CDS views, but data control language (DCL) feature set is not sufficient to define the necessary authorizations

Related Information

ABAP CDS - Custom Entities (ABAP Keyword Documentation)

2.2.1.5 CDS Abstract Entities

A CDS abstract entity defines the type attributes of a CDS entity without defining a database object.

Use

CDS abstract entities can be employed and used as ...

- data types whose type attributes go beyond the regular DDIC structures in the ABAP Dictionary
- prototype definitions of data models without being created as instances of a data object

Use Cases

As an application developer, you want to ...

- type both the input parameters and the result type of actions.
- define the data model for outside-in scenarios.

Related Information

ABAP CDS - Abstract Entities (ABAP Keyword Documentation)

2.2.2 ABAP CDS Code Templates

ABAP CDS code templates are structured descriptions of coding patterns that can be used in the ABAP CDS source code.

Definition

Templates are code patterns for CDS entities that contain variables, enclosed in `${}`, which are used as placeholder(s). At creation of a CDS object, the placeholder will be replaced with the actual value in the source code.

Overview

ABAP CDS code templates ...

- are provided for all CDS objects by SAP.
- are displayed and available in the creation wizard, *Templates* view and code completion in the source code editors.
- can be created for all CDS objects and extended from the *Templates* view or *Data Definition Templates* preference page.

  > **i** Note
  >
  > This documentation describes the possibility using the preferences page for data definitions. In addition, the following preference pages are available for ABAP CDS development:
  > - *Access Control Templates*
  > - *Annotation Definition Templates*
  > - *Behavior Definition Templates*
  > - *Metadata Extension Templates*

- can be imported or exported to share them, for example, with other ABAP developers of your team.

Use

You add a template to your source code by double-clicking it in the *Templates* view or by selection in the *Creation Wizard*.

> **→** Recommendation
>
> If you want to create your own templates, SAP recommends to ...
>
> - use the supported variables. To find all supported variables, see the *Data Definition Templates* preferences page.
> - create the template in the relevant template context. For ABAP development, there is only **one** template context provided. For ABAP CDS development, there is a template context for **each** CDS entity provided.

Context

A context is a group that contains one or more templates for each CDS entity, such as, CDS view entity, CDS DDIC-based view, CDS hierarchy, and so on. You can only use a template for a CDS entity of your choice which it is **part** of the relevant context.

If you create a new CDS object on base of a referenced object, the variables in the template will be replaced in accordance to the use in the referenced object.

> **i** Note
>
> The `data_source_elements` variable is used to insert the elements of the referenced object.

→ Recommendation

The *Data Definitions (deprecated)* context contains the templates that have been previously defined. SAP recommends, do **not** create your new templates in this context. The templates in this context will no more be updated by SAP.

Related Information

Using ABAP CDS Code Templates for Data Definitions [page 39]

2.2.3 Element Information for CDS Views

The *Element Information* popup and the *ABAP Element Info* view display details of ABAP Dictionary Objects used in the context of CDS.

Use

You can use the *Element Information* popup and the *ABAP Element Info* view while programming to get further details, for example, which element of a data source can be used for creating a data model and how it is typed.

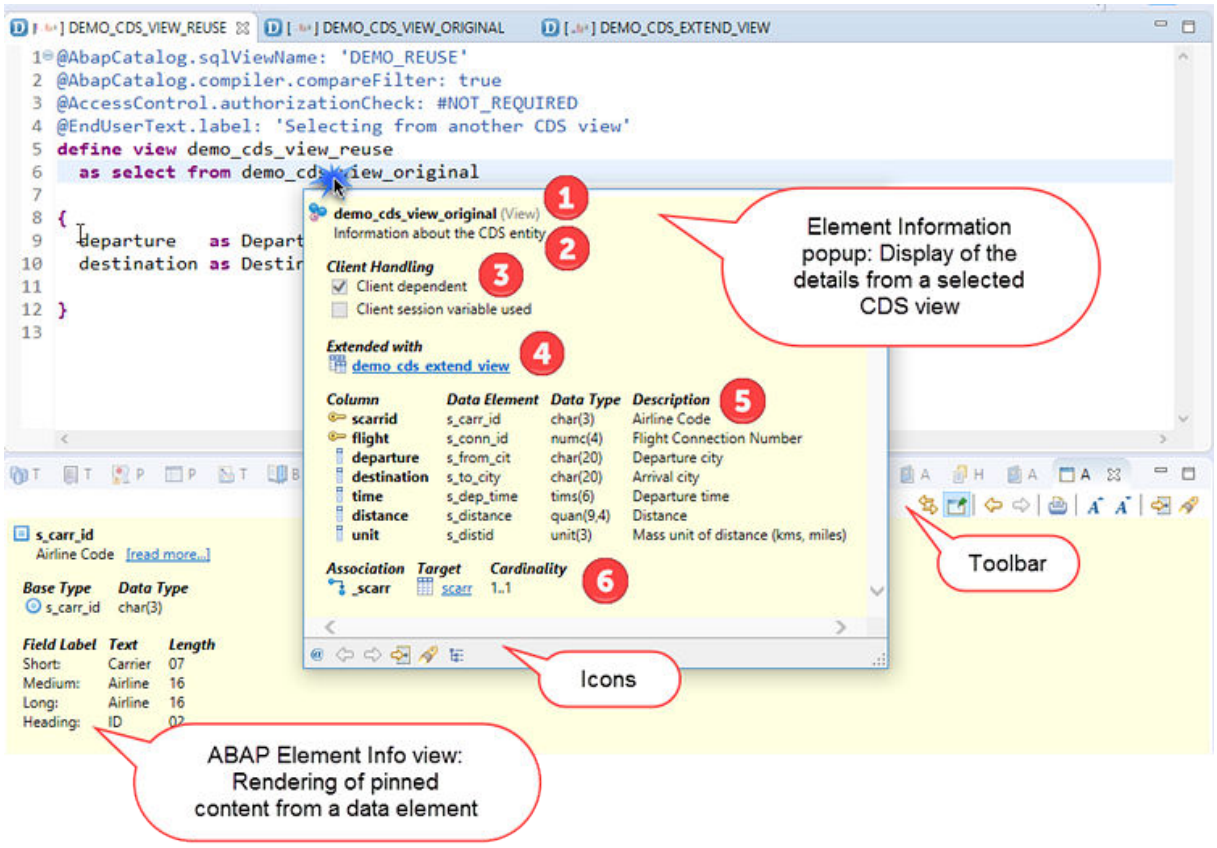
You can display the details of:

- Data sources used in from clauses, joins, and `as` targets of associations/compositions, including details about their elements
- Data elements used as parameter types and in cast statements

Overview

You open the *Element Information* popup from the relevant element in the CDS source code by choosing F2 or *Source Code* > *Show Code Element Information* from the context menu.

From this popup you can open the *ABAP Element Info* view by choosing the @ "Show in ABAP Element Info view" icon. You use this view, to display your CDS source code and the element information simultaneously.



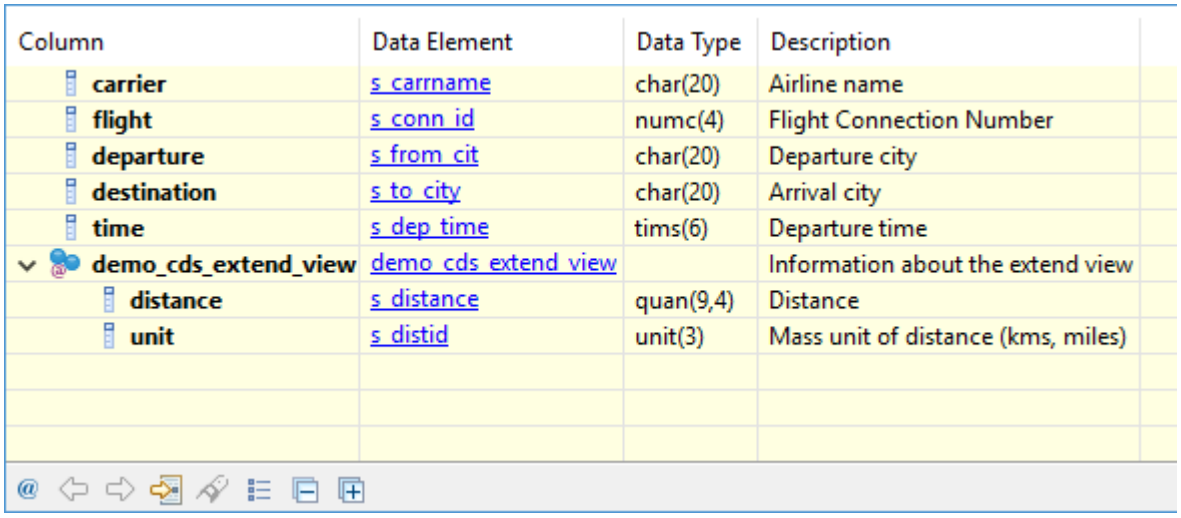
In this example, the Element Information popup and the ABAP Element Info view display CDS source code information for a CDS view

Here you will get the following information:

1. Name of the CDS entity and their types
2. The label specified in the `@EndUserText.label` annotation
3. Information about the automatic client handling for the CDS entity
   The *Client Handling* checkboxes provide additional information about the client handling of the CDS entity that is defined for the underlying data model.
   They provide an effective summary of the following client handling properties:
   - *Client dependent* if the CDS entity returns data that is specific to the logon client
   - *Client session variable used* is set if the `@ClientHandling.algorithm` CDS annotation is added with the `SESSION_VARIABLE` value to the CDS entity itself or to a CDS entity in the entity hierarchy

   > **i Note**
   >
   > This checkbox represents the effective state of the property in the system. Therefore, the checkbox and the specified annotation value might deviate in some cases.
4. One or more object links that refer to the relevant extensions of the CDS view
5. Details about the structure, for example, key elements, elements, parameters, and their types
6. Details about the public associations of the CDS entity

If you use an extension for a CDS view, you can also display the structure of the extend view in the append structure popup. To do this, choose the *Detail view with include/append structure* icon from the bottom or the toolbar.



Column	Data Element	Data Type	Description
carrier	s_carrname	char(20)	Airline name
flight	s_conn_id	numc(4)	Flight Connection Number
departure	s_from_cit	char(20)	Departure city
destination	s_to_city	char(20)	Arrival city
time	s_dep_time	tims(6)	Departure time
demo_cds_extend_view	demo_cds_extend_view		Information about the extend view
distance	s_distance	quan(9,4)	Distance
unit	s_distid	unit(3)	Mass unit of distance (kms, miles)

Example of the append structure popup of the extend view

This tree displays the elements from the select list and the elements from the extend view. By selecting the link in the *Data Element* column, you can navigate to the relevant type.

2.2.4 Extending CDS Entities

You can extend CDS entities of the SAP standard in order to add customer-specific functionality to a data model without resulting in modifications.

CDS View Extensions

A CDS view extension is a transportable extension of a CDS view that can be used to add fields from the entities used by the view (and other clauses) to the view without making modifications.

You use a CDS view extension to:

- add new elements or CDS annotations to a CDS view from its underlying data source or
- define new associations for the CDS view.

You can use the following CDS view extensions to extend CDS entities:

- **CDS Views Extension [page 28]** to add new elements to a CDS view from its underlying data source or define new associations for the CDS view.
- **Metadata Extensions [page 30]** to overwrite existing or add new CDS annotations to a one or more elements or parameters of a CDS entity.

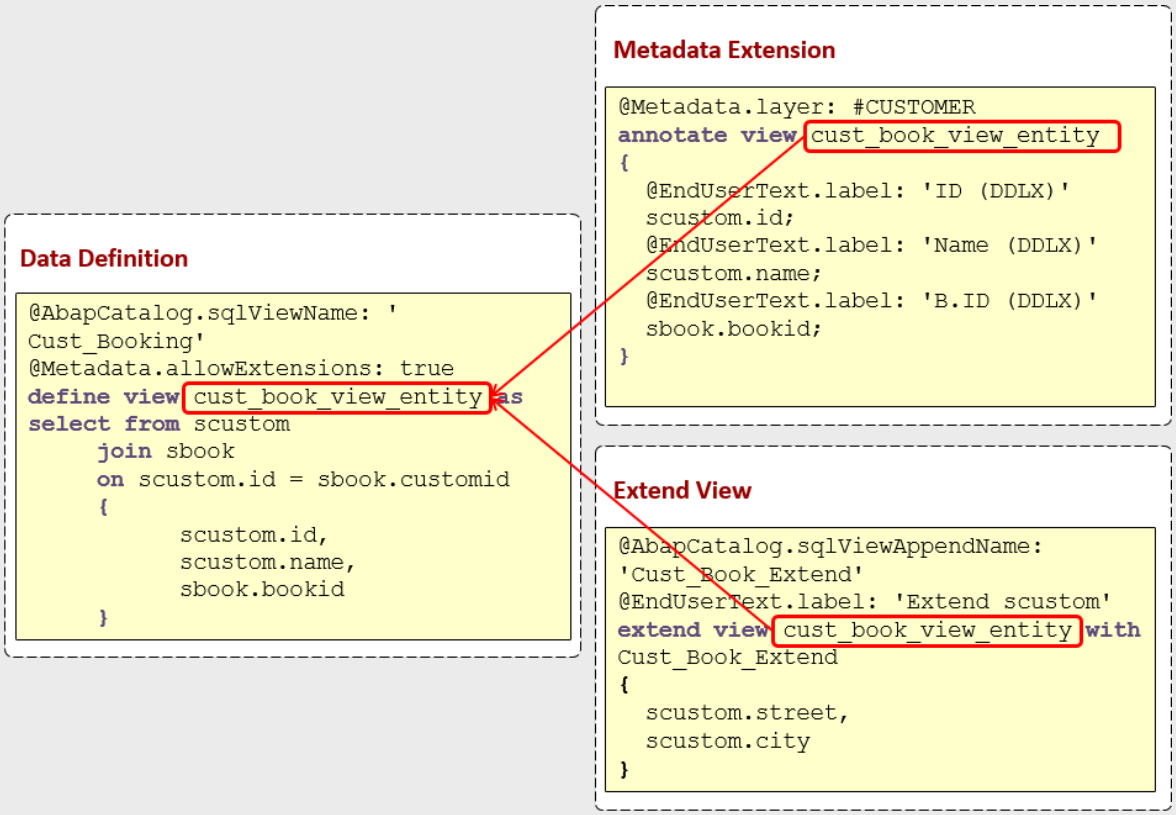
Overview

The following example shows you how to extend a CDS view:

Example

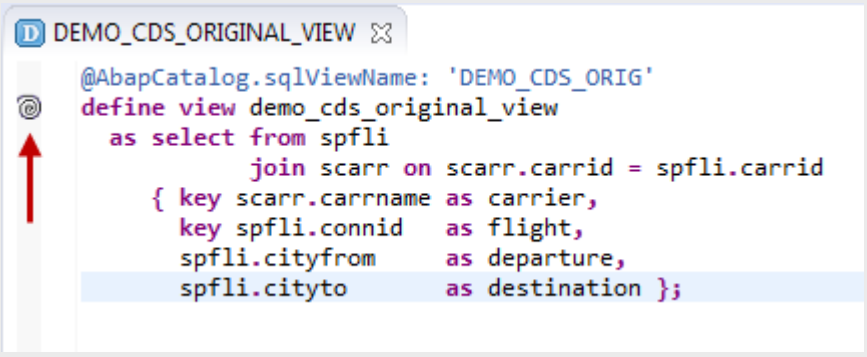
In the `select list` of the `cust_book_view_entity` CDS view:

- The metadata of the `scustom.id`, `scustom.name`, and `scustom.bookid` fields is overwritten by the metadata extension. When the corresponding data definition is consumed, the metadata of the metadata extension is taken into account.
- The `scustom.street` and `scustom.city` database fields are added through the extend view. When you select data from the corresponding data definition, the data of these database fields will also be retrieved.



Possibilities to overwrite existing CDS annotations as well as to add elements to a CDS view

After creating a CDS view extension, the ◎ indicator is added at the `define view` statement to indicate that the select list of the view has been extended.



Marker that indicates that the select list of the CDS view has been extended

2.2.4.1 CDS Views Extension

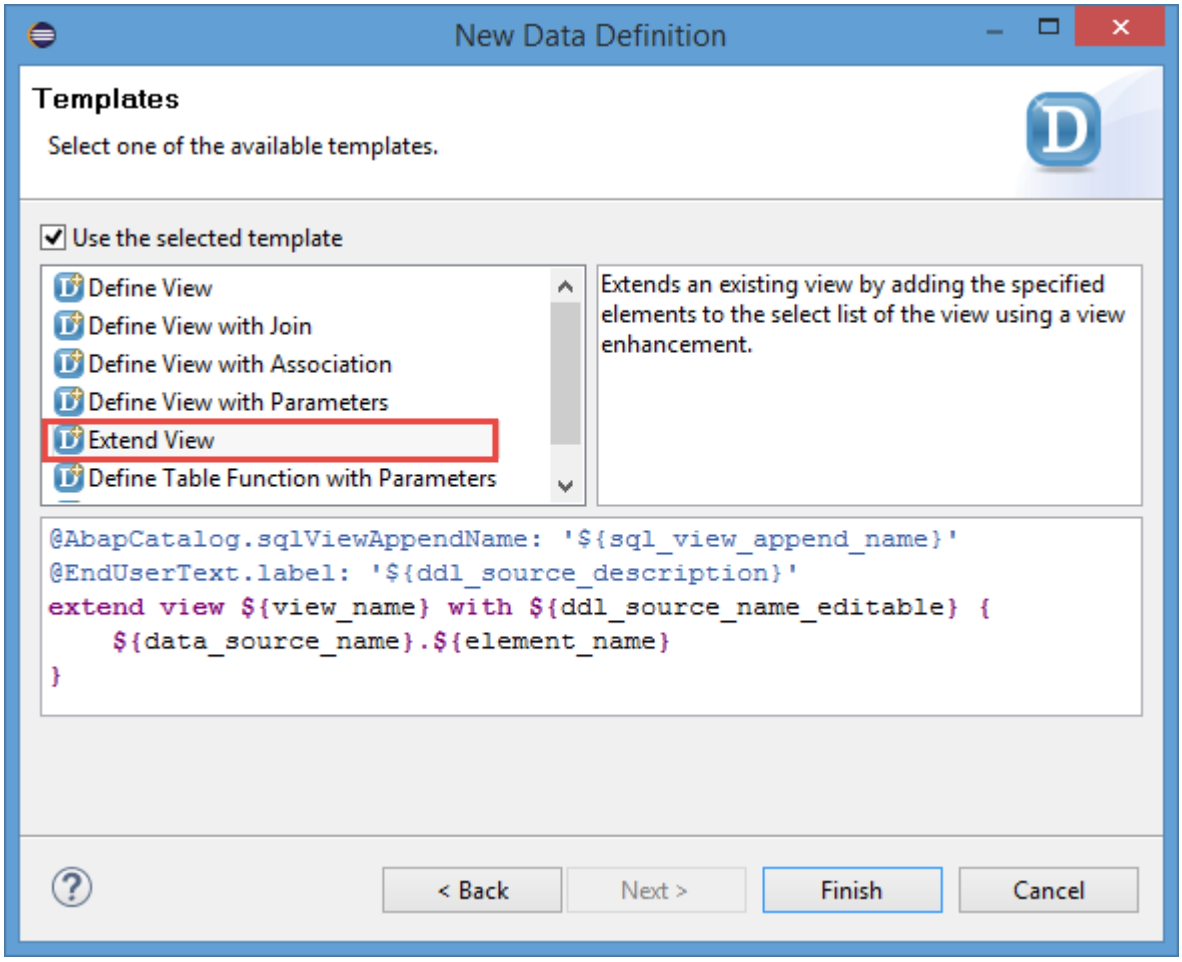
An CDS view extension is a repository object that extends an existing CDS view using the `extend view` statement. The CDS view extension itself is **not** a CDS entity.

Use

When extending, you add further element(s) from the data source (such as a database table or another CDS view) to the select list of an existing CDS view. So, you can enrich a CDS view that is, for example, part of the SAP standard without resulting in modifications.

Creating and Defining View Extensions

You create CDS view extensions on the basis of the *Extend View* template that is provided in the creation wizard of data definitions.



Entry to add the Extend View template through the creation wizard of data definitions

In this template, the following placeholders are provided and need to be adapted:

- `${sql_view_append_name}`: Name of the append view to be created in the ABAP Dictionary when activating the extend view
- `${ddl_source_description}`: Description to provide further information about the extend view
- `${view_name}'`: Name of the CDS view to be extended
- `${ddl_source_name_editable}`: Name of the extend view
- `${data_source_name}`: Name of the data source from which you want to add new elements to the CDS view
- `${element_name}`: Name of the element(s) (for example, fields or associations) to be added

Sample Code

```
@AbapCatalog.sqlViewAppendName: 'DEMO_EXT_VIEW'
@EndUserText.label: 'Demo'
extend view Demo_Data_Model_Base with Demo_Extend_View {
  spfli.countryfr as CountryFrom,
  spfli.countryto as CountryTo
}
```

The `Demo_Extend_View` extend view extends the select list of the `Demo_Data_Model_Base` CDS view with the `countryfr` and `countryto` fields. This data is now also provided whenever the `Demo_Data_Model_Base` CDS view is used.

Activating Extend Views

When activating a data definition containing the `extend view` statement,

- an append view that represents the added field(s) is created in the ABAP Dictionary
- the extended field(s) is added to the existing CDS view on the database

> **i** Note
>
> They are then
>
> - considered for every occurrence where the extended CDS view is used.
> - represented in the SQL Create statement.

Related Information

ABAP CDS - EXTEND VIEW (ABAP Keyword Documentation)
Creating CDS View Extensions [page 83]
Creating a Data Definition [page 33]

2.2.4.2 Metadata Extensions

Metadata extenstions 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.

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 entity

   In 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.

Related Information

Extracting CDS Annotations to a Metadata Extension [page 87]
Creating Metadata Extensions [page 86]
Annotation Propagation [page 8]

2.2.5 Documentation of CDS Objects

A knowledge transfer document (KTD) is a development object. It can contain documentation of a development object and their elements for which it has been created. A KTD is based on markdown markup language with plain text formatting syntax.

Use

You use a KTD to provide documentation for the following CDS objects:

- Data definitions (DDLS)
- Service definitions (SRVD)

Creation

You create a KTD from the context menu of the supported CDS object in the *Project Explorer*. To do this, select the CDS object and choose *New Knowledge Transfer Document*.

After creation, a development object is created for the KTD in the back-end and added to the *Texts* folder in the *Project Explorer*.

2.3 Access Controls

Related Information

Adding Access Controls to CDS Entities [page 75]
ABAP CDS - Access Control (ABAP Keyword Documentation)

3 Tasks

3.1 Creating and Activating Data Models

ABAP Core Data Services (CDS) enables you to define semantic data models which can be used in business applications.

In ABAP Development Tools (ADT), you define the CDS entities for your data model using data definitions.

To create a data definition, you use the creation wizard. After you create the data definition, the back-end creates an inactive version of it in the ABAP Repository and opens it in the source-based editor. From here you can start defining the CDS entity.

When activating the inactive version of a data definition, the CDS entity is created in the ABAP Dictionary and it can then be used in business applications.

> **i** Note
>
> In the creation wizard, ADT provides different templates to define different types of CDS entities.
>
> Choose the relevant template for the required entity.

Related Information

Creating a Data Definition [page 33]
Using ABAP CDS Code Templates for Data Definitions [page 39]
Creating ABAP CDS Objects With Reference to Other Objects [page 37]
Activating Data Definitions [page 41]

3.1.1 Creating a Data Definition

A data definition is a repository object that allows you to define a CDS entity using the CDS DDL of ABAP CDS in DDL source code. It also accesses the standard functionality (transport, syntax check, activation).

Prerequisites

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

Context

A CDS entity can be used, for example, as a datasource in other CDS entities.

You can create the following CDS entities using a data definition:

List of the Supported CDS Entities

CDS Entity	Template Context	Back-End Dependencies	Use Case	Further Information
CDS view entity	*CDS View Entity*		You want to create a data model which retrieves the relevant data, for example, from a database table on the recommended way. → Recommendation SAP recommends using CDS view entities due to technical improvements, such as performance at activation, and so on.	• CDS View Entities [page 12] • CDS DDL - DEFINE VIEW ENTITY (ABAP Keyword Documentation)
CDS view extension	*Extend View*		You want to extend an existing CDS view.	• CDS DDL - EXTEND VIEW ddic_based_view (ABAP Keyword Documentation)
CDS view extension entity	*Extend View Entity*		You want to extend an existing CDS view entity.	• CDS DDL - DEFINE VIEW ENTITY (ABAP Keyword Documentation)

CDS Entity	Template Context	Back-End Dependencies	Use Case	Further Information
Abstract CDS entity	*Abstract Entity*		You want to model an abstract CDS entity like a structure without persisting a new ABAP Dictionary object. To do this, you use abstract entities, for example, during development of an action for an OData service. In this case, abstract entities are used to define the type information for the action parameters.	• CDS Abstract Entities [page 22] • ABAP CDS - DEFINE ABSTRACT ENTITY (ABAP Keyword Documentation)
CDS custom entity	*Custom Entity*		You want to access data that exceeds the CDS feature set.	• CDS Custom Entities [page 21] • CDS DDL - DEFINE CUSTOM ENTITY (ABAP Keyword Documentation)
CDS hierarchy	*Hierarchy*		You want to structure application data to make it easier to consume. This enables you, for example, to select data that is relevant for a specific time frame or to handle grouped objects simultaneously.	• CDS Hierarchies [page 19] • ABAP CDS - Hierarchies (ABAP Keyword Documentation)
CDS projection view	*Projection View*		In a transactional scenario, you want to create and define, for example, a consumption-specific OData service that only exposes relevant data of an underlying data model using a service definition and service binding.	• CDS Projection View [page 14] • CDS DDL - DEFINE VIEW ENTITY AS PROJECTION (ABAP Keyword Documentation)

CDS Entity	Template Context	Back-End Dependencies	Use Case	Further Information
CDS table function	*Table Function*		You want to use a CDS table function as the data source in ABAP SQL read statements.	• CDS Table Functions [page 17] • ☁ CDS DDL - DEFINE TABLE FUNCTION (ABAP Keyword Documentation)
CDS DDIC-based view	*CDS DDIC-Based View*		You want to create a data model which retrieves the relevant data, for example, from a database table. → Recommendation ☁ SAP recommends using CDS view entities due to technical improvements, such as activation, and so on.	• CDS Views [page 11] • ☁ CDS DDL - DEFINE VIEW ddic_based (ABAP Keyword Documentation)

Procedure

1. In your ABAP project, select the relevant package node in the *Project Explorer*.
2. Open the context menu and choose *New* > *Other...* > *Core Data Services* > *Data Definition* to launch the creation wizard.
3. In addition to the *Project* and *Package*, enter the *Name* and the *Description* for the data definition to be created.
4. [Optional:] If you want to create a data definition using the elements from another CDS entity, enter the relevant name as the *Referenced Object*.

   > **i Note**
   >
   > You can refer to another CDS entity or a database table and insert all elements by default.

5. Choose *Next*.
6. Assign a transport request.
7. Choose *Next*.
8. [Optional:] Select the code template to be inserted in the created DDL source code.

> **i Note**
>
> By default, ABAP Development Tools (ADT) considers the template for creation that you have selected at the last time. If you select the *Use the selected template* checkbox, ADT will always use the selected template by default and jumps over the template page at the next creation.
>
> ADT only displays the templates that can be used at this point of your implementation.
>
> You can create and use your own templates. For more information, see ABAP CDS Code Templates [page 22]

9. 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.

In the *Project Explorer*, the new data definition is added to the *Core Data Services* folder of the corresponding package node. As a result of this creation procedure, the source editor will be opened. Here, you can start defining a CDS entity.

In addition, you can start modifying the inserted code template and add annotations to the relevant elements and/or parameters.

The CDS entity is defined at activation of the data definition.

Related Information

CDS Views [page 11]
Editing DDL Source Code [page 44]

3.1.2 Creating ABAP CDS Objects With Reference to Other Objects

A referenced object is a repository object that relates to or is based on another development object.

Context

You want to create a CDS object that relates to another CDS entity or database table. This enables you, for example, to create a metadata extension that contains all elements of a particular data definition.

Procedure

1. In your ABAP project, select the relevant package node and data definition 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.

   The creation wizard is opened.
3. In addition to the *Project* and *Package*, enter the *Name* and the *Description* for the data definition to be created.
4. Enter or search the name of the *Referenced Object*.

   > **i** Note
   >
   > In the *Creation Wizard*, the label of this input fields depends on the type that you want to refer or base on your new CDS object. In the context of ABAP CDS development, you can use the reference functionality when creating the following CDS objects:
   > - **Data definitions**: You want to insert all elements from a referenced CDS entity or database table when creating a data definition. To do this, enter the name of the underlying *Referenced Object* in the *Creation Wizard*.
   > - **Access control**: You want to protect a CDS entity by creating an access control. To do this, enter the name of the underlying *Protected Entity*.
   > - **Metadata extension**: You want to extend the elements of a CDS entity by creating a metadata extension. To do this, enter the name of the underlying *Extended Entity*.
   > - **Service definition**: You want to expose a CDS entity by creating an access control. To do this, enter the name of the underlying *Exposed Entity*.
   >
   > If you select the referenced object in the *Project Explorer* and create the CDS object from the context menu, its name will be displayed automatically inserted.
   >
   > If you use the content assist, only the supported objects will be offered.
5. Choose *Next*.
6. Assign a transport request.
7. Choose *Next*.
8. [Optional:] If requested, select a template.

   > **i** Note
   >
   > ABAP Development Tools (ADT) only displays the templates in a table that are relevant for the current case.
   >
   > By default, ADT Tools considers the template for creation that you have selected at the last time.
9. Choose *Finish*.

Results

In the selected package, the ABAP back-end system creates an inactive version of a metadata extension and stores it in the ABAP Repository.

In the *Project Explorer*, the new metadata extension is added to the *Core Data Services* folder of the corresponding package node. As a result of this creation procedure, the source editor will be opened and contains the elements of the referenced data definition. Here, you can start defining a metadata extension.

After developing and checking your new object, you can activate it.

Related Information

Creating a Data Definition [page 33]
Creating Metadata Extensions [page 86]

3.1.3 Using ABAP CDS Code Templates for Data Definitions

Code templates can help to reduce the time spent on routine coding.

Use

ABAP Development Tools (ADT) provides a number of predefined code templates for data definitions. In addition, you can create further templates for your own use.

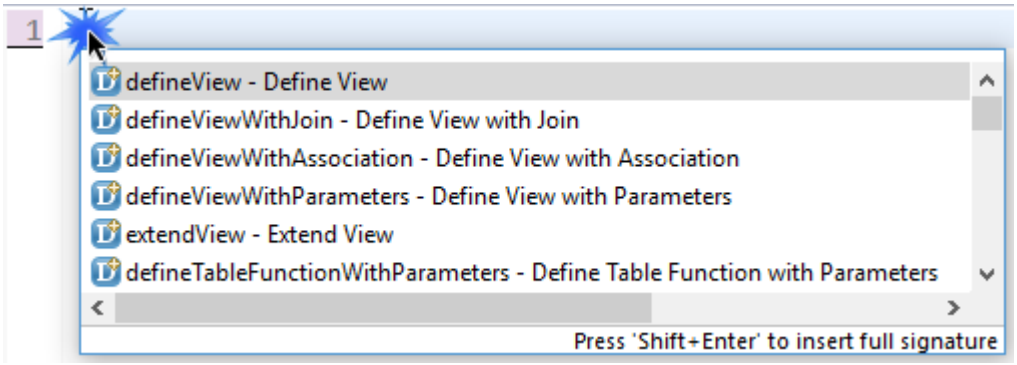
When creating a data definition, the creation wizard provides different templates to define different types of data definitions. Choose the relevant template.

To search for all templates available ...

1. Open the *Preferences* page ▶ *ABAP Development* > *Editors* > *Source Code Editors* > *Data Definition Templates* ▶ from the menu bar.

To use a template ...

1. In the empty DDL source code editor, trigger code completion (`Ctrl` + `Space`).
   A popup is opened which displays the available templates.



Example for displaying and selecting the available templates for data definitions

2. Select the relevant template to be inserted.

> → Tip
>
> Alternatively, you can use the *Templates* view to insert the code template in the DDL editor using **drag & drop**.

To create an additional template for your own use ...

1. Open either the *Templates* view or the *Preferences* page (referred above).
2. Open the *New Template* dialog.
3. Specify the *Name* and the *Description*, and edit the *Pattern* of the template.

   > → Tip
   >
   > If you want to add the new template to the data definition creation wizard, select *Data Definition (creation)* as the *Context*.
   >
   > To add variables to the template in the *Pattern* field, choose the *Insert Variable...* button.

4. Improve the format for the source code template manually.
5. Save the new template with *OK*.

Related Information

Creating and editing templates
Java Editor Template Variables

3.1.4 Activating Data Definitions

At activation of a data definition, the appropriate CDS entities and CDS objects are implicitly created in the ABAP Dictionary.

Prerequisites

Make sure that the data definitions you wish to activate is syntactically correct. The system performs a syntax check of the entire object before it is activated.

Context

You want to generate a CDS view.

Procedure

To trigger activation of data definitions, you have the following possibilities:

1. From the *DDL editor* ...
   a. Open the relevant data definition in the DDL editor.
   b. Choose the icon  (*Activate the ABAP Development Object*) in the toolbar.

   > → Tip
   >
   > Alternatively, you can use the shortcut `Ctrl` + `F3`.

2. From the *Project Explorer* ...
   a. Select the node of the relevant data definition in the ABAP project in the *Project Explorer*.
   b. Open the context menu and choose *Activate*.

Results

In the selected ABAP package, the back-end system creates an active version of the CDS view and the CDS database view and stores them in the ABAP Dictionary. The data definition is added to the *Core Data Services* folder of the selected package.

> i Note
>
> You can specify the client dependency of the CDS view using the `@ClientHandling` CDS annotation. If the CDS view is client-dependent, the client field is automatically added to the CDS database view.

Related Information

Displaying the ABAP Dictionary Log and the Activation Graph [page 42]
Previewing Data Records [page 78]

3.1.4.1 Displaying the ABAP Dictionary Log and the Activation Graph

When a data definition object is activated, the ABAP Dictionary log protocols all changes. The activation graph visualizes the information of the ABAP Dictionary log in a graphical way.

Context

You want to get further details about impact of your changes, for example, in case of errors at activation.

You can open the following views to get more information:

- *Dictionary Log* to get more detailed technical information on diagnosis and troubleshooting in a tabular way
- *Activation Graph* to visualize the order of activation from the ABAP Dictionary and CDS objects which are involved and possible effects for mass activation on depending CDS objects

You have the following possibilities to display further details:

1. For an **error item** displayed in the *Problems* view
2. For example, for an activation issue from the context menu in the *DDL editor*, select the corresponding function in the context menu:
   - To open the *Dictionary log*, choose *Open with* *Dictionary Log*.
   - To display the *Activation Graph*, choose *Open with* *Activation Graph*.

> **i Note**
>
> view: SelectBoth functionalities can also be triggered from a selected data definition in the *Project Explorer*.

3.1.4.2 Viewing Generated SQL Statements

You can display the SQL create statement that was generated at database level for each CDS entity in a popup.

Context

You want to realize some unexpected behavior which took place when you were accessing CDS entities for data selection. Then, you will possibly need to check the syntax of the native SQL statements generated at database level.

> ✤ Example
>
> You might be interested in checking ...
>
> - the generated `SQL` data types, based on their definition in the CDS table functions when running AMDP procedures.
> - the `JOIN` structure at database level when using associations in CDS view definitions.

> ℹ Note
>
> You can use the SQL statement viewer for active, inactive, and even unsaved data definitions – if the source code is syntactically correct. Otherwise, you receive a corresponding message `DDL Statement could not be created. Check data definition for syntax errors.`

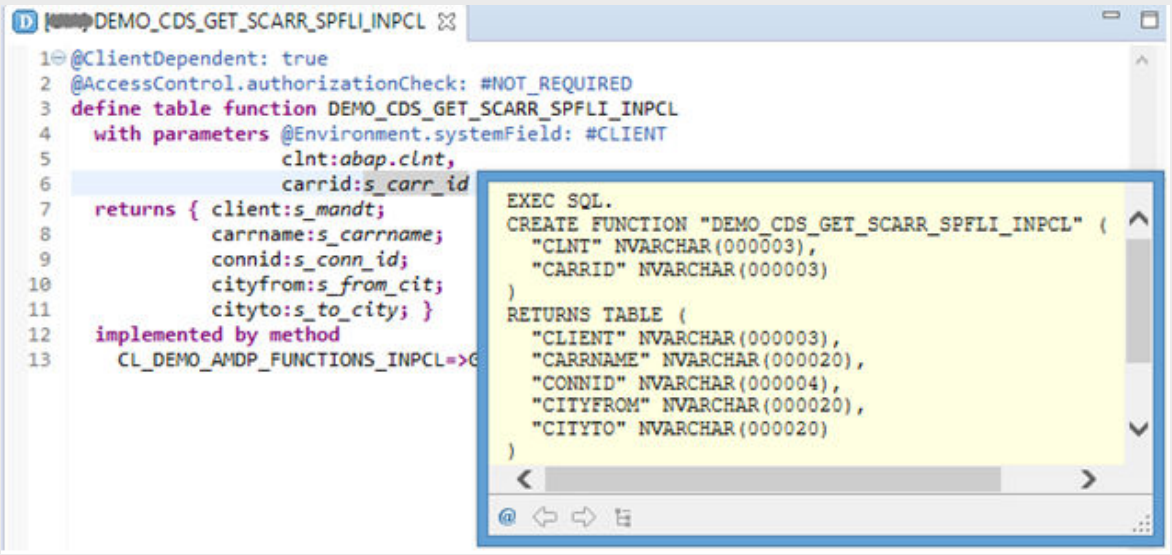
Procedure

1. Open the relevant CDS entity (CDS view, CDS table function) in the DDL source editor.
2. Position the cursor somewhere in the DDL source code.
3. Open the context menu and choose *Show SQL CREATE Statement*.

Results

The corresponding SQL create statement is displayed in the *Element Information Popup*.

Example



The SQL syntax for CREATE FUNCTION is displayed in the Element Information Popup

> **i Note**
>
> The SQL create statement depends on the CDS entity from which you trigger it. Consequently, the create statement might differ from the example above.

> **→ Tip**
>
> From within the *Element Information Popup*, you have the option to open the *ABAP Element Info view* by clicking the *Show in ABAP Element Info View* icon (@).

3.2 Editing DDL Source Code

> **i Note**
>
> In the current version of the DDL editor within ABAP Development Tools, you can only define one CDS entity in a data definition.

Related Information

Getting Support from the Content Assist [page 45]
Getting Help for DDL Source Code [page 51]
Displaying Details in the Element Information Popup and the ABAP Element Info View [page 54]

Navigating Associations [page 56]
Applying Quick Fixes [page 57]
Defining ON Conditions by Use of a Wizard [page 58]
Adding and Removing Comments [page 61]
Hiding CDS Annotations and Comments [page 61]
Changing Colors of DDL and DCL Source Code [page 62]
Comparing DDL Source Code Versions [page 63]
Formatting DDL [page 63]
Checking Syntax of DDL Source Code [page 73]

3.2.1 Getting Support from the Content Assist

In the *CDS source code editor*, the content assist proposes valid keywords and identifiers that can be inserted into the source code at the cursor position. Consequently, you can reduce the time spent on code editing and ensure that you are using the valid syntax and source code elements.

You can reduce the time spent on code editing and ensure that you are using the valid syntax and source code elements by using the content assist.

The following content assist functions are provided for CDS objects:

Content Assist	Description
Keyword Completion [page 46]	Proposed automatically as soon as you start typing to get the best matching keyword in the completion popup.
Code Completion [page 46]	Proposed at the position where you have triggered the `Ctrl` + `Space` shortcut with • All matching keywords or the • Semantic identifiers like the best matching data sources, parameters, or elements of an ABAP Dictionary artifact in the code completion popup.

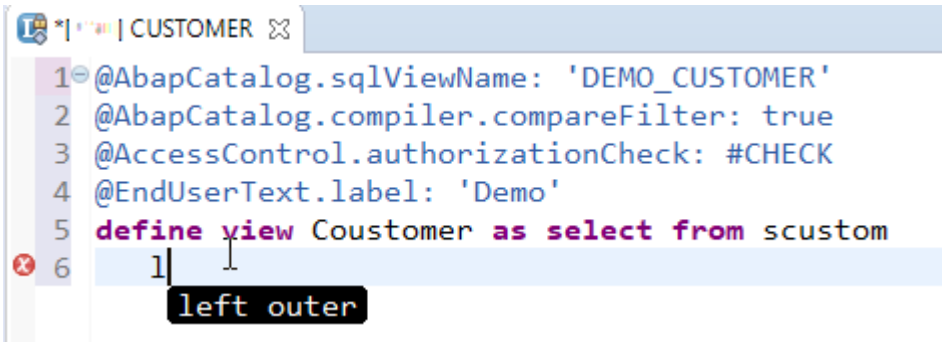
3.2.1.1 Keyword Completion

In the *DDL source code editor*, the best matching keyword is automatically displayed as soon as you start typing the keyword.

Procedure

1. In the DDL editor, start typing the keyword.

   The editor opens the completion popup and displays the most relevant keyword to be inserted.



Keyword completion in the DDL source code editor

2. To use the keyword in your DDL source code, choose the `tabulator` key.

3.2.1.2 Code Completion

In the *CDS source code editor*, you can trigger code completion manually at any position to get the list of the best matching keywords or identifiers.

Context

You want to get a list with proposals of keywords or identifiers that can be added at the current cursor position.

This functionality enables you to:

- Add possible keywords or identifiers such as fields, data elements, CDS annotations, and parameters that are typed with a data element or a predefined ABAP type

  > **i Note**
  >
  > You can use wildcards like the asterisk (*) to limit the list of relevant entries if you do not know the qualified name of the identifier.
  >
  > Using the asterisk at the leading position is not supported.

> If you use several segments within a identifier path, only the asterisk in the last segment is supported.

- Select and add the component of the related identifier, such as one or more fields from a data source (for example, a database table or a select from another CDS view)

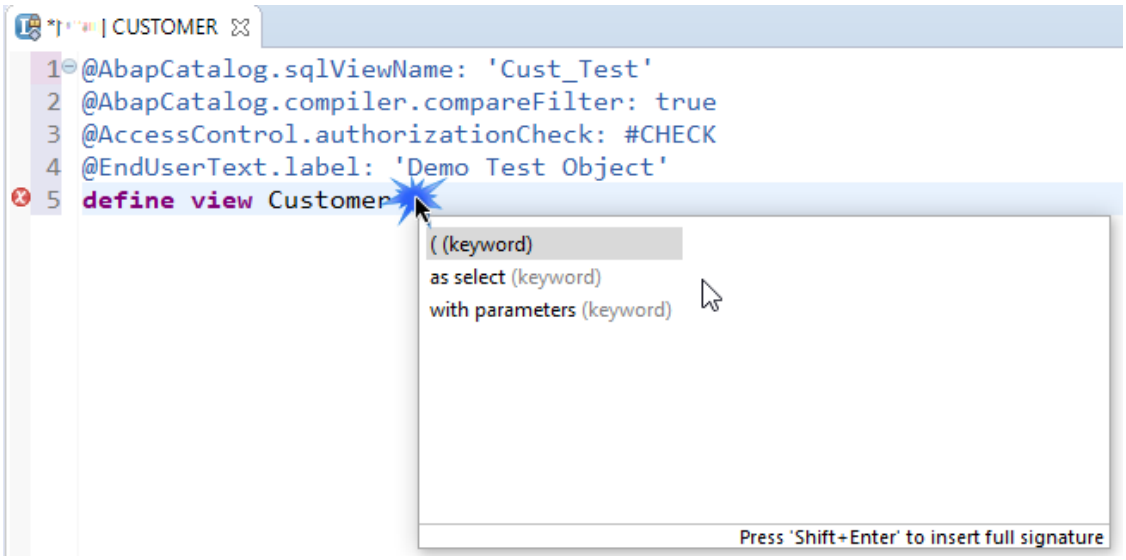
to your ABAP CDS source code.

Procedure

Code completion depends on the position from where you trigger it within the source code. You have the following possibilities to trigger code completion manually:

1. To trigger code completion for keywords, proceed as follows:
   a. At the position (for example, at the beginning of a row or after an identifier) where you want to add a keyword, type its first letter(s).
   b. Choose `Ctrl` + `Space`.

      The popup with the list of the relevant keywords will be opened.

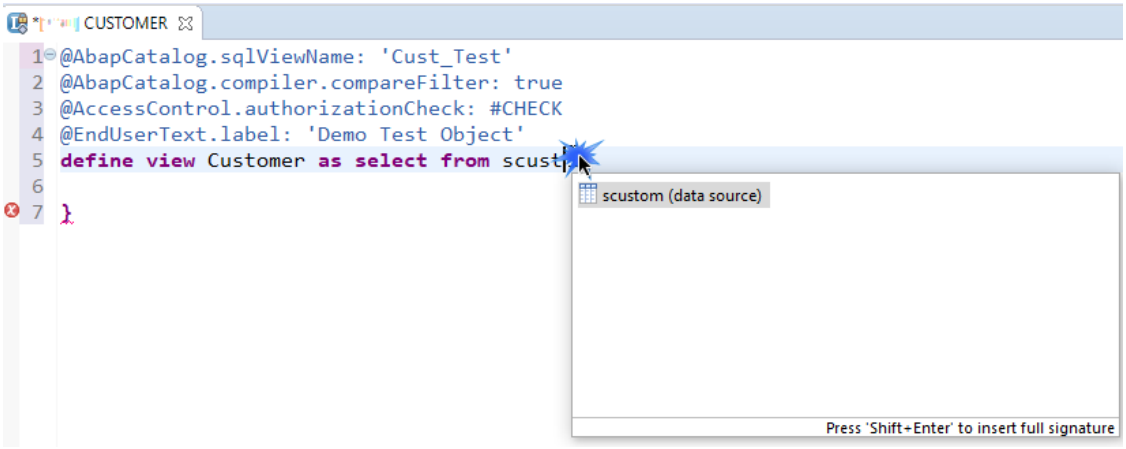


Example of a code completion popup for keywords

   c. Select the relevant keyword and choose `Enter`.

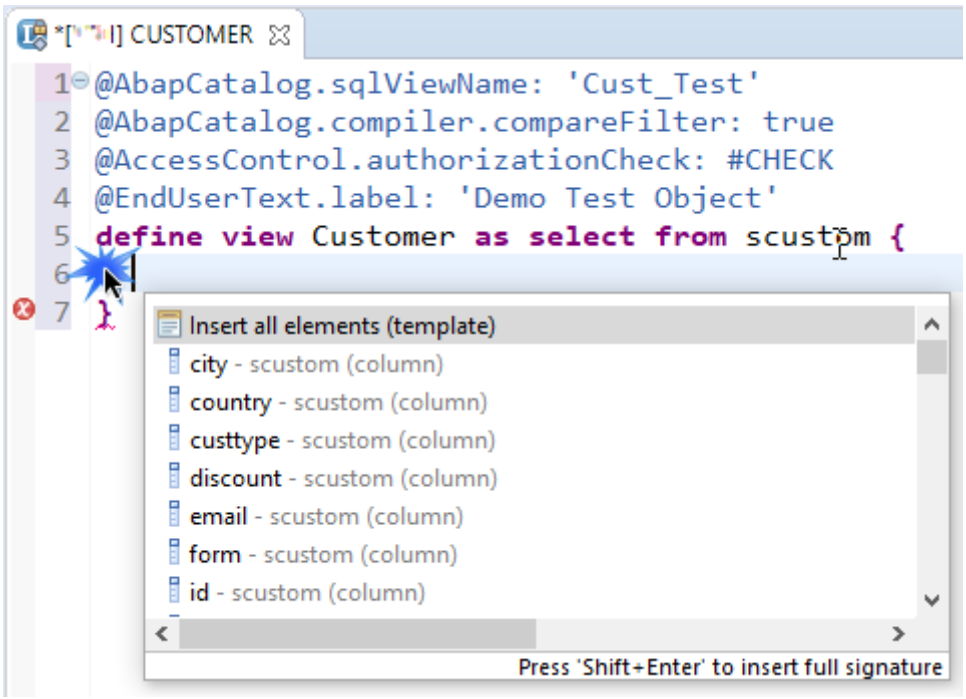
2. To trigger code completion for the name of an identifier, for example, the name of a data source and its parameters, proceed as follows:
   a. At the position where you want to add an identifier, type the first letter(s) of the name from the data source.
   b. Choose `Ctrl` + `Space`.

      The list with the elements of the signature is automatically displayed after you have typed a dot as the component selector. This enables you to add a component of a table to your CDS source code.



Example: Adding a data source to the DDL source code

The list with the elements of the signature is automatically displayed after you have typed a dot as the component selector. This enables you to add a component of a table to the DDL source code.



Completion of elements in the DDL source code editor

c. To add a keyword or an identifier, choose `Enter`. To add a parameter binding for a data source, choose `Shift` + `Enter`.

> **i Note**
>
> Code completion is context-sensitive and detects when parameter binding is required. It then also automatically inserts parameters when choosing a column or a CDS entity from the completion popup, also when you simply press `Enter`.

Related Information

Adding the Name of Data Sources to Elements as Prefix Automatically [page 49]

3.2.1.2.1 Adding the Name of Data Sources to Elements as Prefix Automatically

You can configure the code completion preferences for CDS entities to add the name of a data source as the prefix before an element automatically.

Procedure

1. Open the *ABAP Development* > *Editors* > *Source Code Editors* > *CDS* > *Code Completion* > *Preferences* page.
2. To always add the name of the relevant data source to the element as the prefix, choose the *Always prefix elements with data source name* checkbox in the preferences.
3. Choose *Apply* to add your changes and confirm with *OK*.

Results

If you now add the name of an element that is not unique and contained inside multiple data sources, ADT will automatically add the name of the relevant data source as the prefix. This makes it clear which element is used.

The names of the data source and the element are separated with a dot.

3.2.1.2.2 Adding Elements

You can add any CDS elements and CDS entities to CDS source code using code completion.

Context

The names of identifiers (such as CDS entities, fields, parameters, and associations) used in a CDS object are added in accordance with their definition. When adding an identifier using code completion, the names are added in the same case (camel case, lower case, and so on) in which they were originally defined in their original DDL source code. This ensures that identifier names are used consistently (same casing) in CDS sources.

Additionally, identifiers in CDS objects are always formatted on save, independent of the DDL formatter settings. This ensures consistent casing.

Related Information

Adding Aliases for Inserted Elements from Database Table [page 50]

3.2.1.2.2.1 Adding Aliases for Inserted Elements from Database Table

When creating a data definition, using a database table as datasource, and performing *Insert all elements* **or** when creating a data definition referenced to a database table, ABAP Development Tools (ADT) will generate an alias for each element in the select statement.

Context

You want to create a data definition that contains all elements of its datasource(s). An alias should be automatically generated for all elements of a database table.

> **i** Note
>
> Aliases will only be automatically generated in DDL source code if a database table is used as datasource. The aliases will then be generated for all element names as follows:
>
> - The first character of the element name will be changed in upper case.
> - The character following an underscore will be changed in upper case, while the underscore is removed.
>
> This behavior is enabled **by default** on the *ABAP Development* > *Editors* > *Source Code Editors* > *CDS* > *Code Completion* preference page. To deactivate it, deselect the *Always prefix elements with datasource name* checkbox.

Related Information

Adding Elements [page 49]

3.2.2 Getting Help for DDL Source Code

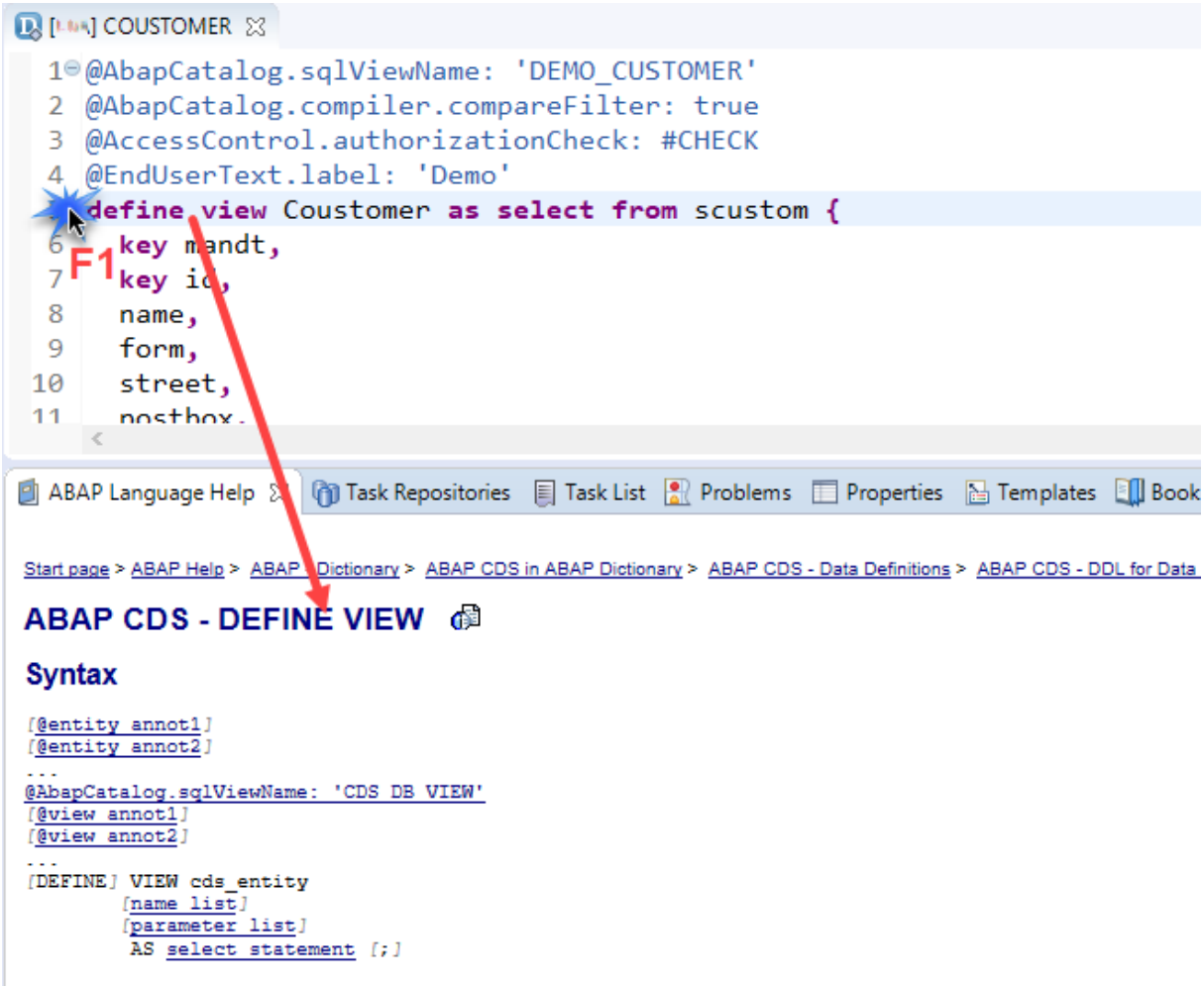
When editing DDL source code, you will generally need to make use of the ABAP CDS reference documentation. Is part of the ABAP Keyword Documentation. Here, you will find more information about the syntax and meaning of ABAP CDS language elements.

To get access to it, ABAP Development Tools provides a context-sensitive link (**F1 help**) from the ABAP CDS keywords within the DDL editor. This help content is displayed in a *ABAP Language Help* view.

In addition, you may need to display the definition and documentation of elements used, like database tables, views, or individual table fields that you used when defining CDS views in the DDL editor. For this purpose, choose F2 to access the *Element information popup*.

Accessing the ABAP Keyword Documentation (F1)

To access the ABAP Keyword documentation with contains the ABAP CDS language reference documentation, position the cursor on an ABAP CDS language element or SAP annotation for which you need help and choose F1.

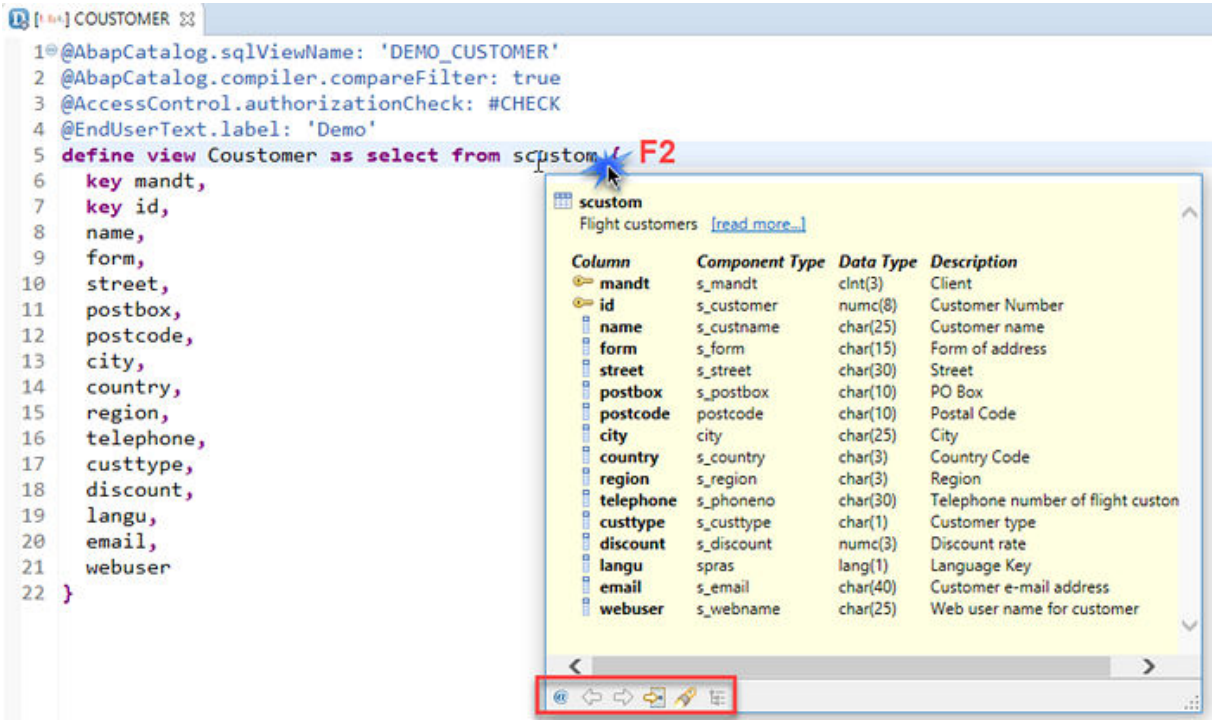


Context-sensitive language help in DDL editor

> → Tip
>
> In addition, you will find the complete ABAP CDS - Language Elements (ABAP Keyword Documentation) on the SAP Help Portal.

Accessing Element Information (F2)

In the DDL editor, move the cursor to the element of your interest and choose F2.

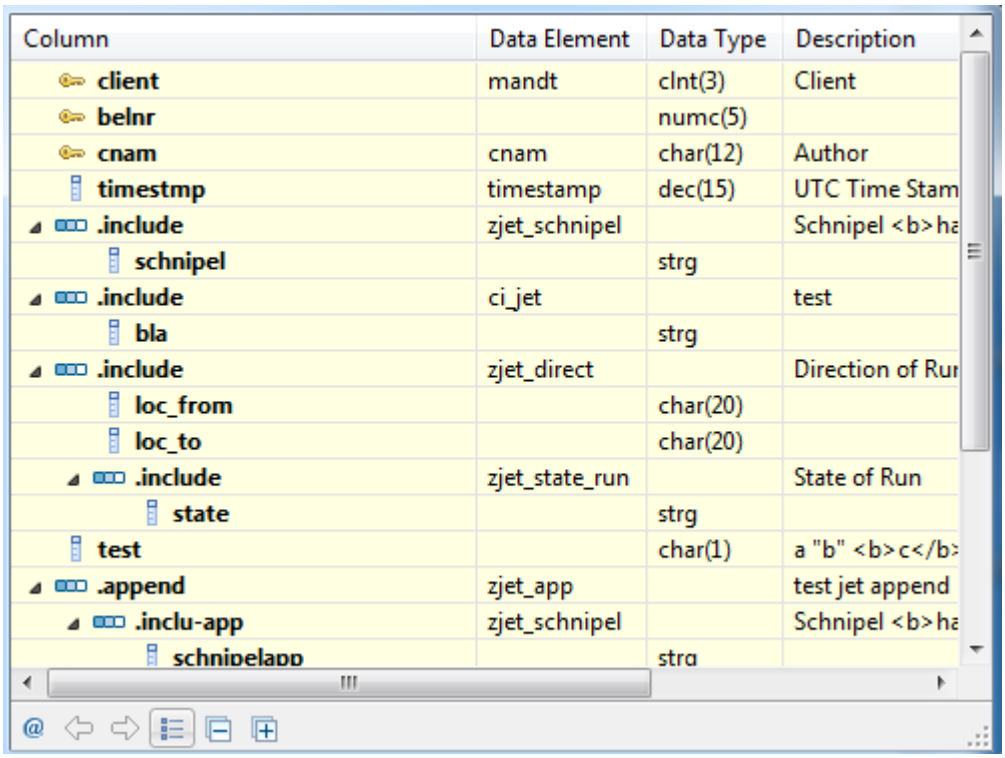


Opening the code Element information popup

From the toolbar within the *Element information popup*, you have the following options:

- To open the selected development object, choose the *Open in Editor (F3)* icon
- To search for elements within the*Element information popup*, choose the *Find (Ctrl+F)* icon
- To change the display mode, choose the *Detail view with include/append structure*
- To open the *ABAP Element Info* view, choose the "@" *Show in ABAP Element Info View* icon.

Example of an *ABAP Element Info* view:



Tree display for a table with appends/includes

> **i** Note
>
> From the table, you have the option to show a tree-based display that indicates the includes or append structures – if they are defined for the table fields.

3.2.3 Displaying Details in the Element Information Popup and the ABAP Element Info View

You can display details of ABAP Dictionary objects used in the context of CDS in the *Element Information* popup and the *ABAP Element Info* view.

Context

You want to display details about a CDS view used in a data definition to get further information.

Procedure

1. In the ABAP CDS source code, position the cursor on the CDS view name in the implementation and choose F2 or *Source Code* > *Show Code Element Information* from the context menu.

   The *Element Information* popup is opened and displays the corresponding details.
2. You can render the same information in the *ABAP Element Info* view.
   a. Choose the @ *"Show in ABAP Element Info view"* icon.
3. You can navigate in the *Element Information* popup / *ABAP Element Info* view, for example, to get more information about an element's type.
   a. Position the cursor on the name of the type in the corresponding element information.

      The type name becomes a link.
   b. To navigate, select the link.

      The element information for the selected type is now displayed.
4. You can open a development object in the relevant source code editor from the *Element Information* popup / *ABAP Element Info* view.
   a. Open the context menu from the relevant element and choose *Open in Editor (F3)*.

      Alternatively, choose the icon.
5. You can display information about the append structures that are generated when the CDS entity is extended with an extend view.
   a. Choose the *Detail view with include/append structure* icon.

      > **i Note**
      >
      > This icon is only available if the append structure can be evaluated.

      The elements of the data source and the extend view are displayed as a tree in the append structure popup.
6. You can browse within the *Element Info* popup / *Element Information* view.
   a. Choose one of the following shortcuts:
      - Alt + Left to navigate backward in the history
      - Alt + Right to navigate forward in the history
7. You can increase/reduce font size in the *Element Info* popup / *Element Information* view.
   a. Choose Ctrl + + / Ctrl + -.
8. You can search within the *Element Information* popup / *Element Information* view.
   a. Choose the find icon from the bottom or the toolbar.

      The search input field is opened.
   b. Enter the search string and choose *Enter*.

      The relevant search results are highlighted.
   c. To navigate within the search results, choose one of the following shortcuts, icons, or keys:
      - Enter or choose to find the next occurrence of your search string

- `Shift` + `Enter` or choose ⇧ to find the previous occurrence of your search string
- `F3` to open the selected search result in its editor
- `F2` to navigate to the selected search result within the element info

The cursor navigates to the relevant results.

d. To skim through search results, you can use the following icons:
   - To lock the current display in order to compare, for example, details of several data sources, choose the icon.
   - To link the display of the selected details with the current cursor position, choose the icon. The display will then automatically be refreshed in accordance to the selected cursor position within the source code.

Related Information

Element Information for CDS Views [page 24]

3.2.4 Navigating Associations

In addition to the `F3` Eclipse navigation functionality, you can also navigate between the origin, the definition, and the target of the associations you are using in your data definition.

You have the following options:

Navigating to the Association Origin

If an association originates in another data source or is defined within the same data source, you can navigate to its origin or definition as follows:

1. In the DDL editor, select the name of the association.
2. Open the context menu on the name and chose *Navigate To*.

> **i Note**
>
> Alternatively, you can press `F3` or `Ctrl` + `click`. A small dialog is then opened below the association name. Here you choose *Navigate To*.

The corresponding data definition id is opened and highlights the relevant origin or navigates to the relevant definition within the same data definition. In both cases, the selected occurrences and names are highlighted.

Navigating to Association Targets

You can navigate to the origin data source of the association's target – for example, another CDS entity or a database table – as follows:

1. In the DDL editor, select the name of the association.
2. Open the context menu on the name and chose *Navigate To Target*.

> **i Note**
>
> Alternatively, you can press `Ctrl` + `click`. A small dialog is then opened below the association name. Here, you then choose *Navigate To Target*.

The target of the association is opened in the appropriate DDL editor.

3.2.5 Applying Quick Fixes

Quick fixes enable you to resolve errors and warnings in the DDL source code using the corresponding functionality that is provided in the *Quick Fix* popup.

In the DDL editor, the following quick fixes are provided:

- Name of CDS database view is missing
  Effect: Adds the annotation `@AbapCatalog.sqlViewName: 'SQL_VIEW_NAME'` , which specifies the name of the CDS database view to be generated in the ABAP Dictionary.
- Name of CDS database view append is missing
  Effect: Adds the annotation `@AbapCatalog.sqlViewAppendName: 'APPEND_VIEW_NAME'`, which specifies the name of the append view that will be generated in the ABAP Dictionary.
- `GROUP BY` clause is missing
  Effect: Adds the `GROUP BY` clause that is required if aggregate functions (`MAX`, `SUM`, ...) are contained in the `SELECT` list. In addition, all elements not defined using aggregate functions are specified after `GROUP BY`.
- `EXTEND GROUP BY` clause is missing
  Effect: If the `GROUP BY` clause is already added, you can extend this group to include all elements not defined using aggregate functions and not yet part of the `GROUP BY` clause. The latter elements, when included, are also added to the existing `GROUP BY` clause.
- Alias is missing
  Effect: Adds an alternative name (alias) for each aggregate function that is used as an element in the `SELECT` list.

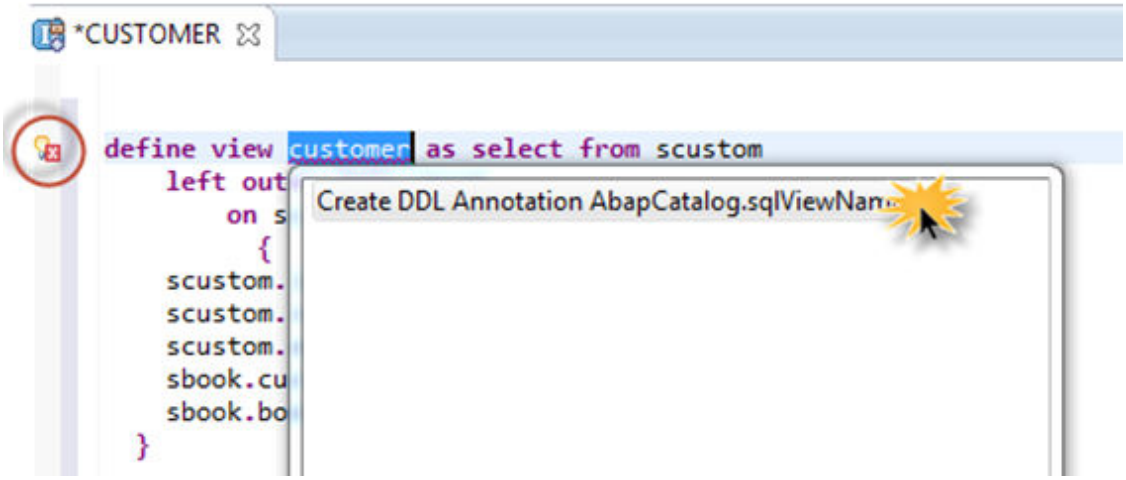
To execute a quick fix...

You can trigger quick fixes in any of the following ways:

1. From the *Problems* view:
   1. Click on the error to open the context menu.

2. Choose *Quickfix*.
3. In the list of possible quick fixes, double-click the relevant quick fix function.

2. From the ruler bar in the DDL editor:
   1. Click the error decorator in the ruler bar.
   2. In the list of possible quick fixes, double-click the relevant quick fix function.



Adding annotation for missing CDS database view name

3. From the *Quick Assist* view:
   1. In the DDL editor, position the cursor where the error occurs and is highlighted.
   2. In the *Proposals* tree, double-click the relevant quick fix function.

3.2.6 Defining ON Conditions by Use of a Wizard

A wizard helps you to define the `ON` conditions in the `JOIN` clauses and `ASSOCIATION` definitions of your CDS view definition.

Prerequisites

This function affects the CDS view in the currently opened editor.

Context

You can use the `ON` conditions wizard when defining:

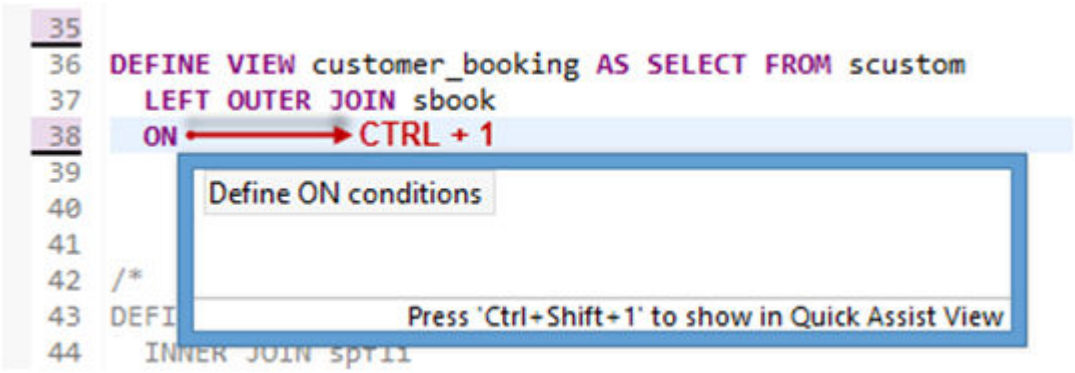
- Joins between two data sources (`JOIN` condition) in a CDS view.
- Conditions for association in a `SELECT` statement.

The wizard considers the used data sources and their elements and provides proposals (strategies) for joining the data sources. As developer you can also specify the elements for a `JOIN` manually using drag and drop (*User Defined* strategy).

To launch the wizard, proceed as follows:

Procedure

1. Open the relevant CDS view in the DDL editor.
2. In the CDS source code, position the cursor on the `ON` keyword where you want to define an `ON` condition for a join or association.
3. Press `CTRL` + `1` key shortcut.



Starting the ON conditions wizard from the quick fix view

A quick fix view appears.

4. On the quick fix view, double-click the *Define ON conditions* entry to start the wizard.

> **i Note**
>
> If the `ON` condition already exists, the mapping between elements of data sources will be displayed on the wizard page, where you can then modify them.

5. To specify the condition expression, define the mapping between the source and the target data source.

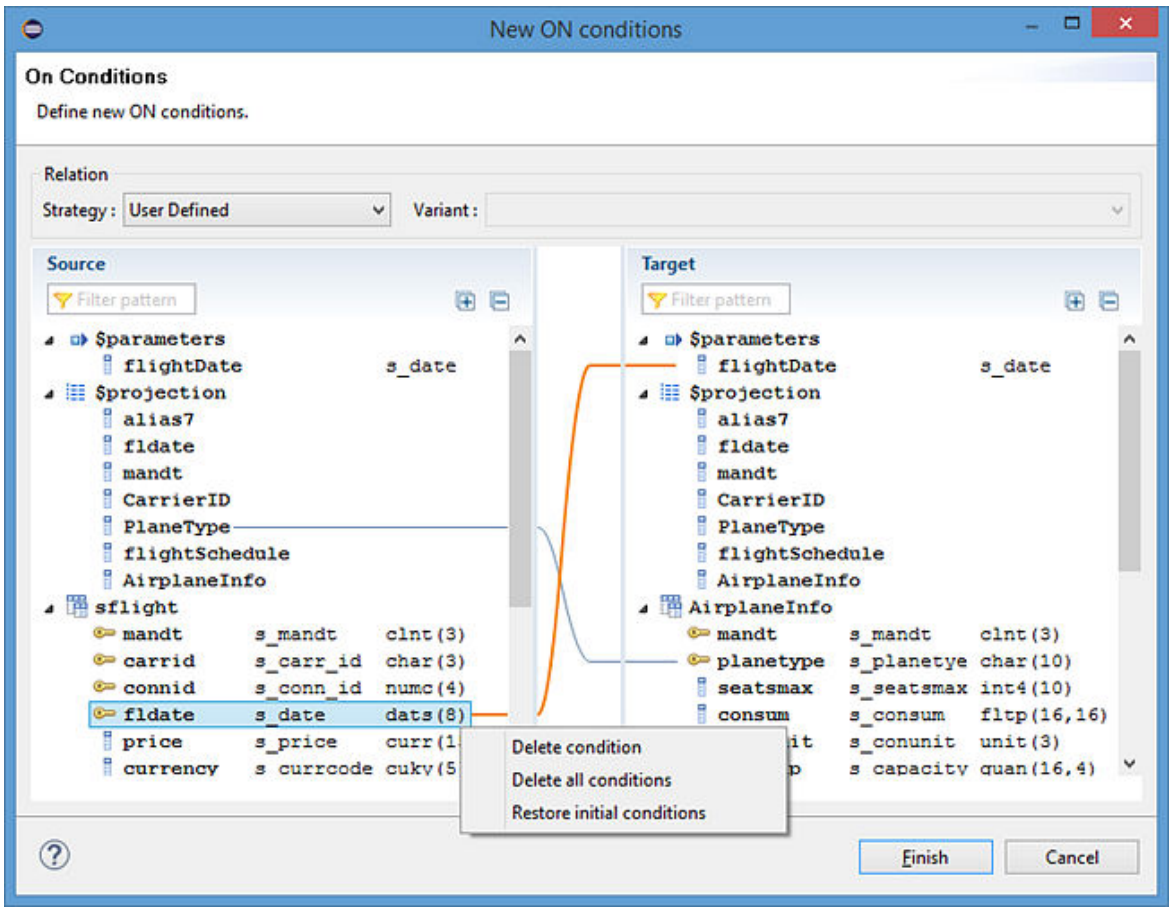
   The wizard offers you automatic proposals for mapping elements of the data sources:
   - *Strategy > By Name*: Elements with identical names are mapped to each other.
   - *Strategy > By Foreign Key*: Based on the foreign key relationship, that has been defined for a table in the ABAP Dictionary, mappings are suggested for each key definition.

   When using the *User Defined* option, you can map the elements in a straightforward manner using drag & drop.

   In addition, the wizard provides you with additional functionality, so that you can:
   - Filter the data sources and their elements by
     - Name
     - Dictionary type
     - Built-in type
   - Expand or collapse all elements of the
     - `$parameter`
     - `$projection`

- data source
- Access context menu functions to...
  - Delete a selected condition
  - Delete all conditions
  - Restore the initial mapping

> **→ Tip**
>
> To open the context menu on the wizard page, select the relevant element from the *Source* or *Target* area.



For defining mapping drag and drop relevant elements from source to target

6. Choose *Finish*.

Results

The wizard inserts the corresponding condition expression after the `ON` keyword into the DDL source code.

> **i Note**
>
> The wizard does not check the syntax or semantic correctness of a CDS view. So, the modified coding might contain syntax or semantic errors.

Related Information

ABAP CDS – SELECT, Association (ABAP Keyword Documentation)
ABAP CDS – SELECT, JOIN (ABAP Keyword Documentation)

3.2.7 Adding and Removing Comments

Within DDL source code, double slashes (`//`) introduce a comment that continues until the end of the line.

You can also use the following shortcuts for comments in the DDL editor:

Shortcut	Menu Entry	Description
Ctrl + <	Add Comment	The editor inserts a double slash (`//`) at the beginning of each selected line.
Ctrl + >	Remove Comment	The editor removes an existing double slash (`//`) at the beginning of each selected line.
Ctrl + 7	Toggle Comment	The editor inserts or removes a double slash asterisk (`//`) at the beginning of each selected line.

> **i Note**
>
> Alternatively, you can access the comments functions using the context menu entries of the DDL source editor (*Source* > *Add Comment...*)

3.2.8 Hiding CDS Annotations and Comments

In the source code of a CDS object (such as data definitions, metadata extensions, and access controls), you can hide CDS annotations and comments to improve readability to get a better visual overview of your coding.

Context

You want to get a clear overview, for example, of the fields in a `define view` statement from which you can select data.

To do this, proceed as follows:

Procedure

1. Open the context menu from the ruler in the DDL source editor.
2. Choose *Hide Annotations (Ctrl+Alt+F6)* to mask CDS annotations or *Hide Comments (Ctrl+Alt+F7)* to mask comments from the source code.

Results

The rows with the comments and annotations disappear from the DDL editor but the row numbering remains as before execution.

Note that the content is not deleted and still available for your CDS object. To display hidden content again, choose the relevant entry that is highlighted with a tick in the same context menu.

> **i** Note
>
> In addition, you can also use the following functionalities to gain a better overview and improve readability of your source code:
>
> - Position the cursor on the entity name and press `F2` in order to open the *Code Element Info* view.
> - Extract the relevant annotations from the `select list` elements to a metadata extension in order to relocate them.

Related Information

Extracting CDS Annotations to a Metadata Extension [page 87]

3.2.9 Changing Colors of DDL and DCL Source Code

The source code editor displays the DDL and DCL source code and annotation types with predefined colors. However, you can change these default settings to adapt them to your personal needs.

Context

You can define the color for displaying the keywords, identifiers, annotation, or further code elements in DDL and DCL source code.

Procedure

1. Open the *General* > *Appearance* > *Colors and Fonts* preference page.
2. Expand the *CDS* folder.
3. Select the text type you want to change and click the *Edit...* button.
4. Select the color and confirm with *OK*.

> → Remember
>
> To restore the default color settings, click the *Restore Defaults* button.

Results

The new color settings become immediately effective in the corresponding editor.

3.2.10 Comparing DDL Source Code Versions

The following options are available for comparing DDL source code:

- Comparing local changes using the local change history
- Comparing changes from one transported version to another
- Comparing changes across the ABAP systems.

For details, see also:

3.2.11 Formatting DDL

You format DDL source code (such as a CDS statement, including the element list, Boolean expressions, JOINs, association definitions, and so on) to structure the code and to improve its readability.

In the context of editing DDL source code, the following use cases are supported:

1. **Using the SAP standard profile**: You want to use the standard profile predefined by SAP. This profile is provided by default and can be used immediately.

   > i Note
   >
   > The SAP standard profile is persisted in the back end. This ensures that the formatting result is always consistent – independently of the installed ADT client version.

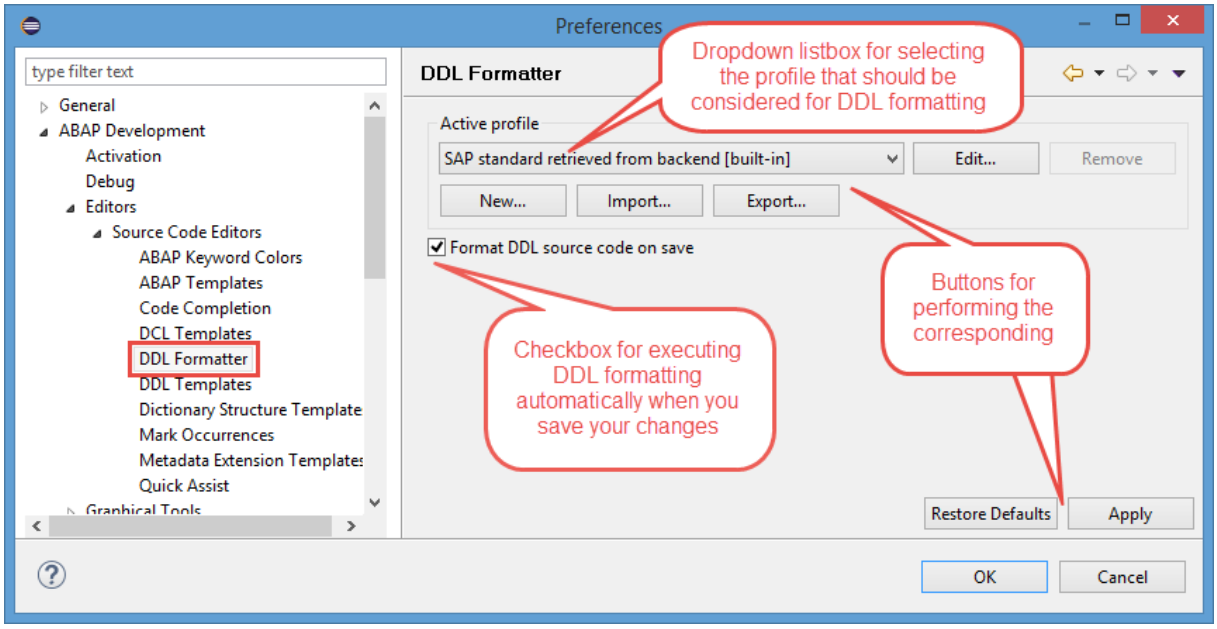
2. **Creating your own profile**: If the SAP standard profile does not meet your needs, you can create and use your own profile. In addition, you can import or export a profile to reuse/share it from/with other ADT developers.

> **i Note**
>
> The local configuration is persisted in the Eclipse-based IDE workspace.

3. **Overruling the SAP standard profile for your team or company**: If a team decides to use its own profile, they can export a profile and make it available for all data definitions of an ABAP package.

> **i Note**
>
> A team profile is persisted in the back end. It does not need to be enabled. On the basis of the ABAP package, the DDL formatter detects whether the SAP standard profile or the team profile is to be used.

The DDL formatting settings are configured in the *ABAP Development* preferences.



Example of the first Preferences page for configuring the DDL formatter

Starting from the *ABAP Development* > *Editors* > *Source Code Editors* > *DDL Formatter* > *Preferences* page, you have the following general options:

- DDL formatting is not automatically executed by default. To perform this, select the *Format DDL on save* checkbox in advance.
- DDL formatting is configured through profiles. Thus, you can:
  - Select the profile you want to work with on your local IDE.
  - *Create* your own profile(s), *Edit* it (them) at a later point in time, and *Remove* it (them).
  - Work with SAP 's standard profile where the settings of the DDL formatter are already pre-configured.
  - *Import/Export* the profile as an XML file from/to another local IDE.
  - Overrule the SAP standard profile at package level.
- You can *Restore Defaults* to cancel your changes and revert to the default provided by SAP.

To execute the DDL formatter, use the context menu *Source Code* > *Format* or the `Shift` + `F1` shortcut at every position. If you have selected the *Format DDL on save* checkbox on the *DDL Formatter* preferences page, formatting is automatically performed when you save your changes.

Related Information

Creating a Profile [page 65]
Editing Profiles [page 70]
Importing Local Profiles [page 71]
Exporting Local Profiles [page 72]

3.2.11.1 Creating a Profile

You can create a profile to configure your own client-specific formatting of data definitions or to provide your profile to a team of ABAP developers using ADT.

Procedure

1. Open the *ABAP Development* > *Editors* > *Source Code Editors* > *DDL Formatter* preference page.

   The *DDL Formatter* preferences page is opened.
2. In the *Active profile* section, choose the *New...* button.

   The *New Profile* dialog is opened.
3. Enter the **Name** of the profile to be created.
4. [Optional:] To reuse the configuration of another profile, select the relevant one from the *Initialize settings with the following profile:* dropdown listbox.
5. [Optional:] To configure your profile at a later point of time, deselect the *Open the edit dialog now* checkbox.
6. Choose *OK* to continue.

   The *Profile* page is opened. The following tabs are provided for configuring the data definition formatting for:

   - *Indentation*: add a predefined number of spaces at the beginning of a statement, clause, or expression
   - *Boolean Expressions*: add line breaks and spaces before/after the `AND` / `OR` keywords
   - *Entity Name*: add line breaks and spaces before/after entity names
   - *Data Source*: add line breaks and spaces before/after data sources
   - *Joins*: define the alignment, line breaks, and spaces before/after joins
   - *Element List*: add line breaks and spaces before/after element lists in general
   - *Element List Entries*: add spaces before/after element lists entries in general
   - *Association Definitions*: align association definitions
   - *WHERE and HAVING Clause*: align occurrences of `WHERE` / `HAVING` keywords

In these tabs, you can define the following settings:

Name	Values	Description	Source Code Example Before Execution (Default)	Source Code Example After Execution

Alignment	○ Same line
	○ Separate line
	○ Break after AND / OR
	○ Break before AND / OR
		To handle the alignment at/before/after a specific occurrence – for example, an entity name	Alignment = Same line
			`...`
			`define view sample_define_view with parameters p_carrid: s_carrid ...`
				Alignment = Separate line
				`...`
				`define view sample_define_view with parameters p_carrid: s_carrid ...`

Align in columns	○ Enabled
	○ Disabled
		To align keywords or identifiers in the same column over several rows	Alignment in columns = Enabled
			`...`
			`define view sample_define_view with parameters p_carrid: s_carrid as select from scarr as carriers left outer join spfli on spfli.carrid = carriers.carrid and spfli.fltype <> 'X' ...`
				Alignment in columns = Disabled
				`...`
				`define view sample_define_view with parameters p_carrid: s_carrid as select from scarr as carriers left outer join spfli on spfli.carrid = carriers.carrid and spfli.fltype <> 'X' ...`

Name	Values	Description	Source Code Example Before Execution (Default)	Source Code Example After Execution

Delimiter positions	○ Comma / Semicolon at the end
	○ Comma / Semicolon at the beginning
		To add a line break before or after the occurrence of a comma/semicolon	Delimiter positions = Comma/semicolon at the end
			`...`
			`{`
			`  key carriers.carrname as carrname,`
			`@EndUserText.label: 'Flight No.'`
			`@EndUserText.quickInfo: 'Flight No.'`
			`  key spfli.connid as fid,`
			`@EndUserText.label: 'Date'`
			`sflight.fldate as fldate,`
			`@EndUserText.label: 'From'`
			`apfrom.name as airpfrom`
			`}`
			`...`
				Delimiter positions = Comma/semicolon at the beginning
				`...`
				`{`
				`  key carriers.carrname as carrname`
				`  ,@EndUserText.label: 'Flight No.'`
				`@EndUserText.quickInfo: 'Flight No.'`
				`  key spfli.connid as fid`
				`  ,@EndUserText.label: 'Date'`
				`sflight.fldate as fldate`
				`  ,@EndUserText.label: 'From'`
				`apfrom.name as airpfrom`
				`}`
				`...`

Name	Values	Description	Source Code Example Before Execution (Default)	Source Code Example After Execution
Existing blank lines	○ Leave as is ○ Remove	To keep or remove empty rows	Existing blank lines = Leave as is `@AbapCatalog .sqlViewName : 'S_SAMPLE_1' @AbapCatalog .compiler.co mpareFilter: true @AccessContr ol.authoriza tionCheck: #NOT_REQUIRE D @EndUserText .label: 'Sample: Define View Statement' define view sample_defin e_view ...`	Existing blank lines = Remove `@AbapCatalog .sqlViewName : 'S_SAMPLE_1' @AbapCatalog .compiler.co mpareFilter: true @AccessContr ol.authoriza tionCheck: #NOT_REQUIRE D @EndUserText .label: 'Sample: Define View Statement' define view sample_defin e_view ...`
Indentation	○ Indent ○ Indent based on join hierarchy ○ None	To add the predefined number of spaces at the beginning of the next line – for example, after an entity name	Indentation = Indent `... define view sample_defin e_view with parameters p_carrid: s_carrid as select from scarr as carriers left outer join spfli on spfli.carrid = carriers.car rid and spfli.fltype <> 'X' ...`	Indentation = None `... define view sample_defin e_view with parameters p_carrid: s_carrid as select from scarr as carriers left outer join spfli on spfli.carrid = carriers.car rid and spfli.fltype <> 'X' ...`

Name	Values	Description	Source Code Example Before Execution (Default)	Source Code Example After Execution

Indentation size	Input field	To predefine the number of spaces that should be added when starting the subsequent line/ block	Indentation spaces = 2
			`...`
			`define view sample_define_view`
			`  with parameters`
			`p_carrid: s_carrid`
			`  as select from scarr    as carriers`
			`    left outer join spfli`
			`       on spfli.carrid = carriers.carrid`
			`   and spfli.fltype <> 'X'`
			`...`
				Indentation spaces = 4
				`...`
				`define view sample_define_view`
				`    with parameters`
				`p_carrid: s_carrid`
				`    as select from scarr    as carriers`
				`left outer join spfli`
				`       on spfli.carrid = carriers.carrid`
				`       and spfli.fltype <> 'X'`
				`...`

> **i** Note
>
> Some of the settings might depend on each other. To make them available, you will need to adjust another setting in advance.

7. [Optional:] A preview enables you to check the effect of your current setting in your source code. Select the relevant setting to get an impact of its consequence in the source code.

   In the left screen, you can see the current formatting. In the right screen, a preview displays the result of your currently selected formatting.
8. Choose *OK* to save your configuration.
9. Choose *Apply* and *OK* to make your profile available.

Results

The configured profile can now be used in the data definition source editor.

> **i Note**
>
> If the data definition source code is not formatted as defined, check if the relevant profile is selected on the *DDL Formatter* preference page.

Related Information

Editing Profiles [page 70]

3.2.11.2 Editing Profiles

You can edit profiles to update a configuration used by the DDL formatter.

Prerequisites

You have created or imported a profile.

Procedure

1. Open the *ABAP Development* > *Editors* > *Source Code Editors* > *DDL Formatter* preference page.

   The *DDL Formatter* preference page is opened.
2. In the *Active profile* section, select the relevant profile from the dropdown listbox of .
3. Choose the relevant *Edit...* button in the same section.

   If your IDE contains several ABAP projects, the *Project Selection* dialog is opened. Choose the relevant project.

   The *Profile* page is opened.
4. In the corresponding tab(s), edit the relevant settings.

   > **i Note**
   >
   > For further information about the displayed settings, see the Related Information section below.
5. Choose *OK* to add your changes.
6. Choose *Apply* and then press *OK* to complete your changes.

Results

Your changes are saved and will be considered the next time the DDL formatter is executed.

Related Information

Creating a Profile [page 65]

3.2.11.3 Importing Local Profiles

You can import another profile for the purpose of reusing a DDL formatting.

Prerequisites

To import another profile, you need the corresponding XML file available on the file system.

Procedure

1. Open the *ABAP Development* > *Editors* > *Source Code Editors* > *DDL Formatter* preference page.

   The *DDL Formatter* preference page is opened.
2. In the *Active profile* area, choose the *Import...* button.

   The *Open* dialog is opened.
3. Select the relevant file location and file.
4. Choose *Open*.
5. To start importing, choose *OK*.

Results

After the import has finished, the imported profile will be available in the dropdown listbox and automatically selected in the *Active profile* section of the *DDL Formatter* preference page.

The *DDL Formatter* preference page is closed.

Related Information

Exporting Local Profiles [page 72]

3.2.11.4 Exporting Local Profiles

You can export your profile to provide this DDL formatting to other ADT developers.

Procedure

1. Open the *ABAP Development* > *Editors* > *Source Code Editors* > *DDL Formatter* preference page.

   The *DDL Formatter* preference page is opened.
2. In the *Active profile* area, choose the *Export...* button.

   The *Save as* dialog is opened.
3. Select the relevant file location and enter the file name.

   When you save the profile, the XML file name is proposed in accordance with the profile name to be exported.
4. Choose *Save*.
5. To start exporting, choose *OK*.

Results

The XML file is created and saved in the selected file location.

The *DDL Formatter* preference page is closed.

If you want to overrule the SAP standard profile at package level with your exported profile, you can copy its XML content to the BAdI implementation class.

Related Information

Importing Local Profiles [page 71]

3.2.12 Checking Syntax of DDL Source Code

Context

With the editor check function, you can check whether the source code of DDL source code is syntactically correct or not.

To enable the check function, there are two options available:

- **Automatic** syntax check: This option is enabled by default for all source code based editors.

  > **i Note**
  >
  > If you wish to disable the automatic syntax check, you have to switch off the corresponding setting in the preferences: *ABAP Development* > *Editors* > *Source Code Editors* > *Automatic syntax check*.

- **Explicit** syntax check: You can use this option whenever the automatic syntax check is disabled.

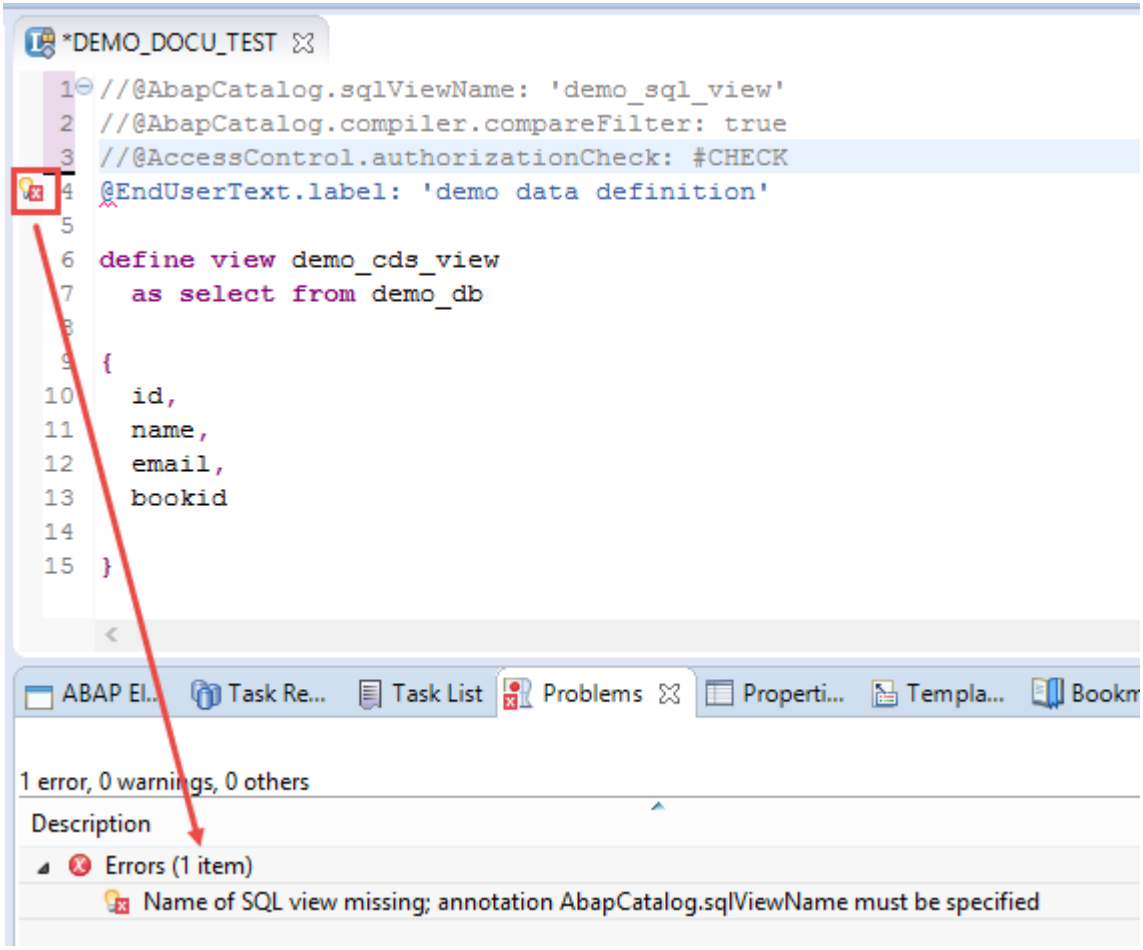
To trigger an explicit syntax check, proceed as follows:

Procedure

1. Open the editor with the relevant DDL source code.
2. Click the icon  (*Check ABAP Development Object*) in the toolbar. Alternatively, you can use the keyboard shortcut `Ctrl` + `F2`.

Results

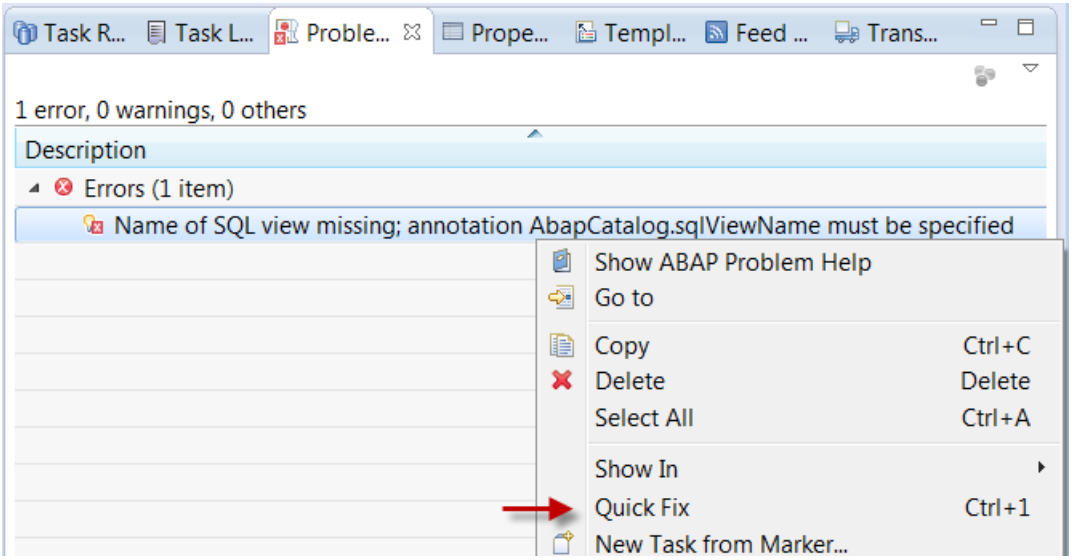
If errors occur during the check, these will be issued to the *Problems* view and displayed there as **ABAP Syntax Check Problem**. In addition, the code line with the error is labeled with a decorator in the DDL source editor.



Displaying syntax error in the DDL source

→ Tip

The DDL editor offers corrections to some problems found. In case of missing annotation, for example, you can take advantage from the **quick fix** for that specific error item.



Accessing quick fix in the Problems view using the context menu of the error item

Related Information

Activating Data Definitions [page 41]
Previewing Data Records [page 78]

3.3 Accessing Data Models

3.3.1 Adding Access Controls to CDS Entities

Prerequisites

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

Context

Use access controls to develop access-control logic for Core Data Services (CDS) entities from AS ABAP. Access controls enable you to filter access to data in the database based on static values or conditions based on user data (classical authorization objects). Use data control language (DCL) to write access controls. If no access control was created and deployed for the CDS entity, a user who can access the CDS entity has access to all data returned by the entity.

> **Example**
>
> For example, you provide a view of sales orders. You can add a condition that users can only view open sales orders or only sales orders for companies, which are in the countries that are listed in a classical authorization object.

> **→ Tip**
>
> We recommend that you protect applications that use CDS entities with classic start authorizations available from AS ABAP.

Procedure

1. Create the access control development object.
2. Edit the source code of the access control.

   Edit the source code of the access control just as you would data definition source code.

   There is no support for the following:

   - Quick fixes
   - Adding and removing comments
   - Comparing source code versions of access controls
3. Check the syntax of the access control.

   Check the syntax of access controls just as you would check the syntax of data definitions.
4. Activate the access control.

   Activate access controls just as you would activate data definitions.

   Activation logs for access controls are not supported.

Related Information

Creating Access Controls [page 77]
Editing DDL Source Code [page 44]
Checking Syntax of DDL Source Code [page 73]
Activating Data Definitions [page 41]
Access Controls [page 32]

3.3.1.1 Creating Access Controls

An access control enables you to limit the results returned by a CDS entity to those results you authorize a user to see.

Prerequisites

- You have the standard developer authorization profile to create development objects.
- You have created the CDS entities for which you want to restrict access.

Context

An access control is a development object, which supports standard functions such as transport, syntax check, and activation.

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* > *Access Control*.
3. In addition to the *Project* and *Package*, enter the *Name* and the *Description* for the access control to be created.
4. [Optional:] If you want to create an access control for a specific CDS entity, enter the relevant entity name as the *Protected Entity*.

   > **i Note**
   >
   > You can only protect CDS entities.

5. Choose *Next*.
6. Assign a transport request.
7. Choose *Next*.
8. Determine if you want to use a template for the access control.

Option	Description
Use a template.	SAP offers example templates for access controls. The template provides you with example coding for you to modify.
Do not use a template.	The tool creates an empty access control for you to code.

9. Choose *Finish*.

Results

In the selected package, AS ABAP creates an inactive version of an access control and stores it in the ABAP Repository. In the *Project Explorer*, the new access control is added to the *Access Controls* folder of the corresponding package node. As a result of this procedure, the access control editor opens. Define the role for the CDS entity.

Related Information

Access Controls [page 32]
Editing DDL Source Code [page 44]

3.4 Previewing Data Records

Data Preview provides a test environment that enables you to verify the output (result set) of a CDS view.

Procedure

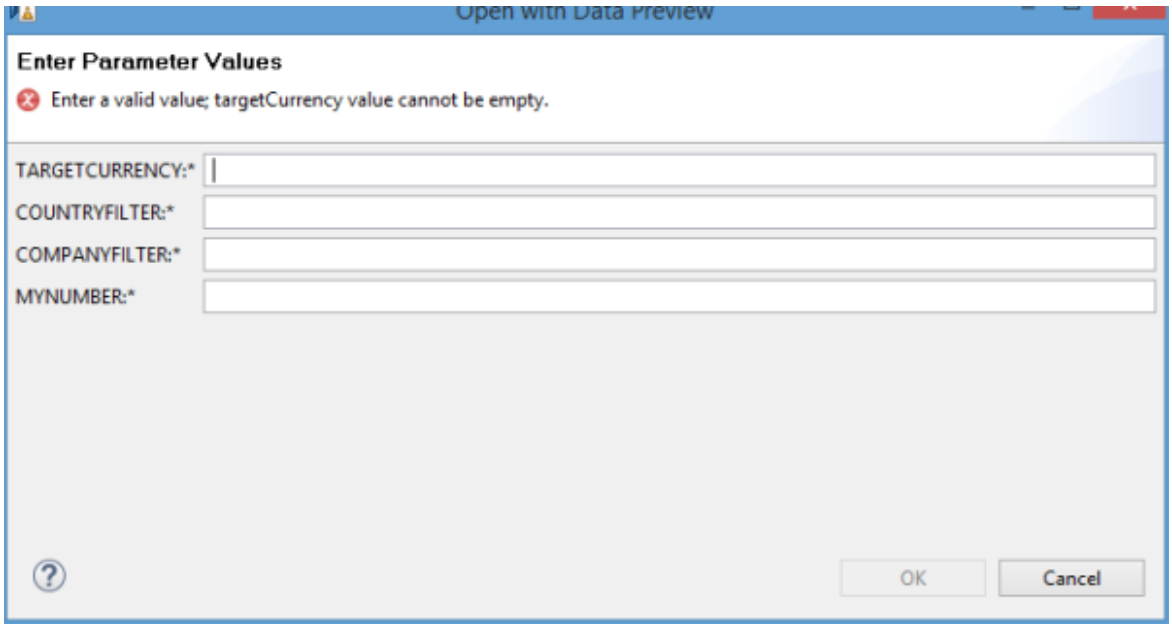
In the *Project Explorer* view, open the context menu of a data definition and choose *Open Data Preview*.

> **i** Note
>
> In addition, you can also open the *Data Preview* view from the source code editor of a data definition. Then choose *Open With* > *Data Preview* from the context menu.

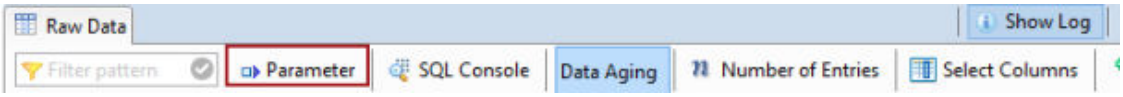
Results

The following possibilities might occur:

1. If the CDS View does not require any parameters, the *Data Preview* displays the result set directly.
2. If the CDS View requires parameters, a dialog to enter parameter values appears:



1. Enter your parameter values.
2. Choose *OK*.
   The *Data Preview* displays the result set.
   The *Outline* view displays parameter values of a CDS View. If you want to modify parameter values, choose the *Parameter* option that appears in the Data Preview tool.



> **Note**
>
> If the result set contains less records than you expect, there may be a access control role for the CDS entity that filters the data returned by the preview.

Related Information

Activating Data Definitions [page 41]

3.4.1 Following Associations in the CDS Data Preview

In CDS, an association represents the relationship between a CDS entity and a data source.

Context

In the *CDS Data Preview*, you follow associations to identify related data sources and display their contents.

Procedure

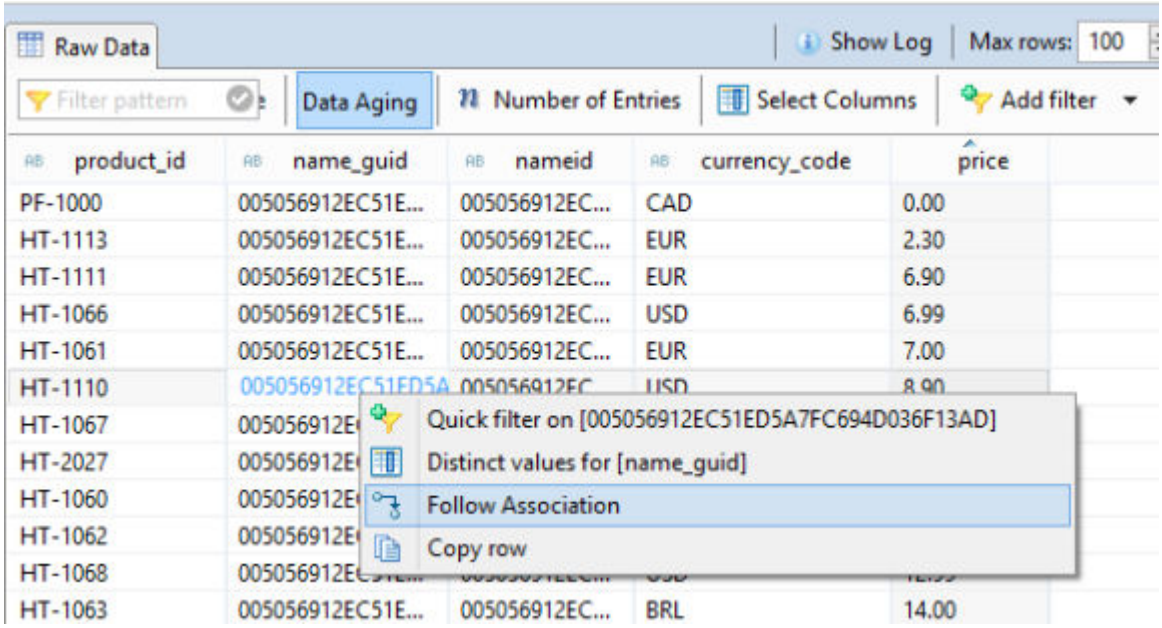
1. In the *Project Explorer* view, open the context menu of a data definition and choose *Open Data Preview*.

   > **i Note**
   >
   > In addition, you can also open the *Data Preview* view from the source code editor of a data definition. Then choose *Open With* > *Data Preview* from the context menu.

   > **i Note**
   >
   > If the selected CDS view requires parameters, a wizard for providing parameter values appears.

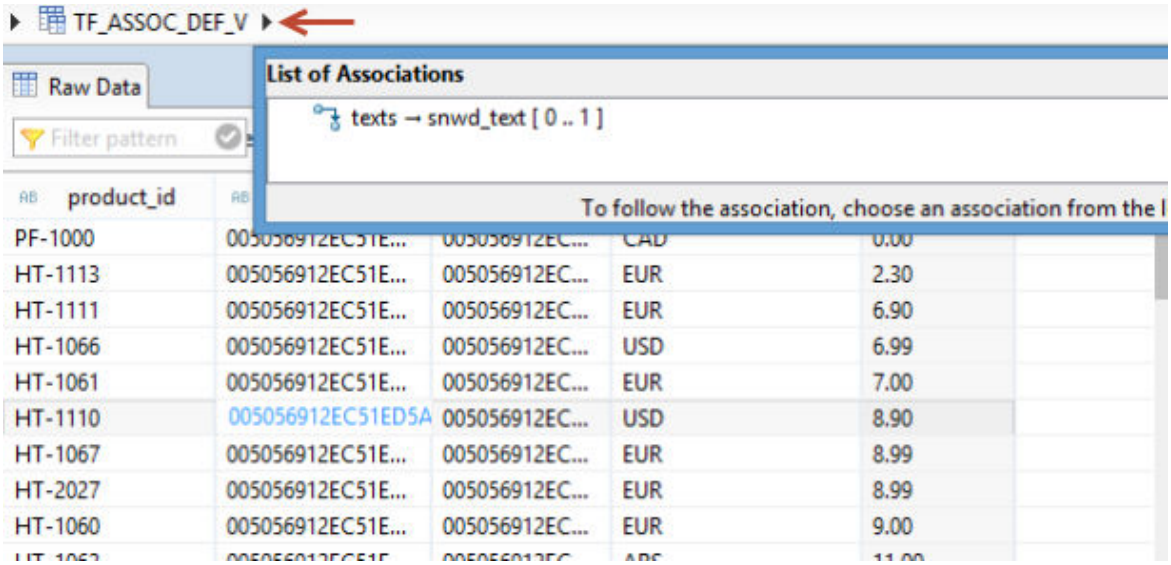
   The Data Preview tool appears and displays the top 100 records by default.

2. In the table context menu, choose *Follow Association*.



   > **i Note**
   >
   > You can also choose > in the breadcrumb bar to follow an association.
   >
   > The selected row/column is not relevant.

   Associations defined for the CDS view are listed.



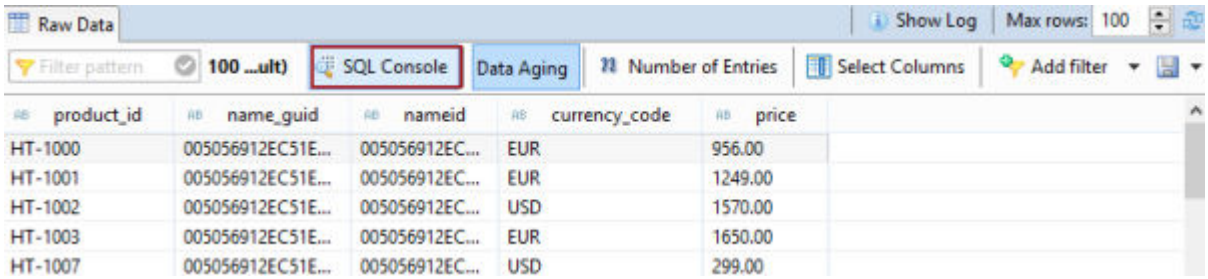
3. Choose an association.

Results

The *Data Preview* displays the result set for the selected association. You can apply filters to the current result set or use the breadcrumb to navigate to the previous result set. Any filters applied to the result sets are retained.

You can repeatedly follow associations through navigating into the hierarchy defined by the associations.

You can use the *Console* option on the *Data Preview* menu to display the generated Open query for an association. The generated query uses the CDS database view to display records.



3.5 Extending Data Models

You want to extend an existing data model with customer-specific data in order to retrieve additional data without creating redundant development objects or modifications.

You can extend the structure and the metadata of CDS views as follows:

- Extending the Structure of Data Models [page 82] to add additional element(s) to the CDS view from its used data sources.
- Extending the Metadata of Data Models [page 85] to add or overwrite the CDS annotations of a CDS entity in a separate development object.

Related Information

Extending CDS Entities [page 26]

3.5.1 Extending the Structure of Data Models

You can add further element(s) to the select list of a CDS view to extend its structure. This is achieved without modifications.

After activation, the additional element(s) will be available for the extended CDS view each time it is used.

You have the following possibilities to add and display structural extensions:

- Creating CDS View Extensions [page 83]
- Displaying and Navigating to CDS View Extensions [page 84]

Related Information

CDS Views Extension [page 28]

3.5.1.1 Creating CDS View Extensions

An CDS view extension allows you to extend the structure of an existing CDS view that is provided, for example, in the SAP standard. This extension will then not cause any modifications.

Prerequisites

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

Context

You want to add new elements, for example, to a CDS view of the SAP standard. Your changes need to be stable when upgrading your ABAP system.

> **i** Note
>
> In the source code, you can use the standard functions, such as transport, syntax check, code completion, and activation.

Procedure

1. In your ABAP project, select the relevant package node in the *Project Explorer*.
2. Open the context menu and choose *New* > *Other...* > *Core Data Services* > *Data Definition* to launch the creation wizard.
3. In addition to the *Project* and *Package*, enter the *Name* and the *Description* for the data definition to be created.

   > **i** Note
   >
   > You cannot specify the same name as the data definition of the CDS view you want to extend.
4. Choose *Next*.
5. Assign a transport request.
6. Choose *Next*.
7. [Optional:] Select the `Extend View` code template to be inserted in the created DDL source code.
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.

In the *Project Explorer*, the new data definition is added to the *Core Data Services* folder of the corresponding package node. As a result of this creation procedure, the source editor will be opened. Here, you can start to enter the name of the CDS entity to be extended and the element(s) that is to be added.

In the extended data definition, the  indicator is added to the `define view` statement. It indicates that an extension exists for this development object. You can navigate to the extend view by hovering over the indicator and following the link provided in the extension popup.

When activating the data definition that represents the extend view,

- An append view is created in the ABAP Dictionary.
- The extended element(s) is added to the existing CDS view.

Related Information

CDS Views Extension [page 28]
Viewing Generated SQL Statements [page 43]

3.5.1.2 Displaying and Navigating to CDS View Extensions

A CDS view extension provides additional fields, elements, or CDS annotations for a CDS view.

Context

A CDS view extension is an extend view or a metadata extension.

You want to investigate possible client handling properties **and/or** to open the CDS view extension(s) of a CDS view.

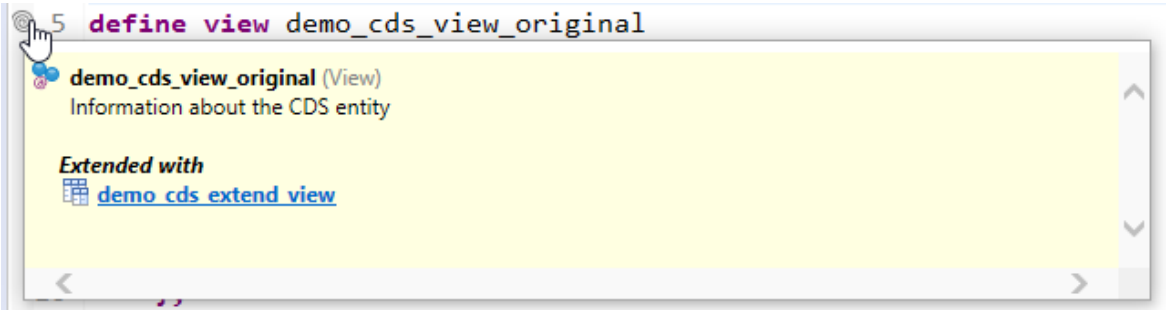
Procedure

1. In the vertical ruler of the DDL source editor, select the  marker.

   > **i** Note
   >
   > This marker indicates for each CDS view that a CDS view extension is assigned to the data definition.

It enables you to navigate to the extend view directly.

An *Extension* popup will be opened.



Example of an extension popup

Form here you will get the following information:

- Name of the CDS entity
- Content provided from the `@EndUserText.label` annotation
- Object link that refers to the extension of the relevant CDS view

2. To navigate to the CDS view extension that exist in the system, choose the underlined object name from the *Extended with* section in the *Element Information* popup.

Results

The CDS view extension is opened.

3.5.2 Extending the Metadata of Data Models

You can add customer-specific metadata to a CDS entity of the SAP standard that does **not** result in modifications. To do this, you provide additional CDS annotations.

You have the following possibilities to create and work with metadata extensions:

- Creating Metadata Extensions [page 86] to handle customer-specific metadata in a separate development object
- Extracting CDS Annotations to a Metadata Extension [page 87] to create a new metadata extension on the basis of a selection of CDS annotations from an existing CDS view

Related Information

Metadata Extensions [page 30]

3.5.2.1 Creating Metadata Extensions

A metadata extension allows you to extend a CDS entity with your own CDS annotations or to modify existing CDS annotations.

Prerequisites

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

Context

You want to provide additional metadata using CDS annotations to a CDS entity.

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. In addition to the *Project* and *Package*, enter the *Name* and the *Description* for the metadata extension to be created.
4. [Optional:] If you want to create a metadata extension for a specific CDS entity, enter the relevant entity name as the *Extended Entity*.

   > **i Note**
   >
   > You can only extend CDS entities.

5. Choose *Next*.
6. Assign a transport request.
7. Choose *Next*.
8. [Optional:] If requested, select a template.

   By default, ABAP Development Tools considers the template for creation that you have selected at the last time.
9. Choose *Finish*.

Results

In the selected package, the ABAP back-end system creates an inactive version of a metadata extension and stores it in the ABAP Repository.

In the *Project Explorer*, the new metadata extension is added to the *Core Data Services* folder of the corresponding package node. As a result of this creation procedure, the source editor will be opened. Here, you can start defining a metadata extension.

After developing and checking your new object, you can activate it.

3.5.2.2 Extracting CDS Annotations to a Metadata Extension

You can create a metadata extension object that contains the annotations of a specific data definition.

Prerequisites

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

Context

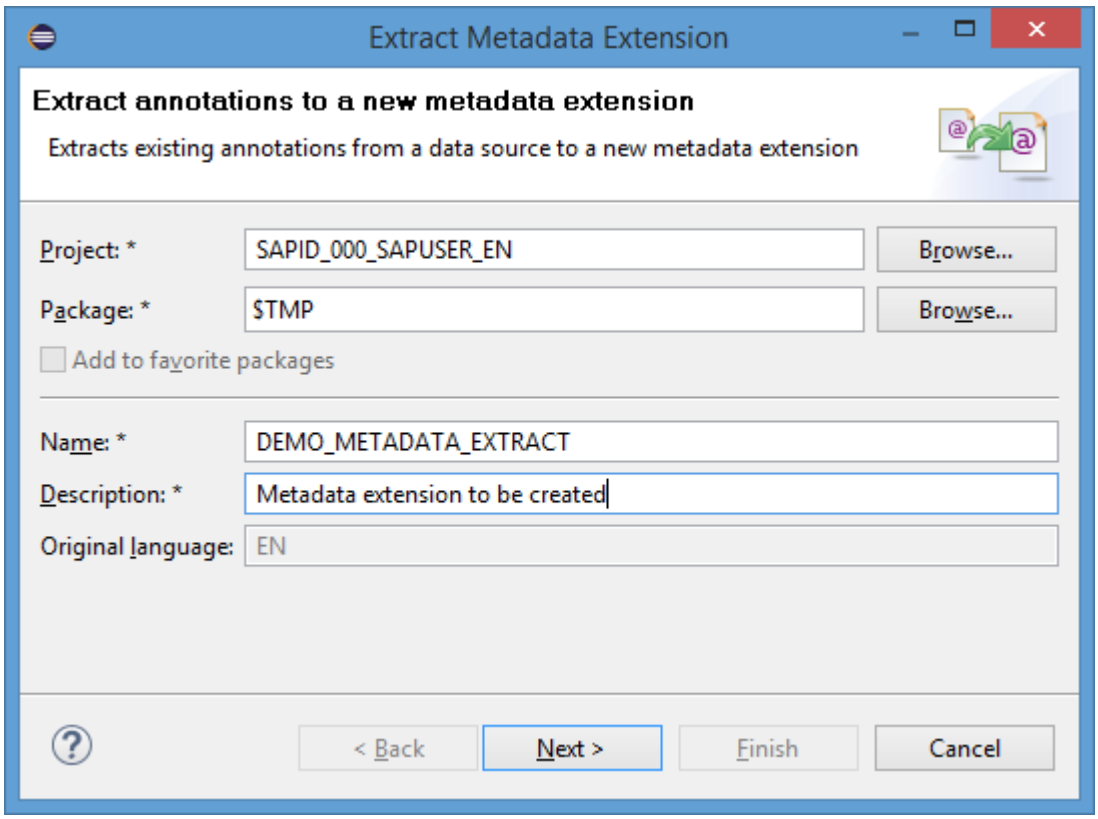
You want to relocate the annotations of a data definition to a new metadata extension to be created.

> **i** Note
>
> The annotations are removed from the data definition after extraction.
>
> You can only extract those annotations that are allowed for usage in metadata extensions.

Procedure

1. Open the source editor for the relevant data definition.
2. Open the context menu in the source editor and choose *Source Code* > *Extract Metadata Extension* to launch the *Extract Metadata Extension* wizard.

   The *Extract Metadata Extension*



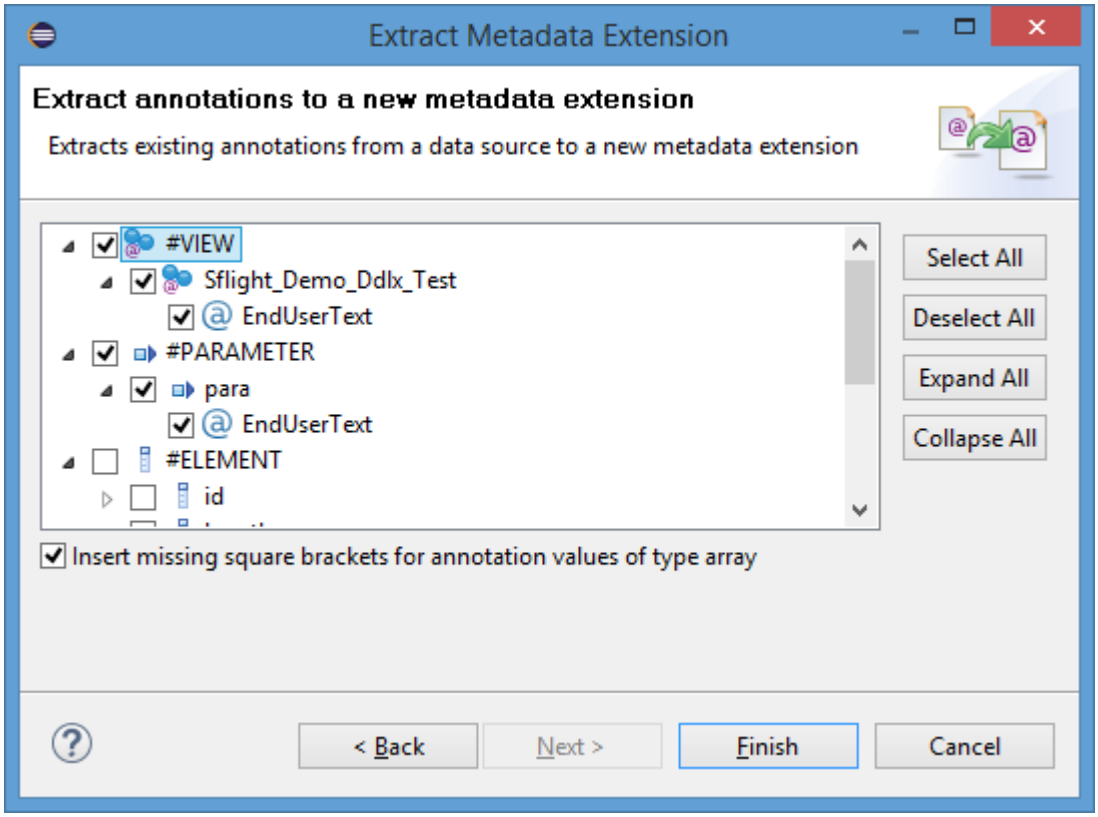
Wizard page for defining a new metadata extension to be created

3. In addition to the *Project* and wizard is opened.*Package*, enter the *Name* and the *Description* wiz for the metadata extension to be created.

> **i Note**
>
> The maximum length for names of metadata extensions is 30 characters.

4. Choose *Next*.

   The second page of the creation wizard is opened.



Wizard page for defining the annotations to be extracted to the new metadata extension

The annotations of the data definition that can be extracted to the new metdata extension are listed here.

5. Select the annotation(s) to be relocated from the data definition.

> **i Note**
>
> Annotation values of type `array` must be set in square brackets in metadata definitions. If the `array` values in your data definition are not set in square brackets, you can use the *Insert missing square brackets for annotation values of type array* checkbox to automatically insert the brackets when the annotation is extracted.

6. Choose *Next*.

   The *Selection of Transport Request* page is opened.

7. Assign a transport request.
8. Choose *Finish*.

Results

The ABAP back-end system creates an inactive version of a metadata extension and stores it in the ABAP Repository.

In the *Project Explorer*, the new metadata extension is added to the *Core Data Services* folder of the corresponding package node. The source editor of the metadata extension is opened and the extracted annotations are added. Therefore, the source code of the metadata extension becomes dirty.

The editor of the data definition also becomes dirty. Here, the annotations are deleted from its source code.

Consequently, you must save and activate both development objects to apply the changes.

Related Information

Annotation Propagation View [page 98]
Annotation Propagation [page 8]

3.6 Analyzing Data Models

3.6.1 Analyzing Dependencies in Complex CDS Views

The Dependency Analyzer provides several possibilities to evaluate the relationships and complexity from a CDS entity with regards to its SQL definition.

Prerequisites

The SQL dependency can only be calculated for the active version of a data definition.

Context

You want to display SQL dependencies of a CDS view. You can also use it to identify, for example, performance issues.

Procedure

1. In the *Project Explorer*, select the data definition that you want to analyze.
2. Open the context menu and choose *Open with* > *Dependency Analyzer*.

   → Tip

   Alternatively, you can open the same context menu from the DDL source editor of the relevant CDS entity.

   The dependencies of data sources involved in the CDS view are calculated. The *SQL Dependency Tree* tab is opened by default and displays the result in a tree structure.

3. To display the relevant information from the *SQL Dependency Tree* tab, proceed as follows:
    - To get this data for the CDS view: Open the *Complexity Metrics* subtab. The aggregated statistics are listed there.
    - To get this data for a specific data source: Select the relevant *SQL Name* entry and choose *Show Metrics Complexity* from the context menu. The *Properties* view is opened where the aggregated statistics are then displayed.
    - To visualize this data for the CDS view, open the SQL *Dependency Graph* tab.

Results

Based on the relevant information, you can now, continue your work and improve your data model.

Related Information

Dependency Analyzer [page 100]

3.6.2 Analyzing Annotation Propagations

You use the *Annotation Propagation* view to understand how the metadata was merged for a CDS View.

Prerequisites

The CDS View is active.

Context

You want to reproduce how the value for an element annotation was derived.

Procedure

1. In the *Project Explorer*, select the data definition you want to analyze.
2. Open the context menu and choose *Open With* > *Annotation Propagation*.

> → Tip
>
> Alternatively, you can open the same context menu from the DDL editor of a data definition.

The *Annotation Propagation* view is opened and displays by default the values that are propagated for the view annotations.

> i Note
>
> If you have placed the cursor on an annotation or an element in the DDL editor, then this element or annotation will be pre-selected and value propagations for this selection will be displayed.

3. In the `<Selection>` section, enter the details of the element annotation you want to investigate.
   a. In the `<Annotations For *>` input field, enter the name of the element.
   b. In the `<Annotation Filter>` input field, enter the name of the annotation.

> → Tip
>
> Use the content assist (shortcut `Ctrl` + `Space`) in the input fields to get proposals for the names. Alternatively, you can choose the `<Browse ...>` button and search for names.

4. To display the value propagation for the selected element annotation, choose the `<Apply>` button.

Results

All objects which assign a value to the element annotation are listed in the `<Value Propagation>` section. The objects are listed in the order of precedence. The active value of the annotation is highlighted together with information about the contributing object.

Related Information

Annotation Propagation View [page 98]

3.6.3 Displaying Annotation Values of an Active CDS View

In the *Active annotations* view, you can display the following content for an active data definition: Which elements/parameters have been annotated, their current value, and their origin.

Prerequisites

The data definition you are currently editing has already been activated.

Context

You want to find out which annotation values (including the propagated ones) does a CDS view contain and where the individual annotation values are originated from.

You can open the *Active annotations* view from the *Project Explorer* as follows:

Procedure

1. In the *Core Data Services* ABAP repository tree, open the context menu from the relevant data definition.
2. Choose *Open with* > *Active Annotations*.

> **i** Note
>
> Alternatively, you can open the same context menu from the DDL editor of a CDS view.

Results

The *Active Annotations* view is opened in the structured mode. It displays all the CDS annotations that are defined in the CDS view itself or are inherited from the underlying data sources or data elements.

If the values are inherited from underlying data sources or data elements, you can navigate to these development objects.

Related Information

Active Annotations View [page 96]

3.7 Tuning Access to SAP HANA

You want to improve the performance of your applications, for example, by reducing the response time when retrieving data from datasources stored in database tables on SAP HANA.

Use

You can use database tuning objects to optimize the consumption of resources and the performance when accessing data on SAP HANA.

→ Recommendation

Using database tuning objects requires memory space on the database. This space is not available for other database operations. Therefore, SAP recommends pondering if the memory space that is required for the aggregation results is smaller than the space that is required for the dynamic caches.

Features

You can optimize the access to SAP HANA with the following database tuning objects:

- Creating Dynamic Caches [page 94]

3.7.1 Creating Dynamic Caches

You can temporarily store the result of one or more aggregations (for example, `sum`, `min`, `max`, `avg`) in a dynamic cache. This avoids repetitive calculations of frequently executed queries, and improves performance. A dynamic cache is a development object that supports the standard functionality (transport, syntax check, and activation).

Prerequisites

Context

You want to improve the performance of evaluating a data model by reusing the result of intense aggregations.

Procedure

1. In your ABAP project, select a package node in the *Project Explorer*.
2. To launch the creation wizard, open the context menu and choose *New* > *Other...* > *ABAP* > *Dictionary* > *Dynamic Cache*.
3. In addition to the *Project* and *Package*, enter a *Name* and *Description* for the dynamic cache to be created.

   i Note

   The maximum length for names of dynamic caches is 30 characters.

4. Choose *Next*.
5. Assign a transport request.
6. Choose *Next*.
7. Select the `Define Dynamic Cache` template.

   By default, ABAP Development Tools takes the last used template for creation.
8. Choose *Finish*.

Results

In the selected package, the ABAP back-end system creates an inactive version of a dynamic cache, and stores it in the ABAP Repository.

In the *Project Explorer*, the new dynamic cache is added to the *Dictionary* folder of the corresponding package node.

As a result of this creation procedure, the source editor is opened. Here, you can start adding the database table, and define its data to be cached.

> **i** Note
>
> ☁ To view and manage the dynamic cache configurations, you can use the `Manage Database Cache Configuration` app.

Related Information

Dynamic Result Cache

☁ Manage Database Cache Configuration

3.8 Ensuring Quality with ABAP Unit

You can ensure the quality of DDL code using CDS Test Double Framework.

CDS Test Double Framework enables you to test the logic expressed in their CDS entities. It helps you to break dependencies between the object under test and the database and to take control over the data.

4 Reference

4.1 Active Annotations View

The *Active Annotations* view displays the CDS annotations that are used for an active CDS view and the relevant data sources in a tabular way. The content, displayed in the *Active Annotations* view, depends on the underlying data definition.

Metadata Origin

CDS annotations and their values can be defined in the CDS view itself or inherited from the underlying data sources or data elements.

Annotation values are only inherited for elements (fields and associations) and parameters. Parameters can only inherit from the corresponding data elements, not from underlying data sources.

In addition, CDS annotations defined by SAP in metadata extensions are also merged.

Overview

In the *Active Annotations* view, the following columns are displayed:

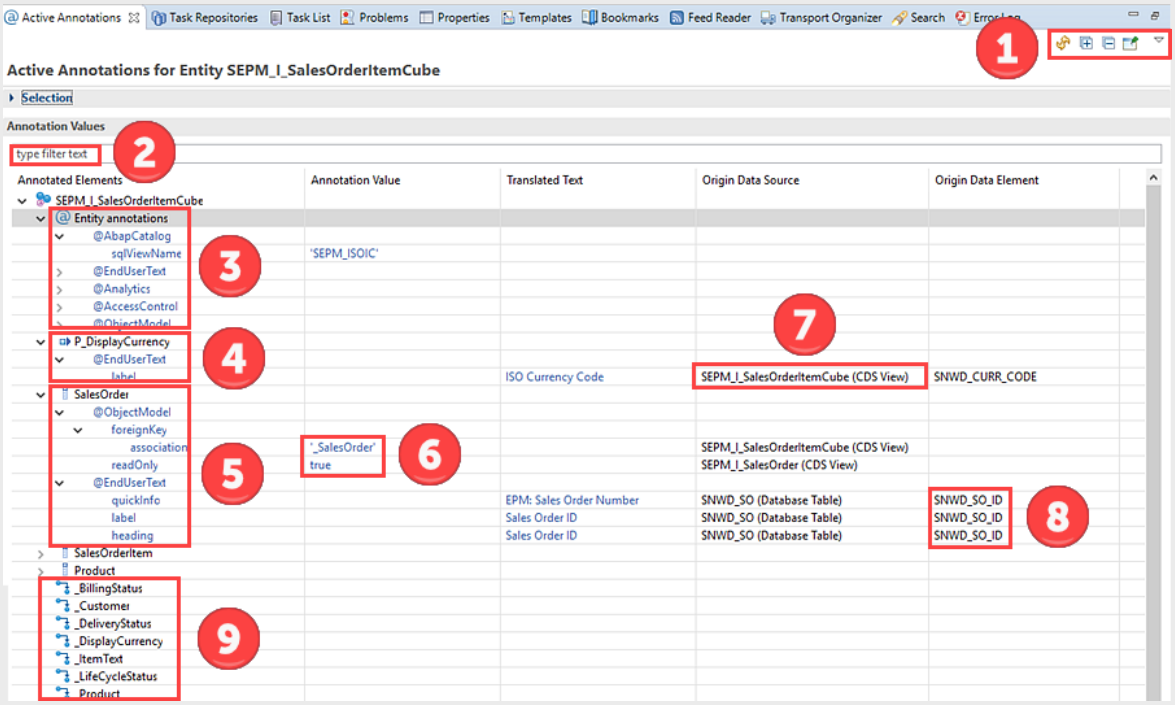
- *Annotated Elements*: The view, its parameters, fields, and associations, and their active annotations
- *Annotation Value*: Values or text in the original language of the corresponding annotation
- *Translated Text*: Text that is provided for the corresponding annotation in the respective logon language
- *Origin Data Source*: Name of the development object, for example, a data definition or database table from which the corresponding annotation is inherited. You can navigate to this object by double-clicking its name.
- *Origin Data Element*: Name of the data element from which the corresponding annotation is inherited. You can navigate to this object by double-clicking its name.

The *Active Annotations* view provides you the following possibilities:

- In the search field, you can enter a filter text to display only specific entries in the whole view.
- From the *View Menu* in the toolbar, you can toggle between the flat or structured display of the active annotations. To switch the display mode, choose the arrow button from the view toolbar and select the relevant entry.

  > **i** Note
  >
  > The structured mode groups the annotations by their parent node, and is set by default.
  >
  > The flat mode lists the annotations in accordance to their fully qualified names.

Example



Example of a setup from an Active Annotations view and its functionalities

The *Active Annotations* view for the `SEPM_I_SalesOrderItemCube` data definition provides the following information:

1. Integrated toolbar
2. Filter field to search for entries in the whole view
3. Entity annotations
4. Parameter and its parameter annotation
5. Element and its element annotations
6. Current values of the corresponding annotations
7. Data source where the annotation value is inherited from, and in brackets its object type
8. Data element where the annotation value is inherited from
9. Associations

Related Information

Displaying Annotation Values of an Active CDS View [page 92]

4.2 Annotation Propagation View

Definition

The *Annotation Propagation* view displays the CDS entity or metadata extension from which the value of a CDS annotation has been propagated.

Use

In a data model, the values of the CDS annotations might derive from different CDS objects. In order to understand how the values of the used CDS annotations are determined and to visualize this merge process, the *Annotation Propagation* view is provided.

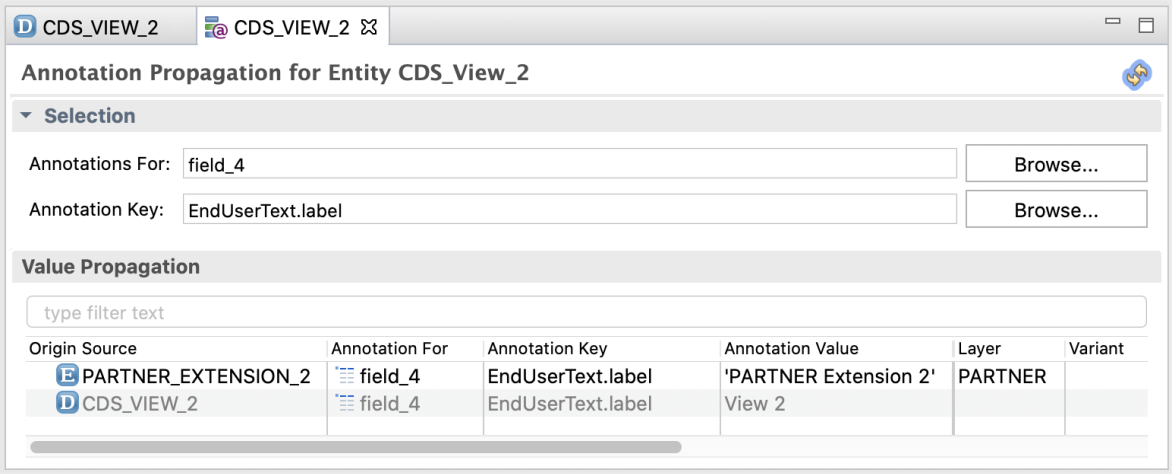
Overview

Element annotations in data definitions and metadata extensions are compounded and propagated along the hierarchy of CDS entities.

In the hierarchies of CDS entities, the element annotation values are propagated from the underlying CDS object to the CDS objects above.

To reproduce the CDS objects from which propagation has been evaluated, you can open the *Annotation Propagation* view. Note that values which are not considered or will be overwritten are marked in gray.

Example



Example of an Annotation Propagation view

In this example, you as a developer have triggered the *Annotation Propagation* view with the following selection:

- *Entity*: `CDS_View_2`
- *Annotations For**: `<field_4>` is the field within the select list.
- *Annotation Key*: The annotation `EndUserText.label` which is specified for `<field_4>`.

The value `View 2` defined in the `CDS_VIEW_2` data definition is overwritten by the value `'PARTNER Extension2'` that is defined in the `PARTNER_EXTENSION_2` metadata extension. Therefore, `CDS_VIEW_2` is displayed gray.

In the *Value Propagation* section, the following general information is provided:

- *Origin Source*: Transport object name of the contributing object
  In this example, the field represents the name of the CDS object where the annotation is defined.
- *Annotation For*: Name of the annotated element, parameter, or data source that is specified in the *Selection* area
- *Annotation Key*: Name of the annotation for which you have created the *Annotation Propagation* view or which filter for
- *Annotation Value*: Current value of the corresponding annotation
- *Layer*: Value that is added to the `@Metadata.layer` annotation that was assigned to the contributing metadata extension.

  > **i Note**
  >
  > This value is used internally by the ABAP infrastructure to implement the precedence of the annotations specified in metadata extensions.
  >
  > This column is empty if the contributing object is not a metadata extension.

- *Entity Name*: Name of the contributing CDS entity or metadata extension

  > **i Note**
  >
  > This column is empty if the contributing object is not a CDS object.

- *Switch Status*. The following columns provide information about the Switch state of a metadata extension:
  - *Switch Status*: Displays the switch state of a metadata extension as follows:
    - If the value `ON` is displayed, the metadata extension is enabled.
    - If `OFF` is displayed, then the metadata extension does not contribute to the metadata of a CDS entity.
    - If a Switch is not assigned to the metadata extension, then the metadata extension is always enabled. In this case, the entries in the *Switch Package* and *Switch Name* columns are empty.
  - *Switch Package*: Name of the corresponding metadata extension's package if a Switch is assigned to the metadata extension
  - *Switch Name*: Name of the Switch which is assigned to the metadata extension
- *Layer Number*: Numeric value of the metadata extension's layer

  > **i Note**
  >
  > This value is used SAP-internally only to implement the precedence of the annotations specified in metadata extensions.

Annotations can be defined/inherited in/from data definitions or metadata extensions.

For a given layer or a CDS variant annotations might be inactive. If so, they are displayed in gray and not considered for evaluation. Active annotations are displayed in black.

Related Information

Annotation Propagation [page 8]
Displaying Annotation Values of an Active CDS View [page 92]
Analyzing Annotation Propagations [page 91]
Metadata Extensions [page 30]

4.3 Dependency Analyzer

The *Dependency Analyzer* evaluates the relationships and complexity from a CDS entity with regards to its SQL definition.

You use the *Dependency Analyzer* to investigate which database objects (such as CDS database views, database tables, database views, and CDS table functions) are used in your CDS view.

In addition, it helps you to understand SQL dependencies and complexity that might negatively affect the performance of your query.

The *Dependency Analyzer* provides you with the following tabs:

- The SQL Dependency Tree [page 101] tab displays dependencies as hierarchy.
- The SQL Dependency Graph [page 104] tab displays dependencies in a graphical map.
- The Complexity Metrics [page 106] tab displays a statistical summary of selected key figures (such as used data sources, SQL operations, function calls, and expressions).

Related Information

Analyzing Dependencies in Complex CDS Views [page 90]

4.3.1 SQL Dependency Tree

The *SQL Dependency Tree* tab displays SQL dependencies of a CDS view on other database objects.

Use Cases

You as a developer open the *SAP Dependency Tree* tab in the following cases:

- You want to edit a CDS view: You want to check the top to bottom SQL dependencies of a CDS view.
- You want to reuse a CDS view: You want to understand the dependencies of a CDS view before reusing it.
- If there are performance or activation issues, you want to find out which database object may be the cause.

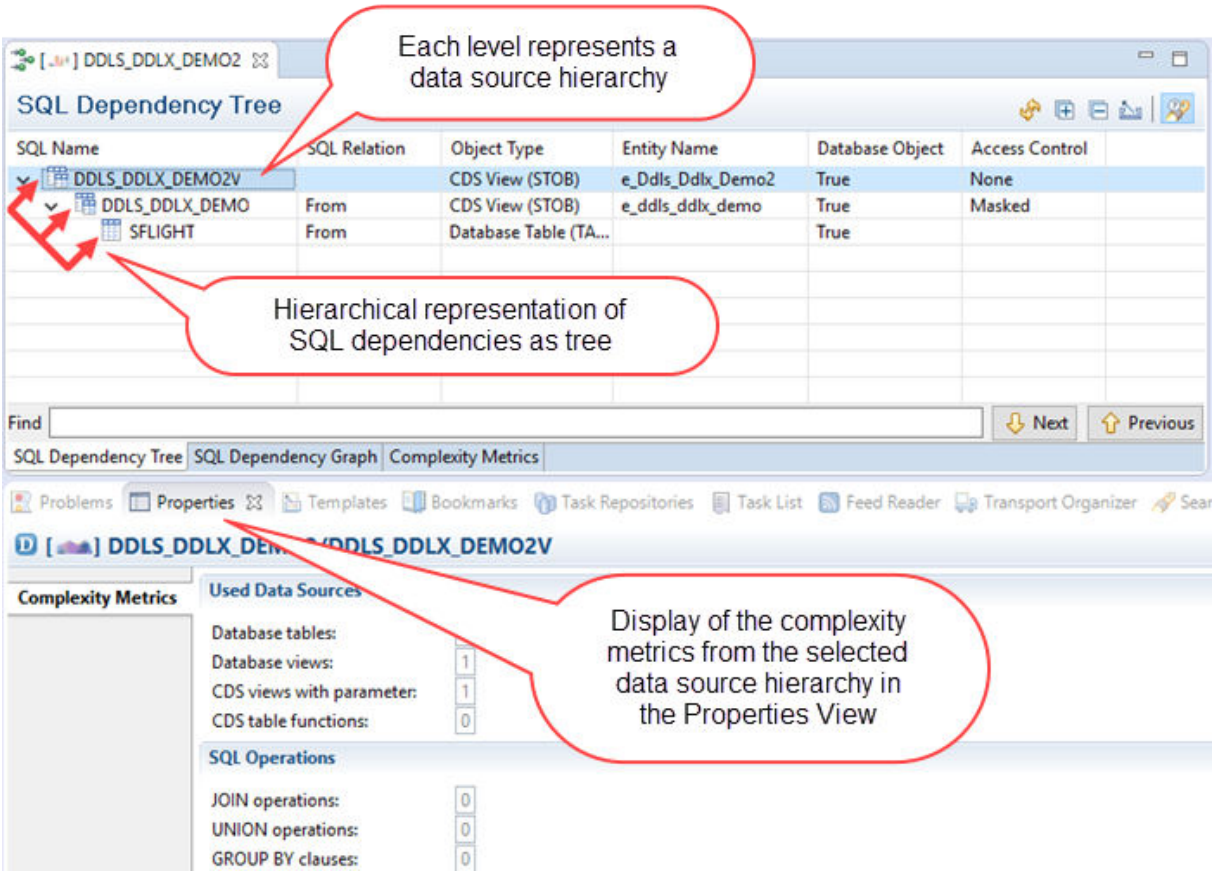
Overview

The data sources involved are determined recursively and the result is displayed in a tree structure. The following database objects are possible as data sources:

- CDS database view
- CDS table function
- Database view
- External view

  > **i Note**
  >
  > External views are used to access the SAP HANA-based views in the ABAP source code. To access this kind of data model, your ABAP systems needs to be connected with a SAP HANA database.

- Database table



Example of a SQL dependency tree

A dependency tree provides the following columns:

- *SQL Name*: Name of the database object
- *SQL Relation*: How a database object may be related to its parent object.

  > ### Example
  >
  > The following relations might be displayed:
  >
  > - Select: `(SELECT) FROM`
  > - Joins: `INNER`, `LEFT OUTER`, `RIGHT OUTER`
  > - Set operations: `UNION, UNION ALL`

- *Type*: Name of the database object type
- *Entity Name*: Name of the CDS view
- *Database Object*: Status if the corresponding artifact physically exists in the database. Then, the value `true` is displayed.

  > ### Note
  >
  > If the field is empty, the corresponding artifact does not exist in the database.

- *Access Control*: Status of the object's access controls with respect to analyzed CDS views
  The following status are displayed:
  - Defined: The access control is defined for the CDS view as root node of the dependency tree.

- Masked: The access control is defined for the CDS view(s) as a sub node of the dependency tree.
- None: No access control is defined for the CDS view.
- [Empty]: The database object is **no** CDS view and provides therefore **no** information about access controls.

> **i** Note
>
> To find the access control that is defined, choose *Open Other* from the context menu of the relevant `Defined` or `Masked` status.

Additional Functionality

In addition to displaying dependencies, you can perform the following functions:

Toolbar

- To refresh the dependency tree, for example, after modifications have been made and the data definition has been activated, choose the *Refresh* icon from the toolbar.
- To expand/collapse all nodes in the tree structure, choose the *Expand All* or *Minimize All* icons from the toolbar.
- To export the graph as PNG file, choose the  icon from the toolbar.

Tree

- To navigate to the listed data sources through double-click or using the context menu.
- To search for objects, enter the relevant name in the *Find* field of the search toolbar. If the search toolbar is not displayed, choose the *Show Search* icon from the graph toolbar to make it available.

Context Menu

- To display the complexity metrics of a CDS or database object, select the relevant object and choose *Show Complexity Metrics* from the context menu. Then, the *Properties* view opened where the relevant information is displayed.

Related Information

SQL Dependency Graph [page 104]
Complexity Metrics [page 106]

4.3.2 SQL Dependency Graph

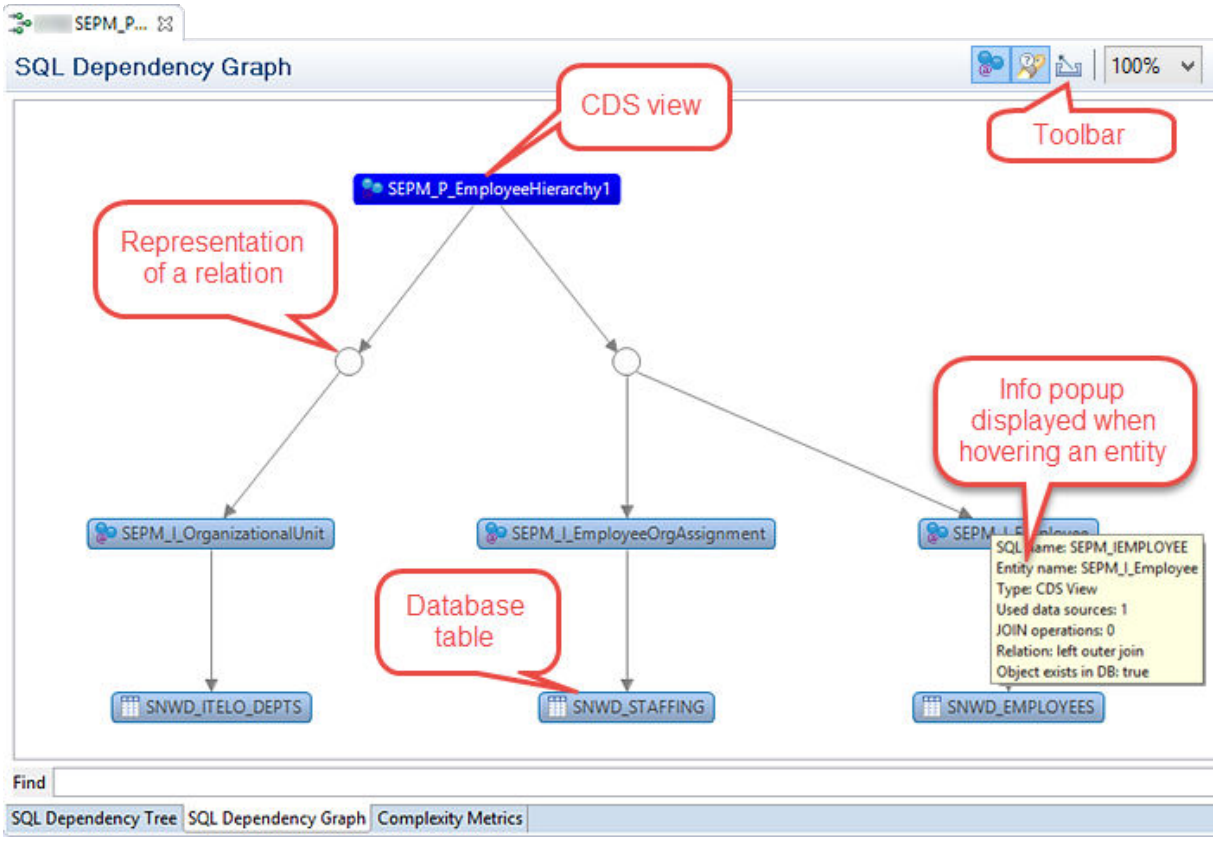
The *SQL Dependency Graph* visualizes SQL dependencies between ABAP Dictionary objects and CDS views.

Use Case

You want to visualize the dependencies of a complex CDS view and its neighboring objects in order to investigate their relationships.

Overview

The *SQL Dependency Graph* tab contains the same information as the *SQL Dependency Tree* tab but it visualizes the data model in a graph. This means the closer the nodes representing the objects are, the closer is their relationship. Hierarchies are represented by the reference to objects on the subsequent level. Objects, that are on the same level, are, if possible, displayed on the same imaginary line. Relations are displayed as circles.



Example of a calculated SQL Dependency Graph

> **Note**
>
> The coloring of the elements used in the *SQL Dependency Graph* is predefined. To adapt the coloring, open the *ABAP Development* > *Graphical Tools* > *Dependencies Analyzer* preference page and assign another color to the element.

Additional Functionality

In addition to displaying dependencies, you can perform the following functions:

Toolbar

- To toggle between the *SQL Name* and *Entity Name* in the graph, select the  icon from the toolbar.
- To export the graph as a PNG file, choose the  icon from the toolbar.
- To maximize or minimize the current position, choose the relevant value from the zoom dropdown listbox in the toolbar.

Graph

- To display general and statistical information about a CDS or database object in the tooltip popup, mouse over the relevant object.
- To follow the dependencies of neighboring objects, select the corresponding node. Then, the contrast of the relevant dependencies is highlighted.
- To search for objects, enter the relevant name in the *Find* field of the search toolbar. If the search toolbar is not displayed, choose the *Show Search* icon from the graph toolbar to make it available.
- To display the complexity metrics of a CDS or database object, select the relevant one and choose *Show Complexity Metrics* from the context menu. The *Properties* view will open where the relevant information is displayed.
- You can highlight the elements of the graph in the context of their interdependencies.
  You can mask the following attributes:
  - Object type
  - State of SQL interrelations,

  To highlight them, choose *Highlight* from the context menu on the white area of the graph. Then select the relevant entries from the submenus of one or more attributes.

  > **Note**
  >
  > If you select one or more entries:
  >
  > - from the submenu of the same attribute, all relevant elements will be highlighted that match at least one of the selected entries.
  > - from several submenus of different attributes, only those elements will be highlighted that match the selection from the submenu of the same attribute **and** from the submenus of the other ones.

Outline

To navigate within large graphs, move the highlighted area in the *Outline* view as required.

Related Information

SQL Dependency Tree [page 101]
Complexity Metrics [page 106]

4.3.3 Complexity Metrics

The *Complexity Metrics* tab enables you to check CDS views regarding performance issues.

Use Case

You want to check the characteristics that influence the performance of your CDS view.

Overview

This tab summarizes statistical information about a CDS view in the following sections:

- *Used Data Sources*: List of the aggregated number of database objects that depend on the selected CDS view

  > **i** Note
  >
  > Each usage is counted separately. Identical data sources are not grouped.

- *SQL Operations*: List of the aggregated number of the SQL operations that might be most relevant for performance issues
- *Performance Related Function Calls and Operations*: List of the aggregated number of expressions and function calls

Related Information

SQL Dependency Tree [page 101]
SQL Dependency Graph [page 104]
ABAP CDS - SELECT, Predefined Functions (ABAP Keyword Documentation)

4.4 Glossary

Access Control

ABAP development object that is used to define authorizations for CDS entities

An access control allows you to limit the results returned by a CDS entity to those you authorize a user to see.

Core Data Services (CDS)

CDS introduce a common set of domain-specific languages (DSL) and services for defining and consuming semantically rich data models.

CDS Database View

Projection onto one or multiple relational database tables or other views

An CDS database view is generated in the ABAP Dictionary after activation of the data definition. The structure of a CDS database view is defined in a CDS entity. A CDS database view is a technical representation of the CDS entity.

CDS Entity

Part of a data definition

The definition of a CDS entity is introduced with the `DEFINE VIEW` statement. A CDS entity is used to specify the structure of a CDS database view.

CDS Entity Under Test (`CUT`)

The "system under test". It is short for "whatever thing we are testing" and is always defined from the perspective of the test. When we are writing unit test the `SUT` is whatever class (a.k.a. `CUT`), object (also known as `OUT`) or method(s) (also known as `MUT`) we are testing; when we are writing customer tests, the `SUT` is probably the entire application (also known as `AUT`) or at least a major subsystem of it. The parts of the application that we are not verifying in this particular test may still be involved as a `DOC`.

Clones

Creating a temporary clone (copy) of the `CUT` in the same database schema.

For all purposes, this clone serves as the CDS entity under test. The logic implemented in the original CDS entity is preserved in the clone but the depended-on components are replaced by the corresponding `Test Doubles` that are created by the CDS Test Double Framework.

Data Control Language (DCL)

DCL is used to define authorizations for CDS entities. The main goal of CDS is to make the usage of SQL easier for application developers. DCL offers a possibility to define the authorizations needed for the CDS entities in a modeled, declarative way.

Data Definition

ABAP development object that is used to define a CDS view entities

A data definition is created in ABAP Repository using a wizard of ABAP Development Tools.

DCL Editor

Text-based editor for editing DCL sources, such as access controls

The DCL editor is part of ABAP Development Tools and allows you to define the role or access policy for the CDS entity.

Data Definition Language (DDL)

Subset of SQL

DDL statements are used to create and delete the entities of a relational database. In AS ABAP, DDL is integrated into ABAP Dictionary.

DDL Editor

Text-based editor for editing development objects containing DDL such as data definitions and metadata extensions

The DDL editor is part of ABAP Development Tools.

Depended-on components (`DOC`)

An individual class or a large-grained component on which the `system under test` (`SUT`) depends. The dependency is usually one of delegation via method calls. In test automation, it is primarily of interest in that we need to be able to examine and control its interactions with the `SUT` to get complete test coverage.

For more information, see *http://xunitpatterns.com/DOC.html*.

Metadata Extension

ABAP development object that is used to annotate CDS entities with metadata without modifications.

5 What's New in ABAP CDS Tools

ABAP CDS tools are an integral part of the client installation of ABAP Development Tools (ADT). ADT is released to customers in combination with the corresponding back end shipment. This means, in order to use certain ADT functionalities, you need to provide the corresponding back end.

The following table gives you an overview of the released ADT versions and ABAP back ends:

☁ ABAP Environment

The following list gives you an overview of the released ADT client versions:

- Version 3.14 [page 110]
- Version 3.12 [page 111]
- Version 3.6 [page 114]
-
- Version 3.0 [page 117]

5.1 ☁ Version 3.14

Here is an overview of the most significant changes in the context of ABAP CDS development that relate to the following:

- Client: ABAP Development Tools (ADT) **3.14**
- Back end version: ☁ **SAP Cloud Platform ABAP Environment 2011**

> **i** Note
>
> The following features that are highlighted with a '*' are client-specific and are therefore available for all supported ABAP systems.

☁ Adding Alias for "Insert All Elements" in CDS Code Completion Preference

When creating a data definition, using a database table as datasource, and performing *Insert all elements* or when creating a data definition on base of a database table, ADT will generate an alias for each element and add it **and** the name of the element in the select statement of the data definition by default

i For more information, see Adding Aliases for Inserted Elements from Database Table [page 50]

5.2 Version 3.12

Here is an overview of the most significant changes in the context of ABAP CDS development that relate to the following:

- Client: ABAP Development Tools (ADT) **3.12**
- Back end version: **SAP Cloud Platform ABAP Environment 2008**

> **i** Note
>
> The following features that are highlighted with a '*' are client-specific and are therefore available for all supported ABAP systems.

In this topic, you can find release information about:

- Creating and Activating Data Models [page 111]
- Documenting Development Objects [page 112]
- Displaying Element Information for CDS Annotations in Source Code Editors [page 112]
- Using Wildcards When Performing Code Completion for CDS Annotations [page 114]

Creating and Activating Data Models

Creating and Editing CDS View Entities

You can now create and edit the new CDS view entities that provide improved functionalities, such as activation, and so on.

To do this, you create or edit a data definition using the `define view entity` template/statement.

> **→** Recommendation
>
> SAP recommends using CDS view entities instead of CDS DDIC-based view due to technical improvements, such as performance at activation, and so on.

i For more information, see

- CDS View Entities [page 12]
- Creating a Data Definition [page 33]

Creating ABAP CDS Objects With Reference to Other Objects

When creating, for example, a data definition, you can now refer to an existing CDS object or database table. This enables you to insert all elements of a referenced object in your new CDS object to be created.

Referencing at creation is provided in the following creation wizards and source code editors:

- **Data definitions**: You want to insert all elements from a referenced CDS entity when creating a data definition.
  Note that inserting all elements is only realized for data definitions.
- **Access control**: You want to protect a CDS entity by creating an access control.

- **Metadata extension**: You want to extend a CDS entity by creating a metadata extension
- **Service definition**: You want to expose a CDS entity by creating an access control.

ℹ For more information, see Creating ABAP CDS Objects With Reference to Other Objects [page 37]

Documenting Development Objects

Providing Documentation for Data Definitions and Service Definitions

In the context of ABAP CDS development, you can now provide documentation using knowledge transfer documents (KTD) for data definitions and service definitions. This enables you to note details about the development goal of your CDS object for other ABAP developers of your team.

To create a KTD, choose *New Knowledge Transfer Document* from the context menu of the relevant CDS object in the *Project Browser*.

ℹ For more information, see

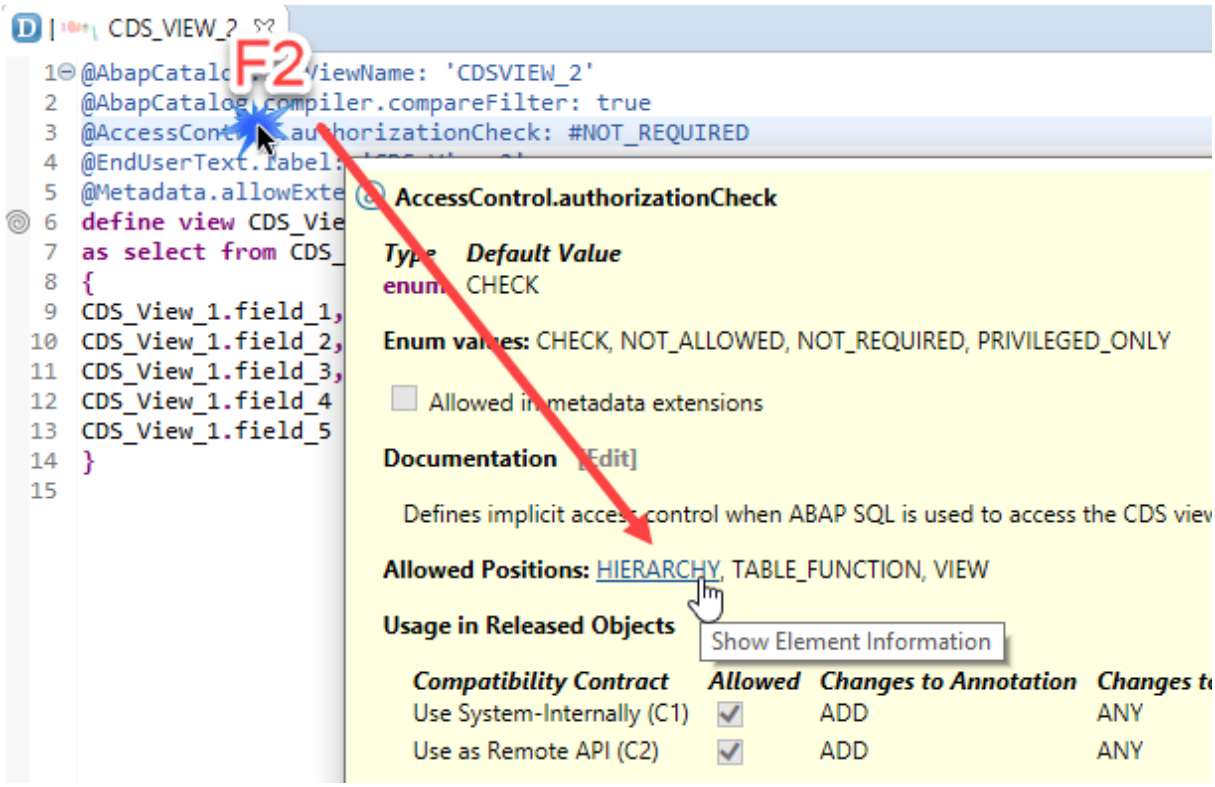
- Documentation of CDS Objects [page 32]
-
-

Editing DDL Source Code

Displaying Element Information for CDS Annotations in Source Code Editors

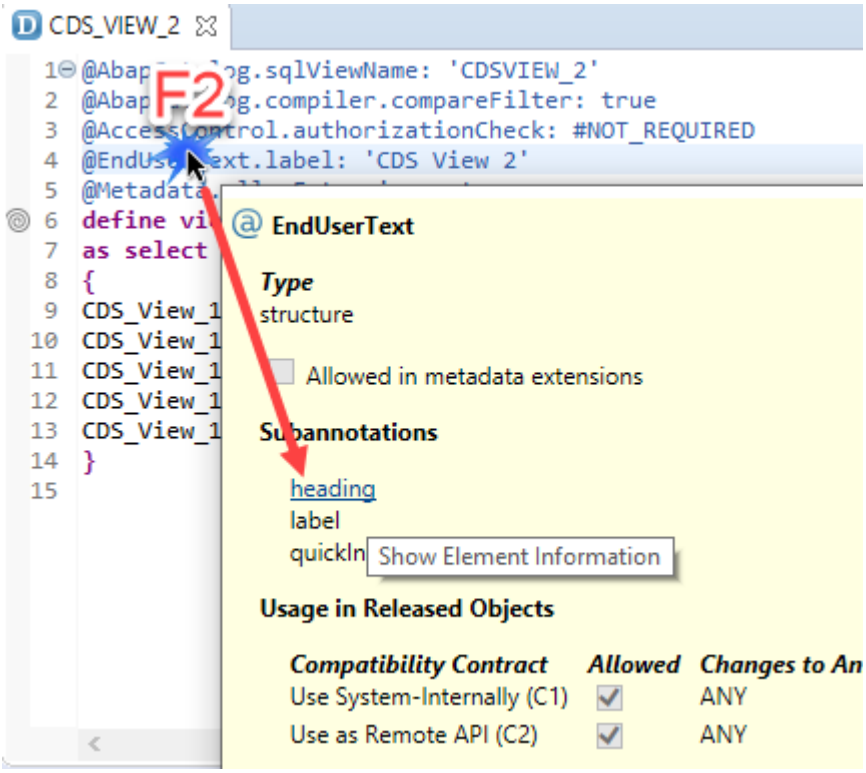
The *Element Info popup* now also displays the

- name of the related main annotation,
- related subannotation(s),
- enumeration values of CDS annotations, and
- technical details.



Sample for displaying technical details

In addition, you can navigate from the main annotation to the element information of the selected subannotation using the *Element Info* popup and vice versa.



Sample for navigating from a main annotation to a subannotation using the Element Info popup

These features support you while developing a data model.

They are available in the source code editors for ...

- data definitions (DDLS),
- access controls (DCLS),
- metadata extensions (DDLX), and
- service definitions (SRVD).

i For more information, see

Using Wildcards When Performing Code Completion for CDS Annotations

You can now use wildcards like the asterisk (*) to limit the list of relevant entries if you do not know the qualified name of the annotation when performing code completion for access controls, service definitions, and metadata extensions.

i For more information, see Code Completion [page 46]

5.3 Version 3.6

Here is an overview of the most significant changes in the context of ABAP CDS development that relate to the following:

- Client: ABAP Development Tools (ADT) **3.6**
- Back end version: ☁ **SAP Cloud Platform ABAP Environment 1911**

> **i** Note
>
> The following features that are highlighted with a '*' are client-specific and are therefore available for all supported ABAP systems.

In this topic, you can find release information about:

- Extending Data Models Using Metadata Extensions [page 115]
- Tuning Access to SAP HANA [page 115]
- Using Wildcards for Code Completion in the CDS Source Code Editor [page 116]

☁ Extending Data Models

Extending the Metadata of Data Models

You can now also use metadata extensions in the context of ABAP environment.

Metadata extenstions are development objects that are saved in the ABAP Repository and can be transported within your ABAP system landscape.

You specify metadata extenstions to extend and to adapt the behavior of CDS entities with customer-specific metadata using CDS annotations.

This enables you to specify your own annotations in metadata extensions to overwrite metadata used in data definitions without causing modifications. When creating metadata extensions, you can use a set of predefined code templates.

i For more information, see

- Metadata Extensions [page 30]
- Extending the Metadata of Data Models [page 85]

☁ Tuning Access to SAP HANA

You can now temporarily store the result of one or more aggregations (for example, `sum`, `min`, `max`, `avg`) in a dynamic cache. This avoids repetitive calculations of frequently executed queries, and improves performance.

> **i** Note
>
> To view and manage the dynamic cache configurations, you can use the `Manage Database Cache Configuration` app.

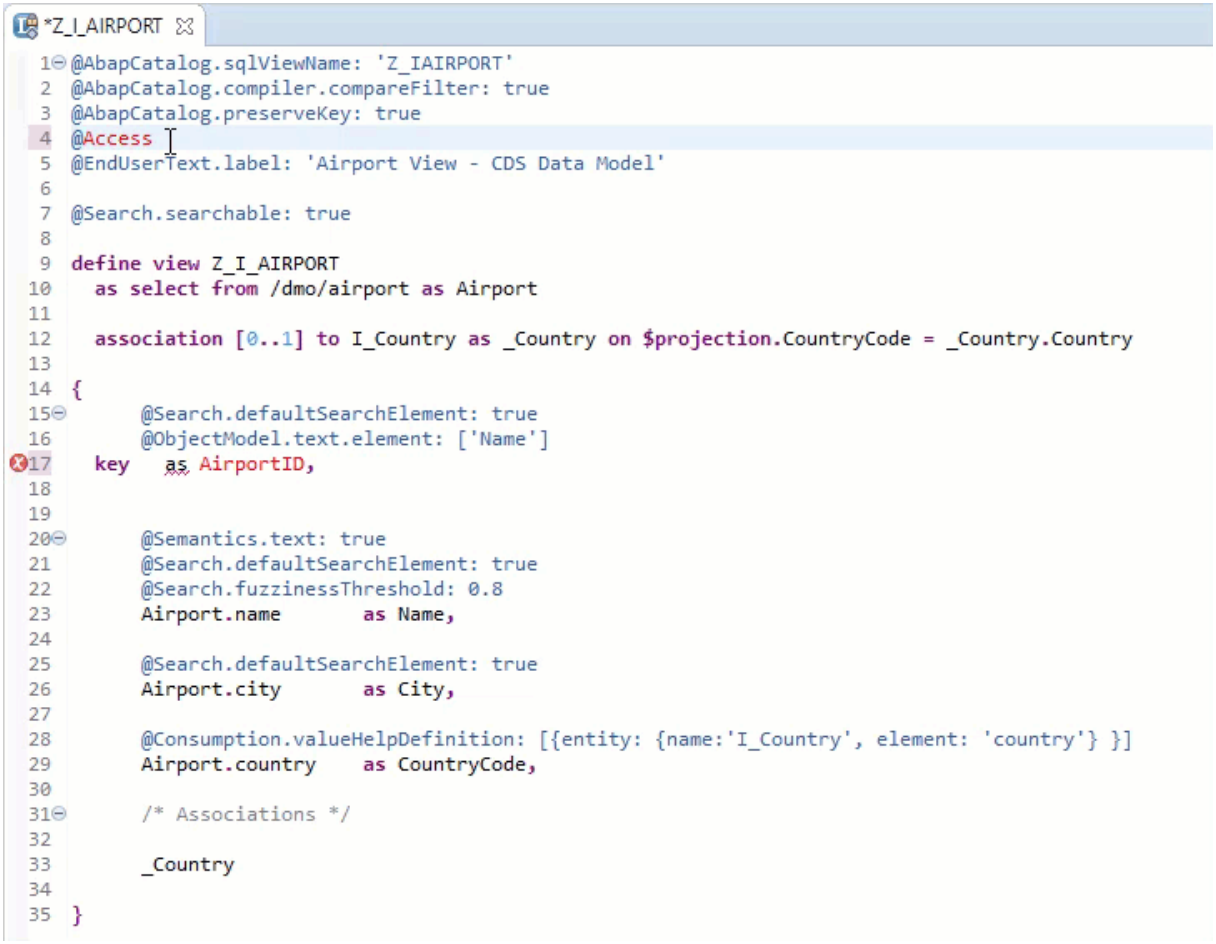
i For more information, see Creating Dynamic Caches [page 94]

Editing DDL Source Code

Using Wildcards for Code Completion in the CDS Source Code Editor

You can now use wildcards like the asterisk (*) to limit the list of relevant entries if you do not know the qualified name of the identifier.

To benefit from this feature, start typing a character in the CDS source code editor and add the asterisk to complete your search string. Finally, trigger code completion (`Ctrl` + `Space`) and takeover the relevant proposal from the code completion list. The selected finding will then be added.



Using code completion in the definition and implementation of CDS entities

> **Note**
>
> Using the asterisk at the leading position is not supported.
>
> If you use several segments within a identifier path, only the asterisk in the last segment is supported.

For more information, see Code Completion [page 46]

5.4 Version 3.0

Here is an overview of the most significant changes in the context of ABAP CDS development that relate to the following:

- Client: ABAP Development Tools (ADT) **3.0**
- Back end version: **SAP Cloud Platform ABAP Environment 1902**

> **Note**
>
> The following features that are highlighted with a '*' are client-specific and are therefore available for all supported ABAP systems.

Ensuring Quality with ABAP Unit

You can now use ABAP CDS Test Double Framework and ABAP SQL Test Double Framework for CDS entities and ABAP classes respectively. These frameworks allow you to replace data sources with test doubles in an ABAP Unit test so that the object under test accesses the test double instead of the actual component.

For more information, see

-
-
-

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.

www.sap.com/contactsap

© 2020 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP